Visualisierung der
abstrakten Programmausführung
Dierk Johannes

Aufbereitung von Shapeanalyseausgaben zur Visualisierung der abstrakten Programmausführung

Dierk Johannes

Dissertation zur Erlangung des Grades des
Doktors der Ingenieurwissenschaften
der Naturwissenschaftlich-Technischen Fakultäten
der Universität des Saarlandes

Saarbrücken, 2009

Datum des Kolloquiums:	20. April 2010
Dekan der Naturwissenschaftlich-Technischen Fakultät I:	Prof. Dr.-Ing. Holger Hermanns
Mitglieder des Prüfungsausschusses	
Vorsitzender:	Prof. Dr.-Ing. Philipp Slusallek
Berichterstattende:	Prof. Dr. Raimund Seidel
	Prof. Dr. Reinhard Wilhelm
Akademischer Mitarbeiter:	Dr.-Ing. Philipp Lucas

Zusammenfassungen

Diese Arbeit behandelt einen Ansatz zur Algorithmenvisualisierung von zeigerbasierten Programmen. Während traditionell die Programmausführung für konkrete Daten visualisiert wird, stützt sich dieser Ansatz auf die abstrakte Programmausführung. Dazu wird vorab mittels Shapeanalyse, einer auf Logik basierenden parametrischen statischen Programmanalysetechnik, eine Beschreibung der an den Programmpunkten auftretenden abstrakten Heapstrukturen berechnet. Diese Shapegraphenmengen sind jedoch in der Regel für eine direkte Visualisierung zu groß. Ein zentrales Thema dieser Arbeit ist die Entwicklung von Methoden, die Analyseausgabe vor der eigentlichen Visualisierung aufzubereiten. Sie führen sowohl zu einer Reduktion der Komplexität als auch zu einer gesteigerten Wirksamkeit der Visualisierung. Im Vordergrund stehen Methoden zur Strukturierung der Analyseausgabe. Ein Ähnlichkeitskonzept gestattet es, bezüglich verschiedener parametrischer Ähnlichkeitsbegriffe zu abstrahieren. Auf diese Weise werden ähnliche Heapstrukturen und ähnliche Programmausführungen identifiziert und zusammengefasst. Mit gleicher Absicht wird ein Konzept zur Ausnutzung von Symmetrie eingeführt. Ergänzend werden Methoden vorgestellt, welche die in einer Menge von Shapegraphen enthaltenen Informationen verdichten. Zu den darüber hinaus behandelten Themen gehören Methoden, die Hilfestellungen bei der Traversierung durch die abstrakte Programmausführung bieten.

*

This work presents an approach to algorithm visualisation of pointer based programs. While traditionally the execution of a program is visualised for concrete data, our approach is based on abstract program execution. Using shape analysis, which is a logic based parametric static program analysis technique, a description of the abstract heap situations that can occur at each program point is computed in advance. However, the resulting sets of shape graphs are generally too large to be visualised directly. A central topic of this work is the development of methods for preparing the analysis output before the actual visualisation. This results in complexity reduction as well as in increased efficiency with respect to visualisation. The focus lies on methods for structuring the analysis output. A similarity concept allows abstraction with respect to various parametric similarity notions. This way similar heap structures and similar execution paths are identified and summarised. With the same purpose in mind, a concept of taking advantage of symmetry is introduced. Additionally, methods are presented that condense the information contained in sets of shape graphs. Further topics include methods that assist in the traversal of the abstract program execution.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

> Ich höre und vergesse. Ich sehe und erinnere. Ich tue und verstehe.
>
> *(Konfuzius)*

Diese Arbeit beschäftigt sich mit der Visualisierung von Algorithmen beziehungsweise Programmen. Den Begriff Visualisierung kann man ganz allgemein als bildliche Formulierung verstehen, als Aufbereitung von Informationen für die visuelle Wahrnehmung. Von den menschlichen Sinnen ist der Sehsinn derjenige, den die meisten Menschen als den primären menschlichen, als ihren primären Sinn ansehen werden. Er ist auch derjenige, der mit Abstand die meisten Informationen pro Zeiteinheit aufnehmen kann. Eine Form von Informationsaufbereitung, die den Sehsinn anspricht, bietet daher eine gute Grundlage, so dass die Informationen von uns gut aufgenommen und verarbeitet werden können. Im Zuge der fortschreitenden Entwicklung der Leistungsfähigkeit von Computern und den daraus resultierenden Möglichkeiten hat sich auch die Informatik dem Themenbereich Visualisierung zugewandt. Anfangs waren primär (mehrdimensionale) Daten der Gegenstand, den es zu visualisieren galt. Später wurde er auf andere Objekte ausgeweitet. Einer der Gegenstände der Visualisierung sind Algorithmen.

Bei den Algorithmen – fürs Erste wollen wir nicht zwischen Algorithmus und Programm unterscheiden und die Worte als Synonyme ansehen – geht es uns konkret um imperative, zeigerbasierte Programme. Die verwendeten Daten beziehungsweise Datenstrukturen werden also im Heap gehalten. Zugriffe auf die Daten erfolgen über Zeiger, und Änderungen an der Struktur der Daten im Heap bedeuten eine Manipulation von Zeigern (beziehungsweise Zeigervariablen). Zeiger sind problematische Objekte. Von einem theoretischen Standpunkt aus sind sie unangenehm, weil Zeiger semantisch zu den am schwierigsten erfassbaren Programmiersprachenkonstrukte gehören. Von einem praktischen Standpunkt aus sind sie unangenehm, weil die Verwendung von Zeigern leicht und häufig zu Fehlern führt; typisch sind die Dereferenzierung von NULL-Zeigern und der Zugriff auf bereits freigegebene Heapbereiche. Demgegenüber bietet die Verwendung von Zeigern nicht zu vernachlässigende Effizienzvorteile, was sie für die Praxis überaus bedeutsam macht. So sind Veränderungen an der Datenstruktur nicht nur möglich, sie erfordern häufig auch nur lokale(re) Änderungen. Daher bieten zeigerbasierte Implementierungen ein hohes Maß an Dynamik und führen zu speicher- und zeiteffizienten Implementierungen. Aus diesen Gründen zeichnen sich zeigerbasierte Algorithmen als interessante

und wichtige Klasse aus, die auch in Bezug auf Visualisierung ein eigenständiges Interesse verdient.

Die konkreten Beweggründe, warum sich jemand Algorithmenvisualisierung zunutze machen möchte, können im Einzelfall recht verschieden sein. Unter anderem sind folgende Anwendungssituationen möglich: Ein Anwender kann als Lernender den Algorithmus verstehen wollen. Ein Anwender kann sich als Programmierer von der Abwesenheit gröberer Fehler überzeugen lassen wollen. Ein Anwender kann als Programmierer an einer Verbesserung des Programmcodes interessiert sein und nach ungeschickt programmierten Codeteilen Ausschau halten. All diesen Anwendungsmotiven ist gemeinsam, dass mit Hilfe der Visualisierung ein Zuwachs an Erkenntnis über den Algorithmus beziehungsweise über das Programm angestrebt wird. Daher wollen wir den Umgang mit Algorithmenvisualisierung ganz allgemein als Lernsituationen interpretieren. Dementsprechend wollen wir Algorithmenvisualisierung als ein Mittel zur Unterstützung von Lernvorgängen auffassen.

Für die folgende Explikation bedienen wir uns eines konkreten Beispieles. Als zugrunde liegende Datenstruktur betrachten wir einen binären Suchbaum, den wir uns gerne auch balanciert vorstellen können. (Der Datentyp Baumelement besitze einen ausgezeichneten Datenwert, einen Schlüssel, bezüglich dem die Sortierung erfolgt.) Als Beispielalgorithmus betrachten wir das Einfügen eines Elements in einen solchen Baum. Der erste Schritt eines solchen Algorithmus ist das Suchen des einzufügenden Elements im Baum, also die Suche nach einem Element im Baum mit gleichem Schlüssel. Wenn ein solches im Baum vorhanden ist, dann werden wir das neue Element gewöhnlich nicht einfügen wollen, üblicherweise erlaubt man keine Elemente mit gleichem Schlüssel, und werden den Algorithmus mit einer Fehlermeldung beenden. Im anderen Fall haben wir mit der Suche die Position im Baum gefunden, an der das neue Element einzufügen ist, so dass der resultierende Baum ebenfalls wieder ein Suchbaum ist. (Nach dem Einfügen muss man sich gegebenenfalls noch darum kümmern, dass die Balancierung wiederhergestellt wird.) Im Folgenden verbleiben wir der Einfachheit halber beim Suchalgorithmus.

Wir haben oben Visualisierung ganz allgemein als Aufbereitung von Informationen für die visuelle Wahrnehmung charakterisiert. Folglich ist Algorithmenvisualisierung die Aufbereitung von Algorithmen (beziehungsweise Programmen) für die visuelle Wahrnehmung. Streng genommen fällt hierunter auch schon ein Ausdruck des Programmcodes. Dies rechnet man jedoch noch nicht zur Algorithmenvisualisierung. Üblicherweise verwendet man den Begriff nur für visuelle Aufbereitungen, die deutlich über eine bloße Präsentation des Codes hinausgehen.

Der übliche Ansatz bei der Algorithmenvisualisierung besteht darin, die Ausführung eines Programms anhand konkreter Daten zu visualisieren. Diesen Ansatz bezeichnen wir im Folgenden auch als traditionelle Algorithmenvisualisierung. Dabei wird die im Heap befindliche Datenstruktur, jeweils als Momentaufnahme, in einer geeigneten Weise dargestellt. Die Visualisierung besteht dann aus einer oder mehreren Sequenzen solcher Darstellungen. Jedes Paar aufeinanderfolgender Darstellungen

einer Sequenz zeigt die Heapsituation vor und nach Ausführung einer Programmaktion. Die Reihenfolge der Aktionen, genauer der Kantenzug durch den Kontrollflussgraphen, ist dabei neben dem Programm implizit durch den im Voraus festgelegten konkreten Eingabedatensatz bestimmt. Ein klassischer Vertreter dieses Ansatzes ist das Softwarepaket TANGO und dessen Nachfolger POLKA, vergleiche [TANGO 1990], für weitere Referenzen und ein Überblick über verschiedene Entwicklungstendenzen kann [Kerren u. Stasko 2002] konsultiert werden. Ein Anwender dieser Art von Algorithmenvisualisierung entwickelt sein Verständnis über das Programm beziehungsweise den Algorithmus aus dem Verhalten des Programms bezüglich der betrachteten Datensätze. Diese Form des Lernens, die vom Konkreten oder Speziellen auf das Allgemeine schließt, heißt induktives Lernen.

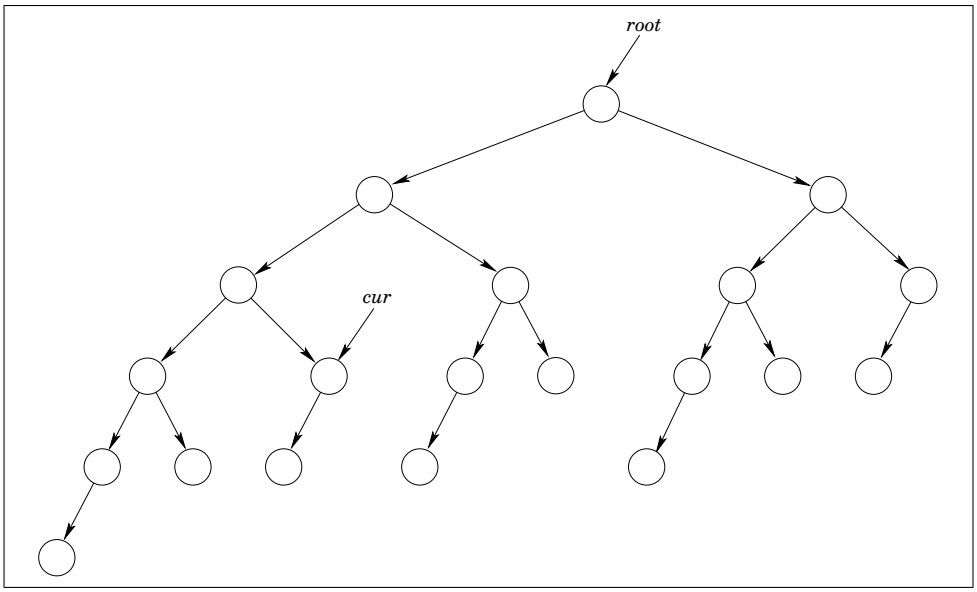

Abbildung 1.1: Konkrete Darstellung eines Suchbaums

In unserem Beispiel handelt es sich bei der Datenstruktur um einen Baum. Er wird üblicherweise in für Informatiker vertrauter Baumgestalt präsentiert. Dem Betrachter werden Darstellungen angeboten, die derjenigen in Abbildung 1.1 ähneln oder entsprechen. (Auf die Angabe konkreter Datenwerte wurde verzichtet.) Normalerweise erfüllen sie folgende Charakteristiken: Der Baum wird als Ganzes dargestellt, das heißt jede Heapzelle ist (im Prinzip) in der Darstellung vorhanden. Seine Ecken sind so angeordnet, dass ihre Aufzählung entlang der x-Achse eine Sortierung der Datensätze (nach dem Schlüssel) ergibt. Einzelne Baumelemente sind ausgezeichnet; hierzu gehören die Ecken, auf die Zeigervariablen zeigen, also in unserem Fall die Wurzel des Baums und das aktuelle Element, welches als Vergleichselement dient.

Eine wesentliche Charakteristik der traditionellen Algorithmenvisualisierung, der

Visualisierung der Programmausführung mit konkreten Daten, ist ihr Fokus auf Veränderungen. Nach der Ausführung einer Programmaktion werden unmittelbar die neue Heapsituation und die neuen Werte der Zeigervariablen präsentiert, also die Auswirkung der Aktion. Wenn wir von Paaren aufeinanderfolgender Darstellungen, also von einzelnen Programmaktionen, zu Sequenzen von Aktionen und damit auch zu ganzen Programmen übergehen, dann nehmen wir vornehmlich einen sich durch die Programmaktionen verändernden Heap wahr. Auch wenn wir Visualisierungen mit verschiedenen Datensätzen durchführen, konzentriert sich die Aufmerksamkeit des Betrachters auf die Unterschiede zwischen den Darstellungen.

In dieser Fokussierung auf Veränderungen ist eine grundlegende Beschränkung des traditionellen Ansatzes zur Algorithmenvisualisierung begründet. Sie impliziert, dass Unveränderliches, also Gemeinsamkeiten und Ähnliches, dass Invarianten nicht explizit herausgearbeitet, nicht explizit visualisiert werden können. Gemeinsamkeiten zwischen zwei aufeinanderfolgenden Darstellungen, sofern sie existieren, sind in der Visualisierung natürlich implizit vorhanden. Der Betrachter muss sie sich aber selbst erarbeiten. Diese Tatsache gilt auch für die Gemeinsamkeiten, die zwischen Programmausführungen mit verschiedenen Datensätzen bestehen.

Prinzipiell gibt es unendlich viele Datensätze, die ein Algorithmus als Eingabe erhalten kann. Beim Einfügealgorithmus kann die Eingabe variieren in der Größe des Baumes, in seiner (Zeiger-)Struktur und in den konkreten Werten der Datenelemente. Bei der traditionellen Algorithmenvisualisierung wird jede dieser Möglichkeiten als eigenständige Instanz behandelt. Wenn wir beispielsweise zwei Bäume betrachten, die dieselbe Zeigerstruktur haben und sich nur in Datenwerten unterscheiden, dann werden die Abarbeitungen des Einfügealgorithmus gleich verlaufen, die den Abarbeitungen korrespondierenden Kantenzüge im Kontrollflussgraphen sind gleich. Auch die im Verlauf der Abarbeitungen entstehenden Heapsituationen sind ihrer Struktur nach paarweise gleich. Des Weiteren gibt es normalerweise viele Eingabeinstanzen, die (in einer gewissen Weise) zueinander ähnlich sind und zu einer ähnlichen Abarbeitung führen. Wir können uns zum Beispiel vorstellen, dass der eine Baum tiefer als der andere ist und dass sich dadurch die Länge des Suchweges vergrößert, was eine höhere Anzahl an Schleifeniterationen impliziert. Wenn aber Abarbeitungen ähnlich sind, dann stellen sie im Grunde genommen nichts wesentlich Verschiedenes (bezüglich dieses Ähnlichkeitsbegriffs) dar.

Wir interpretieren den Umgang mit Algorithmenvisualisierung als Lernsituation. Um ein größeres Verständnis über einen Algorithmus zu erlangen, ist es wenig förderlich, sich Dutzende von Visualisierungen zeigen zu lassen, die alle gleich oder ähnlich zueinander sind. Hierin liegt ein großes Maß an Redundanz und sogar Ineffizienz. Man wird ganz im Gegensatz dazu lieber möglichst viele wesentlich verschiedene Abarbeitungen visualisiert bekommen wollen. Für einen vollständigen Überblick über den Algorithmus ist dies in jedem Fall unerlässlich. Ob gleiche oder ähnliche Abarbeitungen entstehen, ist durch die Eingabeinstanzen bedingt. Aber gerade

ein Anwender mit wenig Vorkenntnissen über den Algorithmus wird das schwerlich im Vorfeld beurteilen können. Der Qualität der (mitgelieferten) Eingabeinstanzen kommt bei der traditionellen Algorithmenvisualisierung damit eine nicht zu unterschätzende Bedeutung zu.

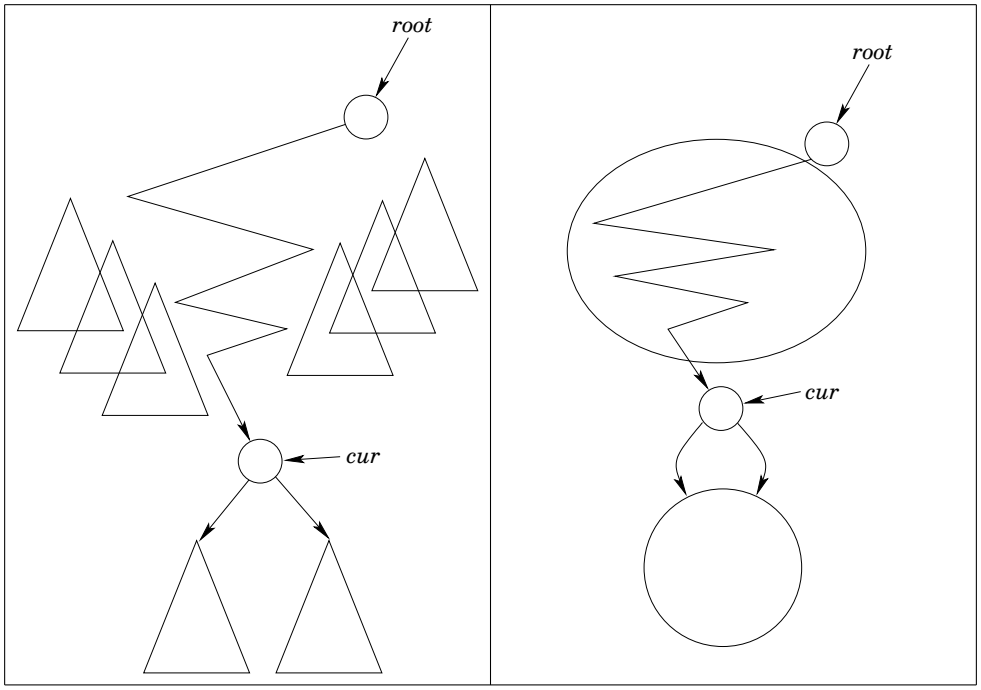

Abbildung 1.2: Abstrahierte Darstellungen eines Suchbaums

Wenn man den Einfügealgorithmus von einem Lehrer vorgeführt bekommt, dann wird man andere Darstellungen zu sehen bekommen als diejenigen, die Abbildung 1.1 ähneln. In einer realen Lernsituation wird ein Lehrer den Blick auf wichtige und wesentliche Bereiche der Datenstruktur lenken und diese herausheben, die übrigen Bereiche werden in den Hintergrund gedrängt. Auf diese Weise wird die Aufmerksamkeit des Lernenden gelenkt. Als Resultat ergeben sich Darstellung, die deutlich abstrakter ausfallen: Große Bereiche des Heaps werden zu Einheiten zusammengefasst sein. Dabei ist es für die Lernsituation normalerweise förderlich, wenn die Darstellung des abstrahierten Heaps nicht zu stark vom Gewohnten abweicht. Abbildung 1.2 zeigt zwei Beispiele.

Zugriffe auf die Datenstruktur im Heap finden über Zeigervariablen statt. Daher sind nur diejenigen Heapzellen von der aktuellen Programmaktion betroffen, auf die Variablen zeigen, gegebenenfalls noch deren Nachfolger. In unserem Beispiel sind dies die Wurzel, sie dient als Eingang zum Baum, und das aktuelle Baumelement, das als Vergleichselement dient. Auf diesen Heapzellen liegt der Fokus. (Bei einzelnen Programmaktionen oder kürzeren Sequenzen dürften oft sogar nur einige von diesen Heapzellen betroffen sein, der Fokus liegt also im Grunde genommen nur auf

einer Teilmenge dieser Zellen.) Da die anderen Heapelemente nicht (direkt) von der Aktion betroffen sind, können sie zusammengefasst dargestellt werden. Man wird sie üblicherweise in Gruppen einteilen, so dass sich Heapzellen, die eine gemeinsame Charakteristik hinsichtlich des Algorithmus aufweisen, in derselben Gruppe befinden.

Betrachten wir zur Verdeutlichung das Suchbeispiel. Dabei nehmen wir an, dass das aktuelle (Vergleichs-)Element kein Blatt des Baumes sei. Die Heapzellen, die zu den Teilbäumen gehören, die am aktuellen Element herabhängen, unterscheiden sich von allen anderen dadurch, dass sie den Teil des Baumes beschreiben, in dem die Suche fortgesetzt werden kann, der noch zu explorieren ist. Gegebenenfalls ist es angebracht, zwischen dem linken und rechten Teilbaum zu differenzieren. Alle anderen Baumteile wurden entweder bereits besucht oder sie wurden als nicht examinationswürdig erkannt. Bei diesen Heapzellen kann gegebenenfalls zwischen den besuchten und nicht besuchten Baumelementen differenziert werden. Erstere formen einen Weg von der Wurzel zum aktuellen Element, letztere gehören zu Teilbäumen, die von diesem Weg herabhängen. In Abbildung 1.2 sehen wir zwei Beispieldarstellungen. In der linken Darstellung wurde zwischen dem linken und rechten Teilbaum, die am aktuellen Element herabhängen, unterschieden. (Teilbäume sind durch Dreiecke dargestellt.) Auch wurde zwischen dem Suchweg und den an ihm herabhängenden Teilbäumen unterschieden. In der rechten Darstellung wurden beide Unterscheidungen nicht durchgeführt.

Dieser Ansatz zum Lernen kontrastiert das induktive Lernen der traditionellen Algorithmenvisualisierung, welches vom Einzelnen auf das Allgemeine schließt. Hier werden, ganz allgemein gesprochen, Begriffe verwendet, die für eine allgemeine, abstrakte Situation stehen. Wie das Baumbeispiel verdeutlicht, nehmen die Begriffe im Umfeld der Algorithmenvisualisierung einen visuellen Charakter an, weswegen wir auch von abstrahierten oder abstrakten Darstellungen sprechen. Ein Begriff ist einerseits durch charakterisierende Eigenschaften determiniert, andererseits umfasst er eine Menge von konkreten Objekten, die diese gemeinsame Charakteristik aufweisen. Ausgehend von einem Verständnis der Begriffe, der abstrahierten Situationen, kann in der Regel leicht auf die unter den Begriff fallenden konkreten Situationen geschlossen werden. Wir haben es mit einer Lernform zu tun, die vom Allgemeinen auf das Spezielle oder Einzelne schließt, sie heißt deduktives Lernen.

In dieser Arbeit behandeln wir einen Ansatz zur Algorithmenvisualisierung, der deduktives Lernen ermöglicht und unterstützt. Anstelle von Darstellungen von konkreten Heapsituationen verwenden wir Darstellungen von abstrakten Heapsituationen, anstelle einer konkreten Programmausführung visualisieren wir eine abstrakte Programmausführung. Zu diesem Zweck benötigen wir als Basis einen Mechanismus, um Heapabstraktionen und Darstellungen von ihnen herstellen zu können. Wir wollen uns dabei nicht mit einer beliebigen Methode zufriedengeben. Stattdessen wünschen wir, dass sie eine Reihe von Kriterien gut erfüllt: 1. Damit die Visualisierung den

aktuellen (Lern-)Zielen angepasst werden kann, erwarten wir Einflussmöglichkeiten auf die abstrahierten Darstellungen. Es muss einen Beschreibungsmechanismus geben, der entweder die Art der Abstraktion oder das gewünschte Aussehen der resultierenden Darstellungen bestimmt. (Beides hängt eng zusammen: Die Abstraktion steuert das Aussehen der Darstellungen.) 2. Die Berechnung der abstrakten Heapsituationen soll automatisch auf der Grundlage einer Spezifikation geschehen. (Die Terminierung der Berechnung impliziert auch, dass die Anzahl der abstrakten Heapsituationen und deren Größe endlich ist.) 3. Der Prozess der Berechnung der abstrakten Heapsituationen soll eine theoretische Fundierung besitzen und die abstrakten Heapsituationen sollen eine klare semantische Bedeutung haben. 4. Die berechneten abstrakten Heapsituationen sollen sich für eine Visualisierung eignen, das heißt aus ihnen sollen sich Darstellungen ableiten lassen, die eine hohe erklärerische Qualität aufweisen.

Eine Methode, welche diese Kriterien zu unserer vollen Zufriedenheit erfüllt, ist Shapeanalyse, weswegen wir sie für unsere Zwecke verwenden. Dabei handelt es sich um einen der zurzeit mächtigsten Mechanismen zur statischen Programmanalyse. Sie basiert auf dreiwertiger Logik. Zuzüglich zum zu analysierenden Programm erhält sie als Eingabe eine logische Sprache, bezüglich derer die Analyse erfolgt. Heapsituationen, die sich hinsichtlich spezifizierter Eigenschaften nicht unterscheiden, werden zusammengefasst. Die so berechneten abstrakten Heapzustände sind logische Strukturen und heißen Shapegraphen. Die Analysemethode stellt sicher, dass nur endlich viele von ihnen existieren (können). Alle konkreten Heapsituationen werden auf eine endliche Menge von Shapegraphen abgebildet. Den konkreten Programmaktionen werden Programmaktionen gegenübergestellt, die auf abstrakten Heapsituationen operieren. Jede Ausführung einer Programmaktion auf einer konkreten Heapsituation wird auf eine Transition zwischen Shapegraphen abgebildet. Damit wird eine konkrete Programmausführung auf eine abstrakte abgebildet.

Durch eine Abstraktion entsteht naturgemäß ein Informationsverlust, was sowohl Vorteil wie Nachteil sein kann. Da Details wegabstrahiert werden, liegt der Fokus auf dem, was sonst gleich und beständig bleibt. Die Menge der Shapegraphen eines Programmpunktes beschreibt alle an diesem Programmpunkt auftretenden Heapsituationen bezüglich der zugrunde liegenden Spezifikation. Damit handelt es sich bei ihnen um Invarianten, und ihr explizites Vorliegen wird zumindest ein Algorithmentheoretiker stets begrüßen. Sollten durch die Abstraktion Details, die man bei der Visualisierung zu betrachten wünscht, verloren gegangen sein, kann man eine neue Shapeanalyse mit veränderter, feinerer Spezifikation durchführen. Im Vergleich zur traditionellen Algorithmenvisualisierung benötigt die Shapeanalyse eine zusätzliche Spezifikation, was einen Mehraufwand bedeutet. Es ist jedoch zu beachten, dass die Spezifikation der Datenstruktur für jede Datenstruktur nur einmal vorgenommen werden braucht. Die Spezifizierung des Algorithmus verhält sich wie seine Kodierung für eine konkrete Ausführung. Auf der anderen Seite spart man die Zusammenstellung einer Menge von guten Eingabeinstanzen ein.

Wir benutzen die Shapeanalyseausgabe als Basis für die Visualisierung. Bei den Shapegraphen handelt es sich zunächst einmal um logische Strukturen, für die Visualisierung benötigen wir visuelle Darstellungen. Shapegraphen erlauben eine relativ intuitive Darstellung als beschrifteter gerichteter Graph, so dass hier keine zusätzliche Aufgabe eingeführt wird. Ein zentrales Problem liegt jedoch in der Größe der Analyseausgabe. Wenn man zum Beispiel nur eine geringe oder mäßige Abstraktion wünscht, dann wird man nicht ernsthaft erwarten können, dass sich die potenziell unendlich vielen konkreten Heapsituationen auf nur einige wenige abstrakte reduzieren lassen. Die Shapeanalyse muss sie alle berücksichtigen, um die Korrektheit der Analyse und des Ergebnisses zu garantieren. Die entstehende Vielfalt und der Detailreichtum sind aber für die darauf aufsetzende Visualisierung nicht immer vonnöten. Auch kann die Shapeanalyseausgabe Redundanz enthalten. Aus praktischer Sicht zeigt sich bei Verwendung „sinnvoller“ Abstraktionen, dass bei Algorithmen, die einfache Datenstrukturen wie einfach oder doppelt verkettete Listen verwenden, noch keine Probleme hinsichtlich der Ausgabegröße auftreten. Aber schon bei (binären) Bäumen wird das Problem akut.

Wir stehen in der Regel also der Situation gegenüberstehen, dass die Shapeanalyseausgabe zu groß ist, als dass sie direkt visualisiert werden sollte. Wohlgemerkt geht es hier nicht darum, ob oder inwieweit man (sehr) große Datenmengen visualisieren kann. Wir interpretieren den Umgang mit Algorithmenvisualisierung als Lernsituation. Die neurophysiologischen Gegebenheiten und Beschränkungen des menschlichen Gehirns bedingen, dass ein Anwender nur eine bestimmte Menge an Informationen, also auch nur eine bestimmte Komplexität der abstrakten Programmausführung, handhaben kann. Eine große Analyseausgabe stellt also ein Problem dar. Ein angestrebter Lernerfolg kann einfacher realisiert und effektiver gestaltet werden, wenn die Shapeanalyseausgabe vor der eigentlichen Visualisierung im Hinblick auf das Lernziel aufbereitet wird.

In dieser Arbeit elaborieren wir Methoden, die Shapeanalyseausgabe für eine Visualisierung aufzubereiten. Themen, die sich mit der konkreten Darstellung, also unter anderem dem Zeichnen, von Shapegraphen beziehungsweise von Sequenzen von Shapegraphen befassen, sind nicht Gegenstand dieser Arbeit. Wir sind also gewissermaßen zwischen der Shapeanalyse und der eigentlichen Visualisierung positioniert. Auf die Fragen, welche Shapegraphen einer Analyseausgabe für eine Visualisierung besonders wichtig sind und welche nicht, welche sich besonders gut eignen und welche weniger oder welche eine besonders signifikante Aussage enthalten und welche nicht, wird man wohl kaum eine allgemeingültige Antwort finden können. Eine solche Beurteilung hängt nicht zuletzt von der konkreten Zielsetzung einer Visualisierungssitzung ab. Wir verfolgen einen anderen Zugang. Es werden diverse Ähnlichkeitskonzepte eingeführt und untersucht. Diese erlauben uns, den Grad von Ähnlichkeit und Verschiedenheit sowohl von Shapegraphen als auch von abstrakten Ausführungen genauer beurteilen zu können. Diese Methode unterstützt uns zum Beispiel auch bei der Beurteilung, ob Shapegraphen beziehungsweise abstrakte Programmausführungen allgemeinere Fälle oder speziellere Fälle beschrei-

ben. Aufbauend auf diesen Klassifikationen versehen wir die Shapeanalyseausgabe mit zusätzlicher Struktur. Dies resultiert in einer deutlichen Erleichterung bei der Navigation durch die abstrakte Programmausführung. Insgesamt kann auf diese Weise der durch die Visualisierung unterstützte Lernprozess effektiver gestaltet werden.

Auch bei der klassischen Visualisierung war man bemüht, von konkreten Darstellungen der Datenstruktur zu abstrahierteren Darstellungen überzugehen. Dem Thema Abstraktion hat man sich auch systematisch genähert, in [Cox u. Roman 1992] werden beispielsweise Typen von Abstraktionen klassifiziert. Die Idee, Abstraktionen von Heapsituationen für eine Visualisierung zu nutzen, wurde aber bisher kaum verfolgt. Der einzige uns bekannte Ansatz ist Michails visuelles Programmiersystem OPSIS, die Projektseite ist [OPSIS 1996], das zum Zwecke des Unterrichtens binärer Bäume entwickelt wurde, vergleiche [Michail 1996]. Auch dort werden Baumelemente, die bestimmte Eigenschaften teilen, zusammengefasst. Jedoch ermangelt sein Ansatz einer semantischen Fundierung. Das führt beispielsweise dazu, dass Schleifeninvarianten vom Programmierer gefunden und formuliert werden müssen. In [Johannes u. a. 2005, Abschnitt 6] wird sein Ansatz mit Bezug auf den hier präsentierten ausführlicher diskutiert.

2 Lernen

> Tantum scimus, quantum memoria tenemus. (Wir wissen soviel, wie wir im Gedächtnis haben.)
>
> *(Römisches Sprichwort)*

Visualisierung kann ganz allgemein als bildliche Formulierung, als Aufbereitung von Informationen für die menschliche Wahrnehmung verstanden werden. Damit rücken die physiologischen Gegebenheiten der menschlichen visuellen Wahrnehmung und die psychologischen Aspekte der Informationsaufnahme und -verarbeitung in den Blickpunkt des sich mit Visualisierung beschäftigenden Informatikers. Je besser diese Aspekte bei der Visualisierung berücksichtigt werden, desto effektiver kann sie gestaltet werden. Jedoch scheint es, dass Informatiker diese Aspekte eher zögerlich in ihre Arbeit integrieren. In neueren Büchern über Visualisierung, [Schumann u. Müller 2000] sei als Beispiel angeführt, wird ihnen aber schon ein beachtenswerter Teil des Platzes zugestanden. Allerdings werden gewöhnlich lediglich die physiologischen Aspekte des menschlichen Sehsinns behandelt.

Diese Arbeit behandelt einen Ansatz zur Visualisierung von Algorithmen. Grundsätzlich dient Algorithmenvisualisierung zur Unterstützung und Hilfe beim Prozess, einen Algorithmen besser zu verstehen. Daher fassen wir Algorithmenvisualisierung in einem weiten Sinn als lernunterstützendes Mittel auf. Aus diesem Grund soll auch den (psychologischen) Aspekten des Lernens Aufmerksamkeit zukommen, was in diesem Kapitel geschieht. Lernen ist ein komplexer Vorgang: Er beinhaltet Aspekte der Wahrnehmung, das heißt der Aufnahme von Informationen. Auch beinhaltet er ein Verarbeiten von Informationen, sie werden unter anderem in das Beziehungsgeflecht des Bekannten eingeordnet. Dann beinhaltet Lernen auch den Aspekt des Erinnerns, des Abrufens eingespeicherter Informationen. Er beinhaltet mentale Prozesse wie Erkennen und Verstehen sowohl von Sachverhalten als auch von Beziehungen zwischen ihnen. Schließlich beinhaltet er das Speichern von Informationen, damit auf sie in der Zukunft zurückgegriffen werden kann.

2.1 Gedächtnis und Lernen

Manfred Spitzer beginnt sein Buch über das Lernen [Spitzer 2003] mit den Worten: „Wenn es etwas gibt, was Menschen vor allen anderen Lebewesen auszeichnet, dann ist es die Tatsache, dass wir lernen können und dies auch zeitlebens tun.“ Damit

wird Lernen als ein wesentlicher Aspekt der menschlichen Existenz, als eine wichtige Fähigkeit des Gedächtnisses ausgezeichnet. Es kann hier natürlich kein dem Thema gerecht werdender Überblick präsentiert werden, die Auswahl der behandelten Aspekte erfolgte hinsichtlich ihres Zusammenhangs und ihrer Relevanz für unser Thema. Die Literatur zum Themenkomplex Gedächtnis und Lernen ist reichhaltig, ihre Schwerpunkte und Betrachtungswinkel hängen stark vom fachlichen Hintergrund der Autoren ab. Die hier dargestellten Zusammenhänge entstammen [Squire u. Kandel 1999], [Markowitsch 2002] und [Spitzer 2003].

Gedächtnis

Grundsätzlich ist es nützlich, zwei Ebenen zu unterscheiden. Die erste Ebene beschreibt die materiellen Aspekte. Man spricht vom *Gehirn*, wenn das biologische Organ gemeint ist. Die zweite Ebene beschreibt die immateriellen Aspekte, man spricht dann vom Geist und von geistigen oder *mentalen Prozessen.* Die Darstellung des Themenkomplexes Lernen und der dazugehörenden Vorgänge wie Wahrnehmen, Erkennen, Erinnern oder Vergessen erfolgt primär auf der mentalen Ebene. Die materielle Grundlage der mentalen Prozesse ist das Gehirn, daher versteht man unter dem Gehirn auch den Sitz der mentalen Prozesse.

Wegen dieses Zusammenhangs ist es plausibel zu glauben, dass zwischen Gehirn und Geist eine deutliche Verbindung besteht. Das haben auch viele Gehirnforscher der letzten Jahrhunderte so gesehen. Heutzutage ist diese Hypothese in der Gehirnforschung allgemein akzeptiert. Lediglich die Vorstellung, wie die Verbindung aussieht, ist Gegenstand der Diskussion. Das Spektrum der Ansichten reicht von der Vorstellung, dass sich geistige Vorgänge (zu einem gewissen Grad) im Gehirn „widerspiegeln“, bis zu der Ansicht, dass (alle) geistigen Prozesse Funktionen des Gehirns sind, sich also letztendlich vollständig aus der (biologischen) Funktion des Gehirns erklären lassen.

Das Gehirn und seine mentalen Prozesse können unter funktionalen Aspekten betrachtet werden. Derjenige Teil, der sich mit der Aufnahme von Informationen, ihrer Be- und Verarbeitung, ihrer Speicherung und ihrer Wiederabrufung befasst, ist das *Gedächtnis*. Somit ist Lernen eine Fähigkeit des Gedächtnisses. Allgemein gesprochen ist unser Gedächtnis die Instanz, die uns den Alltag gestalten lässt. Sie holt (unsere) Vergangenheit in die Gegenwart und verleiht ihr dadurch einen Kontext, eine Bedeutung. Sie ermöglicht ein an die Situation und an die Verhältnisse angepasstes Verhalten und Handeln und seine Ausrichtung auf die Zukunft. Ein Verlust des Gedächtnisses bedeutet häufig ein Verlust des Ichs, der Lebensgeschichte, und führt häufig zu einer dauerhaften Störung der Beziehungsfähigkeit zu Mitmenschen.

Die Inhalte, die wir im Gedächtnis speichern, stammen aus zwei Quellen. Zum einen handelt es sich um Informationen, die wir über die uns umgebene Welt und über uns aufnehmen. Primär sind das Sinneseindrücke, aber es kommen auch Informationen

vom peripheren Nervensystem (z. B. Schmerz- und Temperaturempfindungen), vom Gleichgewichtssystem sowie vom Zeitempfinden hinzu. Wir fassen diese zusammen und bezeichnen sie als Wahrnehmung. Der Verarbeitung von Wahrnehmungsinformationen im Gedächtnis kommt eine wichtige Bedeutung zu. Sie bestimmt, wie wir die Welt „sehen“. Zum anderen handelt es sich um eigene Gedanken und (Ergebnisse von) Überlegungen, die wir als neues Wissen speichern. Ihre Quelle ist das Gedächtnis selbst.

In den frühen Untersuchungen erscheint das Gedächtnis häufig als Einheit. Im Zuge der fortschreitenden Ausdifferenzierung anatomischer Strukturen im Gehirn erwies sich auch eine Differenzierung der darauf aufsetzenden geistigen Strukturen als sinnvoll und notwendig. Aufgrund medizinischer Erfahrungen wurde klar, dass das Gehirn und auch das Gedächtnis zwischen kurzfristigem und langfristigem Behalten unterscheidet. Deshalb wurden Modelle vorgeschlagen, die das Gedächtnis hinsichtlich des Kriteriums Zeit unterteilen. Das wohl einflussreichste dieser sogenannten Mehrspeichermodelle wurde 1968 von Richard. C. Atkinson und Richard Shiffrin vorgestellt. Es teilt das Gedächtnis in drei Komponenten ein: Ultrakurzzeitgedächtnis, Kurzzeitgedächtnis und Langzeitgedächtnis. Der Informationsfluss vollzieht sich in dieser Reihenfolge. Die Kapazitäten von Ultrakurzzeit- und Kurzzeitgedächtnis sind beschränkt und die Informationen haben nur eine kurze Speicherdauer, was insgesamt zu einer Filterung der eingehenden Information führt.

An dieser Art Modelle kann eine Reihe von Kritikpunkten gefunden werden. An der grundsätzlichen Einteilung des Gedächtnisses in ein Kurzzeit- und in ein Langzeitgedächtnis hält man heute fest. Das Kurzzeitgedächtnis sieht man aber umfassender. Man teilt es in zwei, zeitlich hintereinander liegende Teile: in das unmittelbare Gedächtnis, welches das Kurzzeitgedächtnis des Mehrspeichermodells ist, und in ein zeitlich daran anschließendes *Arbeitsgedächtnis*. Das Arbeitsgedächtnis kann man als die durch Wiederholung entstehende zeitliche Ausdehnung des unmittelbaren Gedächtnisses auffassen. Man hält das Arbeitsgedächtnis für grundlegend für Sprachverständnis, für Denken und für Lernen. Von funktionaler Seite gesehen, handelt es sich beim Arbeitsgedächtnis um den Teil des Gedächtnisses, der unsere Präsenz in der Gegenwart und damit unsere Aufmerksamkeit mit den passenden gespeicherten Informationen der Vergangenheit zusammenbringt und somit unser Verhalten und unser Handeln lenkt.

Damit Informationen behalten werden, müssen sie vom Arbeits- ins Langzeitgedächtnis gelangen, sie müssen eine dauerhafte Langzeitform annehmen. Die Herstellung der dauerhaften Form nennt man Konsolidierung und sie benötigt einige Zeit, wahrscheinlich vollzieht sich der Prozess auch in mehreren Stufen. Die Zeitspanne zwischen dem Verlassen des Arbeitsgedächtnisses und einer stabilen Verankerung im Langzeitgedächtnis ist eine Übergangszeit. Bis die Konsolidierungsphase vollständig abgeschlossen ist, sind die Informationen störanfällig. Wie lange die Konsolidierungsphase genau dauert, ist noch nicht vollständig geklärt, allerdings ist der

Großteil der Prozesse wohl nach mehreren Stunden abgeschlossen. Nach der Konsolidierung schließen sich Stabilisierungsprozesse an, die unter Umständen Jahre dauern können.

Heutzutage sieht man auch das Langzeitgedächtnis nicht mehr als Einheit an. Es wird nach dem Inhalt von Gedächtnisleistungen ausdifferenziert. Es herrscht weitgehend Einigkeit darüber, dass eine solche Gliederung möglich und sinnvoll ist. Über die genaue Einteilung wird noch diskutiert. Zunächst wird zwischen einem deklarativen und einem nichtdeklartiven Gedächtnis unterschieden. – Das *nichtdeklarative Gedächtnis* umfasst Systeme, die einer bewussten Erinnerung nicht zugänglich sind. Hier werden zum Beispiel mechanische und motorische Fähigkeiten und Handlungsabläufe gespeichert, also beispielsweise Fertigkeiten wie Autofahren oder die Bewegungsabläufe beim Sport. Das Erlernen solcher Inhalte ist (mit Ausnahmen) relativ langsam. Der Umgang mit den Inhalten ist unflexibel, andererseits sind diese Gedächtnissysteme äußerst zuverlässig. Das nichtdeklarative Gedächtnis teilt sich ebenfalls in verschiedene Gedächtnissysteme, die wir aber nicht unterscheiden wollen.

Das *deklarative Gedächtnis* kann dadurch charakterisiert werden, dass seine Inhalte einer bewussten Erinnerung zugänglich sind. Es werden Repräsentationen von Objekten der Welt und deren Beziehung zueinander gespeichert. Die Inhalte haben einen verbalen oder bildhaften Charakter. Sie sind nicht als Einzelinformationen abgespeichert, sondern sie sind konzeptionell verbunden und organisiert. Lernen solcher Inhalte geht relativ schnell und der Umgang mit den Inhalten ist sehr flexibel. Dafür sind die gespeicherten Inhalte nicht immer zuverlässig. Sie sind oft unvollkommen und anfällig für Ungenauigkeiten und Verzerrungen. Details, die man zum Zeitpunkt des Lernens für wenig wichtig hielt, werden häufig nicht mitgespeichert und bei der Wiederabrufung sinngemäß rekonstruiert. Andererseits ist das deklarative Gedächtnis in dem Sinne zuverlässig, dass es allgemeine Kenntnisse, allgemeine Bedeutungen, das Wesentliche sicher abspeichert. Es wird weiter in das Wissensgedächtnis und in das episodische Gedächtnis unterteilt. Das Wissensgedächtnis, gelegentlich wird es auch semantisches Gedächtnis genannt, ist ein Faktengedächtnis und enthält unser Wissen über die Welt und über generelle Zusammenhänge. Das episodische Gedächtnis, welches auch autobiographisches Gedächtnis genannt wird, speichert Informationen, die uns unmittelbar betreffen. Dabei handelt es sich sowohl um Informationen über die Vergangenheit als auch um solche über die Zukunft, beispielsweise wenn wir uns etwas vorgenommen haben. Seine Inhalte enthalten Raum- und Zeitinformationen. In diesem Sinne ist es ein Quellengedächtnis, wir merken uns darin, wann und wo wir uns unser Wissen angeeignet haben.

Die Lehr- und Lerninhalte der Informatik sind kognitiver Art, in erster Linie handelt es sich um deklaratives Wissen. Beim Lernenden sind die Lerninhalte jedoch unter Umständen mit episodischen Informationen angereichert. Es ist häufig für den Lernerfolg förderlich, wenn der Lernende die Begleitumstände beim Lernen mitspeichert. Diese episodische Anreicherung ist eine Folge des Lernprozesses, des eige-

nen Beteiligtseins beim Lernen, sie ist nicht integraler Bestandteil des behandelten Stoffes. Häufig verlieren sich bei erneuter Bearbeitungen die episodischen Anteile. Das Themengebiet Algorithmen fügt sich in diesen Rahmen ein, bei der Visualisierung von Algorithmen handelt sich um deklaratives Wissen, das kommuniziert wird.

Lernen

Wenden wir uns nun konkreter dem Lernen zu. Die ersten universellen Lernmodelle, die gefunden beziehungsweise entwickelt wurden, entstammen der Tradition des Behaviorismus. Diese Richtung vertritt die These, dass Verhalten mit (natur)wissenschaftlichen Methoden untersucht und erklärt werden kann. Lernbedingte Veränderungen werden als Verhaltensänderungen angesehen und interpretiert. Eines dieser Modelle, die klassische Konditionierung, geht auf Iwan P. Pawlow zurück. Sie bietet eine angemessene Beschreibung von Lernsituationen, bei denen die Reaktion unwillkürlich auf den Reiz erfolgt, wie dies beispielsweise bei Reflexen der Fall ist. Zeitlich parallel begann Edward L. Thorndike Lernsituationen mit willkürlichem Verhalten zu untersuchen. Das Lernmodell, dass sich daraus entwickelt hat und heute stark mit dem Namen Burrhus F. Skinner verknüpft ist, wird operante Konditionierung genannt. Sie stellt den Zusammenhang zwischen Verhalten, zwischen der Aktivität des Organismus, und den darauf folgenden Konsequenzen in den Vordergrund. (Ein Versuchstier kann dadurch trainiert werden, dass erwünschtes Verhalten mit Futter belohnt wird, auch kann unerwünschtes bestraft werden.) Aspekte, die den Untersuchungsmethoden nicht zugänglich sind, werden außer Acht gelassen. Daher berücksichtigen behavioristische Lernmodelle in der Regel keine mentalen Faktoren, insbesondere lassen sie deklaratives Wissen außer Acht. Die behavioristische Sicht von Lernvorgängen hatte bis in die 1960er Jahre starke Bedeutung. In diesem Zeitraum setzte eine Wende ein und man wandte sich vermehrt kognitiven Aspekten zu.

Unter den Modellen, die mentale Aspekte einbeziehen, erlangte die sozial-kognitive Theorie von Albert Bandura große Bedeutung. Sie wurde mehrfach ergänzt und erweitert, heute stellt sie sich als eine tragfähige Synthese des behavioristischen und kognitiven Standpunktes dar. Derjenige Aspekt, der das Lernen beschreibt, heißt Beobachtungslernen oder Modelllernen. Die behavioristische Tradition betont die Rolle der eigenen Erfahrung beim Lernvorgang. Das Beobachtungslernen geht von der „Beobachtung“ aus, dass eigene, unmittelbare Erfahrung für das Lernen nicht zwingend erforderlich ist. Lernen kann auch durch die Beobachtung fremden Verhaltens und dessen Konsequenzen, durch die Beobachtung eines Modells erfolgen. Vermutlich beruht sogar der Großteil der menschlichen Lernvorgänge auf Beobachtungslernen, sicher aber der Großteil des deklarativen Wissens. Das beobachtete Modell kann real sein, etwa eine Person. So wusste beispielsweise bereits die griechische Pädagogik um die Wirkungen eines Vorbildes. Das beobachtete Modell kann aber auch auf einer abstrakteren Ebene, symbolisch, etwa in einem Buch, vorliegen.

In der symbolischen Schilderung können Verhaltensmuster und die darauf folgenden Reaktionen der Umwelt mittelbar erfahren werden.

Wenn man beim Lernen Aspekte mitberücksichtigen möchte, die über das Reagieren auf Reize, über das Bilden von Assoziationen und über das Reagieren auf die Konsequenzen des eigenen Verhaltens hinausgehen, dann spricht man häufig vom *Komplexen Lernen.* In diesem Lernmodell sind auch Denkprozesse und die Verwendung von Problemlösungsstrategien mit eingeschlossen. Ein Charakteristikum dieses Modells besteht in der Einführung einer neuen, mentalen Abstraktionsebene: Als Erstes wird aus den Wahrnehmungen ein mentales Abbild erzeugt. Als Zweites werden verschiedene Verhaltensmöglichkeiten durch Manipulation des mentalen Abbildes simuliert und mögliche Konsequenzen abgeschätzt. Schließlich und Drittens wird eine Verhaltensweise ausgewählt und in die Praxis umgesetzt. Auf Wahrnehmungen wird nicht mehr unmittelbar, sei es reflexartig oder erlernt, reagiert, sondern das mögliche eigene Verhalten wird zuerst reflektiert. Das mentale Abbild repräsentiert die verschiedenen Aspekte der Realität in unterschiedlichem Maße: einige sind stark ausgeprägt, andere können ganz vernachlässigt sein. Die Herstellung des mentalen Abbildes schließt Auswahl und Priorisierung von (Aspekten der) Wahrnehmung ein. Wir können das mentale Abbild ebenfalls als Abstraktion auffassen, seine Herstellung als Abstraktionsprozess.

Grundsätzlich gibt es zwei Möglichkeiten für die Herstellung einer Abstraktion: Sie kann induktiv oder deduktiv gebildet werden. Mit *Induktion* bezeichnet man allgemein eine Methode, die vom Einzelnen, vom Besonderen auf etwas Allgemeines, Gesetzmäßiges schließt [Schmidt 1982]. In der Philosophie und Erkenntnistheorie ist sie als Methode der Erkenntnisgewinnung schon Sokrates und den Epikureern bekannt. Aus einzelnen Beobachtungen wird eine Vermutung über den Zusammenhang der betrachteten Phänomene abgeleitet. Diese Vermutung kann dann, hauptsächlich durch planmäßige Beobachtungen und Experimente, anhand der Tatsachen überprüft werden. Bei guter Übereinstimmung kann die Vermutung zur Gewissheit erhoben werden. Problematisch ist dieses Vorgehen immer dann, wenn die Verallgemeinerung auf der Basis von nur wenigen Beobachtungen erfolgt. Dann ist die Gefahr gegeben, dass Wichtiges in dem vermuteten Zusammenhang nicht mitberücksichtigt ist. Gleiches gilt, wenn die Überprüfung der aufgestellten Vermutung nur oberflächlich erfolgt. Im Gegensatz zur Induktion steht die *Deduktion.* Sie bezeichnet die Ableitung des Besonderen aus dem Allgemeinen. Die Anwendung der Deduktion setzt die Kenntnis des Allgemeinen voraus. Daher eignet sie sich nur für solche Situationen und Phänomene, für die allgemeine Beschreibungen existieren. Aus ihnen kann dann auf das Besondere geschlossen, deduziert werden. Die Deduktion hat als Methode der Erkenntnisgewinnung in denjenigen Wissenschaftsbereichen ein hohes Ansehen, die über allgemeine Modelle verfügen. Traditionell sind das die Mathematik und die Naturwissenschaften, aber auch die (theoretische) Informatik reiht sich hier ein.

Alle Lebewesen können Reize von der Umwelt aufnehmen. Wenn wir uns auf Tie-

re beziehen, dann sind die Sinnes- und Körperwahrnehmungen die primären Reizquellen. Aus ihnen bildet sich ein Großteil des Wissens und des Verständnises der (Um-)Welt. Aus diesem Grund stellt sich die Induktion als eine sehr natürliche Lernmethode dar. In der evolutionären Entwicklung des Menschen hat sie sich als Methode gefestigt und ausgeprägt. Menschen sind bei der Anwendung der induktiven Methode sehr leistungsfähig, vergleiche [Stangl 2009].

Induktives Lernen benötigt allerdings für eine genaue Grenzziehung bei Abstraktionen in der Regel viel Zeit. Betrachten wir dazu ein Beispiel zum Kategorienlernen: Wir wollen die Kategorien Blumenschale und Blumenvase lernen, wobei wir uns auf die beiden Aspekte Größe der Grundfläche und Höhe des Gefäßes beschränken. Wir betrachten dazu eine Reihe von Gefäßen, welche die funktionalen Voraussetzungen erfüllen. Ist die Höhe eines Gefäßes im Vergleich zu seinem Durchmesser klein, so werden wir es als Blumenschale ansehen; ist der Quotient groß, dann werden wir es als Vase ansehen. Dazwischen gibt es einen Bereich, wo die Zuordnung unklar ist, Zweifelsfälle können wir nur schwer beurteilen. Dieser Sachverhalt zeigt sich auch in Experimenten: Wenn Objekte vorgelegt werden und entschieden werden soll, ob sie zu einer bestimmten Kategorie gehören, dann gelingt die Zuordnung bei typischen Vertretern der Kategorie schneller als bei untypischen. Die Grenzziehung wird genauer, je mehr zweifelhafte Gefäße beurteilt werden müssen.

Da sich eine genaue Grenzziehung bei der Induktion als langwierig herausstellt, ist es gewöhnlich eine gutes Vorgehen, an eine induktive Lernlektion möglichst bald eine deduktive anzuschließen. Nach der induktiven Lektion wurde ein grundlegendes Verständnis für die zu lernende Abstraktion, also eine Basisabstraktion, entwickelt. Typische Vertreter können (sicher) zugeordnet und typische Nicht-Vertreter können (sicher) abgelehnt werden. Eine deduktive Lektion vervollständigt die Abstraktion, insbesondere wird die Grenzziehung genauer. Die schon entwickelte Basisabstraktion erleichtert zudem das Verständnis des präsentierten Modells (der Abstraktion).

Das Gedächtnis ist kein isolierter Teil des Gehirns, es ist auf vielfältige Weise mit anderen Gehirnprozessen verknüpft, die auch auf das Lernen wirken. Zwei Beispiele sollen das erläutern. Es besteht eine sehr enge Verknüpfung zwischen dem Gedächtnis und Emotionen. Der Großteil der deklarativen Informationen passiert auf dem Weg ins Langzeitgedächtnis Gehirnbereiche, die diese Informationen mit emotionalen Zuständen integrieren, sie mit Emotionen anreichern. Der emotionale Kontext, bei dem Informationen eingespeichert (und abgerufen) werden, hat einen Einfluss auf die Erinnerungsleistung, was in zahllosen Experimenten nachgewiesen werden konnte. Wenn Informationen in einem positiven emotionalen Kontext gespeichert werden, dann können sie leichter erinnert werden. Ein weiteres Beispiel ist die Aufmerksamkeit, die zwei verschiedene Phänomene zusammenfasst. Zum einen beinhaltet er die Vigilanz, d. h. die Wachheit, die den Grad der allgemeinen Aktivierung des Gehirns beschreibt. Hiervon zu unterscheiden ist die selektive Aufmerksamkeit. Sie beschreibt die Zuwendung zu einem bestimmten Sachverhalt und die Ausblendung

anderer Sachverhalte, also die Fähigkeit, bestimmte Reize vorrangig zu bearbeiten. Selektive Aufmerksamkeit ist ein Prozess, der steuert, welche Informationen aus dem Ultrakurzzeitgedächtnis ins Arbeitsgedächtnis gelangen. Dieser Begriff beschreibt Konzentration.

Der Themenkomplex Lernen ist bei weitem noch nicht befriedigend genug verstanden. Lernbedingte Verhaltensänderungen lassen sich angemessen beschreiben und erklären. Sobald es aber um die beim Lernen stattfindenden kognitiven Prozesse geht, herrscht Unklarheit. So können viele wichtige Fragen bezüglich des Lernens bisher nicht abschließend beantwortet werden. Ungeachtet der teilweise unklaren wissenschaftlichen Kenntnislage lassen sich einige allgemeine Prinzipien und Hinweise zum Lernen herauskristallisieren. Lernen ist auch keine Erfindung unserer Zeit, deshalb können wir auf die Lernerfahrungen vieler Generationen zurückgreifen. Sie treten uns als Lernstrategien oder Lernhinweise in mehr oder weniger konkreter Form entgegen. Einige Faktoren werden häufig genannt, weswegen man ihnen einen allgemeinen Charakter zugestehen kann. Von diesen erscheinen uns die folgenden besonders bedeutend: 1. Je mehr wir dem Lernstoff eine „Bedeutung“ (für uns) geben können, je mehr wir Grund haben ihn zu lernen, desto effektiver ist der Lernprozess. 2. Je ganzheitlicher der Lernprozess gestaltet wird, das heißt je mehr man von sich einbringen kann, desto effektiver ist er. Aktivität ist förderlicher als Passivität. Die Nutzung vieler Sinne ist vorteilhaft, insbesondere bildhafte Vorstellungen. Auch lässt sich etwas leichter lernen, wenn man den Lernkontext, die Begleitumstände beim Lernen mitspeichert. 3. Eine Strukturierung des Stoffes verbessert die Gedächtnisleistung. Die einzelnen Teile können in ein Gesamtkonzept eingeordnet werden, ihnen kann eine Bedeutung für das Ganze zugeschrieben werden. Die einzelnen Lerneinheiten sollten in Quantität und Qualität überschaubar sein, der Schwierigkeitsgrad ansteigend. 4. Je mehr Assoziationen wir bei einem Lernstoff herstellen können, desto stärker ist die Vernetzung im Gedächtnis. Diese vergrößert die Abrufemöglichkeiten. 5. Rechtzeitiges Wiederholen unterstützt eine dauerhafte Einspeicherung. Die Wiederholungsphasen sollen im Vergleich zu den Lernphasen eher kurz sein und die Abstände zwischen zwei Wiederholungseinheiten dürfen sich vergrößern. 6. Feste Lerngewohnheiten sind förderlich, dazu gehören: feste Lernzeit, fester Lernort, bei Unkonzentriertheit besser die Lerneinheit abbrechen und so weiter.

2.2 Computerunterstütztes Lernen von Algorithmen

Im vorangegangenen Teilabschnitt haben wir den Themenkreis Lernen behandelt. Das war durch die Tatsache motiviert, dass wir Algorithmenvisualisierung in einem weiten Sinn als ein Mittel zur Unterstützung von Lernvorgängen verstehen. Nachdem dort einzelne Aspekte zum Lernen in eher allgemeiner Weise dargestellt

wurden, soll in diesem Teilabschnitt konkreter auf Lernsoftware und Algorithmenvisualisierung eingegangen werden. Ausgehend von der Frage, welche Rolle technische Medien im Allgemeinen und Computer im Besonderen bei Lernvorgängen spielen (können), diskutieren wir die Rolle, die Algorithmenvisualisierung beim Lernprozess einnehmen kann. Auch behandeln wir die Art, wie bei der Nutzung der Algorithmenvisualisierung die Abstraktionsbildung beim Anwender erfolgt.

In den letzten fünfzig Jahren wurden eine Reihe neuer, gewöhnlich technischer, Medien entwickelt, die sich uns heute als ein Bestandteil unseres alltägliches Lebens darbieten. Man begann sich auch für die ihnen innewohnenden Möglichkeiten in Bezug auf Lernvorgänge zu interessieren. Dabei hat sich die Überzeugung herauskristallisiert, dass ihnen eine Rolle beim Lernen zuzugestehen ist. Allgemein gesprochen sind sie geeignet, eigene Erfahrung vorzubereiten, womit sie sich in das Modell des Beobachtungslernens einordnen, oder sie zu vertiefen, fortzuführen oder zu differenzieren. Computer bieten ein reiches Potential an Flexibilität und Interaktion, was sie für Lernzwecke besonders interessant erscheinen lässt. Spätestens seit dem Aufkommen und der größer werdenden Verbreitung von Personal Computern gewann auch das computerunterstützte Lernen an Interesse und Bedeutung.

Diese Zeit war von einer recht großen Technikgläubigkeit im Hinblick auf das Thema Lernen geprägt. Neue Medien und Techniken wurden enthusiastisch gefeiert oder zumindest freudig begrüßt, so als läge in ihnen der Schlüssel zur Lösung existierender Lernprobleme. Sollte eine neue mediale Technologie doch einmal die in sie gesteckten Erwartungen nicht erfüllen, dann richtete sich das Augenmerk auf einen Nachfolger. In der öffentlichen Diskussion in Deutschland gewann man den Eindruck, besonders in der Folge der PISA-Studien, dass der Schule im Allgemeinen und den Lehrern im Besonderen immer weniger an Kompetenz zugestanden wurde. Und dieses ungeachtet der Tatsache, dass es sich bei Lehrern um über viele Jahre ausgebildete Fachkräfte handelt. Daher erscheint es als sehr erstaunlich, dass man Lernsoftware ein solches Maß an Qualifikation zuzugestehen bereit ist.

Viele der in den anfänglichen Jahrzehnten geäußerten Ansichten über die Rolle und Bedeutung medialer Lernformen haben sich inzwischen als überzogen oder sogar falsch herausgestellt. Kritische Sichtweisen, stellvertretend sei auf [Dillon 1996] verwiesen, klassifizieren sie sogar als Mythen. Damit sind natürlich auch viele Hoffnungen unrealistisch, die man in den neuen Medien gesehen hat. Diese prinzipielle Einschätzung gilt auch für computerunterstützte Lernmethoden. Mittlerweile dürfte (wenigstens unter Fachleuten) die Euphorie einer realistischeren Einschätzung gewichen sein. Die Beurteilung und Bewertung von (medialen) Lernmethoden ist im Allgemeinen schwierig und die Ergebnisse sind äußerst kontrovers. Das Hauptproblem dürfte darin begründet liegen, dass der eigentliche Lernvorgang nicht beobachtbar ist. In [Schulmeister 2002] wird die derzeitige Situation sehr prägnant zusammengefasst: „Die meisten experimentellen Vergleiche von Unterrichtsmethoden erzielen keine signifikanten Ergebnisse, und die wenigen signifikanten Resultate widersprechen sich gegenseitig.“

Ungeachtet des kontroversen Kenntnisstandes lassen sich einige allgemeine Bewertungen abgeben. Unter den positiven können als wichtig hervorgehoben werden: 1. Computerunterstütztes Lernen kann mehrere Sinne ansprechen, an der Verarbeitung der zu lernenden Informationen sind dann mehrere Gehirnregionen beteiligt. 2. Es kann von Zwängen befreien, die durch Lehrinstitutionen oder Lehrer bedingt sind. 3. Es erlaubt die Umsetzung didaktischer Prinzipien, die bei traditionellen Lehrmethoden nicht oder nur schwer zu realisieren sind. (Ungeachtet der Bewertung spezieller didaktischer Prinzipien ist allein die Möglichkeit ihrer Verwendung positiv zu werten.) Auf der negativen Seite wird hauptsächlich zu bedenken gegeben, dass computerunterstütztes Lernen im Vergleich zu anderen Lernformen zu einer Reduktion der sozialen Beziehungen führen kann.

Der Begriff *Visualisierung* ist allgemein ausgedrückt eine Bezeichnung für die bildliche Aufbereitung, Darstellung und Kommunikation von Informationen. Die Aufnahme visualisierter Informationen erfolgt über den Sehsinn. Gemessen an der Anzahl Nervenfasern, die von den Sinnesorganen zum Gehirn führen, und an der Anzahl Neuronen[1], die mit der Verarbeitung dieser Sinnesreize befasst sind, ist der Sehsinn der dominierende Sinn. Visualisierung spricht als Methode also den auf materieller Ebene am besten ausgebauten Eingangskanal an. – An dieser Stelle könnte eingewandt werden, dass hieraus nicht per se eine positive Charakterisierung der Visualisierung folgt. In den letzten Jahren ist die Vorstellung sehr populär, dass sich, je nach bevorzugten Wahrnehmungskanälen, Lerntypen charakterisieren lassen. Eine dieser Lerntypentheorien (nach J. Hüholdt) postuliert neun Lerntypen: den visuellen, den auditiven (hörsinnorientiert), den audiovisuellen, den haptischen (tastsinnorientiert), den olfaktorischen (geruchssinnorientiert), den abstrakt-verbalen, den kontakt- bzw. personenorientierten, den mediumorientierten und den einsicht- bzw. sinnanstrebenden Lernenden. Diese Lerntypen sollen nicht für sich allein sondern nur in Mischformen auftreten, auch sollen sie sich zu einem gewissen Grad an die jeweiligen Gegebenheiten anpassen können. Solche Lerntypentheorien sind aber keineswegs unumstritten, vergleiche beispielsweise [Stangl 2009]. Zusammenfassend lässt sich ungeachtet persönlicher Präferenzen bei Wahrnehmungskanälen festhalten, dass Visualisierung den bei Menschen dominierenden Wahrnehmungskanal verwendet.

Bei oberflächlicher Sicht kann man Visualisierung als eine alternative Form der Präsentation von Informationen ansehen. Als solche wird sie den Anspruch erheben, wichtige Sachverhalte deutlicher und klarer als konkurrierende Methoden zu kommunizieren. Ihre weiterreichende Bedeutung zeigt sich, wenn es um die Darstellung großer Datenmengen geht, wie sie zum Beispiel als ökonomische Daten in Unternehmen, als Messwerte aus Experimenten oder als meterologische Daten aus Wetterbeobachtungen auftreten. Solch große Datenmengen können wir symbolisch nicht komplett erfassen, da die Kapazitäten unseres Arbeitsgedächtnisses sehr klein sind. Traditionelle Methoden der Datenaufbereitung filtern wichtige Daten heraus

1. Neuronen sind diejenige Zellart, die die informationsverarbeitenden Prozesse bewerkstelligen.

oder verdichten sie durch Ableiten von Kennzahlen. Visualisierung ist eine Methode, die auch große Datenmengen für Menschen erfassbar macht. Sie eignet sich damit auch allgemein für die Darstellung komplexer Sachverhalte.

Für Visualisierung am Computer zum Zwecke des Lernens gilt sowohl das über computerunterstütztes Lernen als auch das über Visualisierung Gesagte. Sind Algorithmen der Gegenstand der Visualisierung, dann spricht man von *Algorithmenvisualisierung*. Wir unterscheiden zwei Arten: Bei der klassischen Algorithmenvisualisierung wird die Ausführung des Programms für konkrete Eingabedaten visualisiert, das heißt der Programmablauf wird Schritt für Schritt durchgegangen und die aktuellen Datenwerte werden präsentiert. Da in der Regel immer nur ein Datensatz betrachtet wird, ist die Datengröße normalerweise überschaubar. In dieser Arbeit beschreiben wir einen alternativen Ansatz zur Algorithmenvisualisierung. Wir visualisieren die abstrakte Ausführung eines Programms. In einem Vorberechnungsschritt werden mittels statischer Programmanalyse Invarianten für jeden Programmpunkt berechnet. Als Ergebnis erhalten wir für jeden Programmpunkt eine (textuelle) Beschreibung der an diesem Programmpunkt möglichen Datenkonfigurationen. Die Größe und Komplexität dieser Beschreibungen hängt von zwei Faktoren ab: Je komplizierter die Datenstruktur ist, desto komplizierter ist die Analysespezifikation und desto umfangreicher sind die resultierenden Beschreibungen. Je feiner die Analysespezifikation gestaltet wird, das heißt je mehr Aspekte berücksichtigt werden sollen, desto umfangreicher sind die resultierenden Beschreibungen. Im Gegensatz zur Visualisierung der konkreten Programmausführung hat der hier beschriebene Ansatz dem Problem Rechnung zu tragen, dass große Datenmengen vorliegen. Auch unter diesem Aspekt bietet sich Visualisierung als Methode an.

Bei beiden Ansätzen zur Algorithmenvisualisierung erzeugt der Lernende ausgehend von der Visualisierung ein mentales Abbild, eine Abstraktion (von Aspekten) des Algorithmus. Allerdings ist die Methode der Abstraktionsbildung in beiden Fällen verschieden: Traditionelle Algorithmenvisualisierung führt zu einer induktiven Abstraktionsbildung, der hier beschriebene Ansatz führt zu einer deduktiven Abstraktionsbildung. Wir Menschen sind bei der Anwendung der induktive Abstraktionsbildung sehr leistungsfähig. Allerdings erlaubt sie uns kaum, in vertretbarer Zeit eine genaue Grenzziehung vornehmen zu können. Dieser Aspekt ist bei Algorithmen aber wichtig. Ein Algorithmus muss für alle Eingaben die an ihn gestellte Aufgabe lösen, er muss immer eine korrekte Ausgabe produzieren. Das Verständnis eines Algorithmus impliziert, dass wir für jede Eingabe seine Aktionen verstehen und sicher sind, dass dieses Vorgehen die Aufgabe löst. Konkret müssen wir also verstehen, warum er für bestimmte Eingaben, in bestimmten Fällen, dieses tut, während er für andere jenes tut. Wünschenswerterweise sollten wir auch verstehen, welche Eigenschaften der Daten es sind, die solche Unterscheidungen bedingen. Es ist also eine klare Grenzziehung zwischen den auftretenden Fällen vonnöten. Die deduktive Methode der Abstraktionsbildung bietet eine Alternative zur induktiven, sie erlaubt eine klar(er)e Grenzziehung.

Zu Beginn dieses Abschnittes haben wir drei positive Charakterisierungen für mediale Lernformen angeführt. Sie gelten natürlich auch in Bezug auf Algorithmenvisualisierung. Wir greifen die Punkte der Reihe nach wieder auf und ergänzen sie durch konkrete Anmerkungen. Dass Visualisierung den dominierenden Sinn anspricht, haben wir schon dargelegt. Die Befreiung von durch Institution und Lehrer bedingte Zwänge und Restriktionen ist weitreichend. Voraussetzung für die Nutzung von Visualisierungssoftware ist der Zugang zu einem geeigneten Computer. Größtmögliche Flexibilität wird bei Verfügbarkeit eines eigenen Computers erreicht. Dann kann der Lernzeitpunkt und die Länge der Lerneinheit flexibel gewählt werden und unter anderem auch dem eigenen Tagesrhythmus und den eigenen Lerngewohnheiten angepasst werden.

Als Student ist man mit spezifischen Lehr- und Lernsituationen konfrontiert. Die vorherrschende Lehrform an der Universität ist die Vorlesung. Sie besteht primär aus monologischer Vermittlung. Eine weitere typische Veranstaltungsart ist das Seminar, dass sich als gemischt monologisch und dialogisch bezeichnen lässt. Eine Lernsituation mit dialogischem Charakter sind Arbeitsgruppen. Eine andere Möglichkeit zur Klassifikation (universitärer) Lernsituationen besteht im Grad der Aktivität des Lernenden. Vorlesungen sind in dieser Hinsicht passiv, es werden vorrangig kognitive Prozesse wie Aufmerksamkeit, Einspeichern und Erinnern angesprochen. Demgegenüber haben Arbeitsgruppen einen aktiven Charakter. Schulunterricht wird bezüglich dieser Kriterien häufig nicht homogen gestaltet, Phasen der monologischen Vermittlung wechseln mit Phasen der Kommunikation oder Aktivität ab. Didaktische Prinzipien nutzen also häufig eine Mischung verschiedener Lernmethoden (Lernsituationen). Wenn eine Visualisierungssoftware den Anspruch erheben möchte, didaktische Prinzipien umsetzen zu können, dann sollte sie in der Lage sein, verschiedene Lernsituationen zu modellieren. Für eine universitäre Zielgruppe ist es sicherlich nicht verkehrt, wenn sie die Möglichkeit einer „monologischen“ Vermittlung im Sinne einer Vorführung von Fakten und Zusammenhängen bietet. Außerdem sollte sie Möglichkeiten zur Exploration des Lernstoffes anbieten, damit sie in den Übungsbetrieb eingegliedert werden kann und der Lernende sich mit dem Stoff auf aktive Weise beschäftigen kann.

Nachdem wir typische Lernsituationen beschrieben haben, wenden wir uns der Rolle von Lernsoftware beim Lernprozess zu. Trotz vieler Unterschiede halten wir Lernsoftware in vielen Aspekten mit einem Buch vergleichbar. Für eine Reihe von Lernfaktoren verhalten sie sich ähnlich: Aspekte der individuellen Sinngebung liegen außerhalb beider Medien, das heißt sie geben in der Regel keine Antwort auf die Frage, warum man sich mit ihnen beschäftigen soll. Auch das Gedächtnis modulierende Prozesse wie Emotionen oder Motivation liegen ebenfalls außerhalb des Mediums. Vielleicht möchte man auf die These entgegnen, dass Papier ein lineares Medium sei, was eine (große) Einschränkung darstellt, während dies bei Softwareprodukte nicht notwendigerweise der Fall ist. Dieses Argument ist zweifelhaft, der lineare Charakter von Papier wird beispielsweise in [Dillon 1996] als Mythos charakterisiert. Diese Einschätzung ist auch konsistent mit den Empfehlungen vieler

Lesetechniken, die Lesen eher als einen dialogischen Prozess ansehen. (Sie interpretieren Lesen als eine Phase, in der man vom Text Fragen beantwortet haben möchte, die man sich vorher nach einer Phase des Textüberfliegens gestellt hat.) Natürlich hat jedes Medium spezielle, ihm eigene Ausdrucksmöglichkeiten. Software kann unter anderem Texte, Tabellen, Grafiken und Bilder anzeigen; damit sind im Medium Software die Möglichkeiten des Mediums Buch (im Wesentlichen) enthalten. Software kann Eigenschaften zur Verfügung stellen, die ein Buch nicht anbieten kann, denken wir beispielsweise an Interaktionsmöglichkeiten. Aber ob ein mehr an Möglichkeit auch ein mehr an Lernqualität bedeutet, ist nicht gesichert. Auch mit Notizzettel und Schreibstift kann man aktiv und kreativ mit dem Lernstoff umgehen.

Wir sehen unseren Ansatz der Algorithmenvisualisierung als ein Mittel zum Lernen von Algorithmen. Dabei fassen wir den Begriff Algorithmenlernen in einem weiten Sinne auf. Zum einen meinen wir damit das Algorithmenlernen im engeren Sinn, das Erlernen, das Verstehenwollen eines Algorithmus, zum Beispiel im Rahmen einer Vorlesung über Algorithmen und Datenstrukturen. Als Benutzergruppe sehen wir in erster Linie Studenten. Zum Anderen kann unser Ansatz auch anderen Benutzergruppen neue und nützliche Sichtweisen bieten, zum Beispiel kann er beim Debuggen und Optimieren von Programmen hilfreich sein. Solche Nutzer wollen letztendlich ihr Verständnis des Programms im Hinblick auf ihre Zielsetzung, etwa Fehler zu finden oder bestimmte Aspekte effizienter programmieren zu wollen, verbessern. Deshalb subsumieren wir diese Nutzung ebenfalls unter den Begriff Lernen. Bei solchen Benutzern wird es sich ebenfalls um, gegebenenfalls fortgeschrittene, Studenten oder um Personen mit ähnlichem Wissensstand handeln. In Kapitel 9 behandeln wir die verschiedenen Rollen, die eine Person im Umgang mit der Visualisierungssoftware einnehmen kann, mit ihren jeweiligen Zielsetzungen und Anforderungen detaillierter.

Da wir den hier vorgestellten Ansatz zur Algorithmenvisualisierung in einem weiten Sinn als lernunterstützendes Mittel verstehen, sahen wir es als angebracht, ausgewählte Aspekte des Lernens zu besprechen. Im ersten Teilabschnitt dieses Kapitels waren das Gedächtnis und das Lernen unter einem allgemeineren Gesichtspunkt der Gegenstand der Diskussion. In diesem Teilabschnitt wurde auf die Algorithmenvisualisierung fokussiert. Es wurde die Rolle der Algorithmenvisualisierung beim Lernen behandelt und bezüglich einzelner Aspekte besprochen. Als computerbasierte Visualisierung ordnet sie sich in den Bereich des computerunterstützten Lernens ein, der diskutiert wurde. Ein Hauptaugenmerk lag auf der Art, wie beim Anwender die Abstraktionsbildung erfolgt: Bei der traditionellen Algorithmenvisualisierung entsteht sie durch Induktion, bei der Visualisierung der abstrakten Programmausführung durch Deduktion.

3 Shapeanalyse

> Mit der Logik allein ist die menschliche Natur nicht zu besiegen. Die Logik sieht drei Möglichkeiten, dabei gibt es ihrer eine Million!
>
> *(Fjodor M. Dostojewski)*

3.1 Einleitung

In dieser Arbeit behandeln wir einen Ansatz zur Visualisierung von Algorithmen beziehungsweise Programmen. Bei diesen beziehen wir uns auf imperative, zeigerbasierte Algorithmen. Das übliche Vorgehen bei der Algorithmenvisualisierung besteht darin, die Ausführung eines Programms anhand konkreter Daten zu visualisieren. Demgegenüber verwenden wir einen Ansatz, der die abstrakte Programmausführung visualisiert. Anstelle von konkreten Heapinstanzen verwenden wir abstrakte Heapzustände. Dazu benötigen wir einen Mechanismus, der solche Abstraktionen herstellt. Die im Programm enthaltenen Aktionen operieren nicht mehr auf konkreten Heaps, sondern auf deren Abstraktionen. Natürlich müssen die Abstraktionen der Heapinstanzen und ihr Zusammenspiel mit den Programmaktionen miteinander harmonisieren. Zudem wünschen wir von Seiten der Visualisierung Einflussmöglichkeiten auf die Abstraktion, damit sie und damit auch die Visualisierung den aktuellen Zielen der Visualisierungssitzung angepasst werden kann. Es wird also auch ein Spezifikationsmechanismus für die Abstraktion gewünscht.

Als Mechanismus zur Herstellung der Abstraktionen verwenden wir die sogenannte Shapeanalyse, wie sie unter anderem in [Reps u. a. 2002; Sagiv u. a. 2002] beschrieben ist. Dabei handelt es sich um eine allgemeine Methode zur statischen Programmanalyse von großer Universalität und Mächtigkeit. Als statische Analysemethode wird sie ausschließlich auf der Basis des Programms durchgeführt, konkrete Datenwerte werden nicht berücksichtigt. Dies korrespondiert mit unseren Wünschen von Seiten der Visualisierung. Es sind nicht die konkreten Werte von Variablen, seien sie numerischer Art oder Zeichenketten, sondern es ist primär die Zeigerstruktur im Heap und der Zeigervariablen, auf die wir unsere Aufmerksamkeit lenken. Statische Programmanalyse wird als Technik beim Compilerbau eingesetzt. Dort interessiert man sich traditionell zum Beispiel für die Tatsache, ob ein Zeiger (Variable oder Zeigerkomponente eines Heapelements) NULL ist oder ob zwei Zeigerausdrücke auf die gleiche Heapzelle zeigen (aliasing). Wenn man die Blickrichtung von Zeigern auf

Heapzellen wechselt, dann interessiert man sich beispielsweise dafür, ob auf eine Zelle mehrere Zeiger zeigen (sharing) oder ob die Datenstrukturen, auf die zwei Zeiger zeigen, zueinander disjunkt sind.

Shapeanalyse ist eine statische Programmanalysemethode, die sehr viel umfassendere Antworten anstrebt. Anstelle nur auf die Zeiger und deren Struktur zu fokussieren, werden Aussagen über die Form (shape) der Datenstruktur abgeleitet. Dazu zählen Aussagen über die Erreichbarkeit von Heapzellen, beispielsweise von Zeigervariablen, aber auch untereinander, oder über Zyklizität, also das Vorhandensein oder Nichtvorhandensein von Kreisen in einem Heapbereich. Überdies zählen hierzu auch nichtstrukturelle Eigenschaften, also Aussagen, die sich nicht auf die Zeigerstruktur beziehen. Wenn man zum Beispiel ein neues Element in einen Suchbaum einfügen will, dann muss man dabei sicherstellen können, dass nach dem Einfügen wieder ein Suchbaum vorliegt; dazu sind Sortierinformationen über den Baum und einzelne Teilbäume nötig.

Shapeanalyse basiert auf dreiwertiger Logik. Als Analyseeingabe erhält sie neben einer logischen Spezifikation der Datenstruktur, die auch die Abstraktion bestimmt, eine Codierung des Programms und der Eingabeinstanzen. Daraus berechnet sie automatisch die abstrakte Programmausführung. Es wird zu jedem Programmpunkt die Menge der an diesem Programmpunkt vorkommenden abstrakten Heapzustände berechnet. Jede Menge kann als Invariante des entsprechenden Programmpunktes verstanden werden, wobei sie im Vokabular der Analysespezifikation formuliert ist. Ein einzelner dieser abstrakten Heapzustände heißt *Shapegraph*. Des Weiteren liegen die abstrakte Übergänge gemäß den Programmaktionen in Form eines Transitionsgraphen vor. Als Methode besitzt Shapeanalyse eine theoretische Fundierung. Damit hat insbesondere ihre berechnete Ausgabe eine klare semantische Bedeutung, was auch aus Sicht der Visualisierung als Vorteil zu bewerten ist.

Die Shapeanalyseausgabe dient als Eingabe für die Visualisierung. Eine Visualisierungssitzung erfolgt im Hinblick auf eine bestimmte Zielsetzung. Die Analyseausgabe soll daher auf die Zielvorgabe zugeschnitten sein. Dies impliziert, dass eine bestehende Analysespezifikation gegebenenfalls abgeändert oder gar eine neue erstellt werden muss. Daher müssen wir uns mit der Spezifikation von Shapeanalysen und dem Shapeanalysealgorithmus beschäftigen. Im praktischen Umgang ist in erster Linie ein Verständnis der Spezifikationsmechanismen wichtig. Ein Verständnis des Shapeanalysealgorithmus bietet hierfür Grundlagenkenntnisse, die besonders dann wichtig sind, wenn eine Spezifikation stark angepasst oder neu erstellt werden muss. Wir trennen beide Themenbereiche: In diesem Kapitel behandeln wir den Shapeanalysealgorithmus und die logischen Grundlagen, im folgenden Kapitel 4 widmen wir uns der Spezifikation von Shapeanalysen. Ziel dieses Kapitels ist, ein prinzipielles Verständnis des Algorithmus zu vermitteln; es geht nicht darum, die Theorie der Shapeanalyse in vollem Detailreichtum auszubreiten. Dabei werden wir die logischen Aspekte insgesamt ausführlicher besprechen, da wir sie im Verlauf der Arbeit benötigen.

3.2 Logik als Beschreibungsmittel

Logik hat sich im Laufe ihrer Geschichte für viele Anwendungsfälle als ein probates Mittel erwiesen. Unsere abendländische Logik hat ihren Ursprung in der Antike. Wenn man ihn an eine Person knüpfen möchte, dann wird gewöhnlich Aristoteles (384 v. u. Z. – 322 v. u. Z.) als ihr Gründungsvater angeführt. Ein zentrales Thema seiner logischen Untersuchungen ist das korrekte Schließen, also die Frage, wie aus richtigen Aussagen wieder richtige Aussagen folgen. Die ganze Aristotelische Logik kreist um einen Begriff: dem Syllogismus (vergleiche [Smith 2007]). Dabei handelt es sich um eine Schlussfigur, um einen Typ eines logischen Arguments. Syllogismen haben alle das gleiche Muster: Aus zwei Aussagen (Prämissen) wird eine andere (neue) Aussage (Konklusion) abgeleitet. Die Syllogismen unterscheiden sich untereinander in ihrer Struktur, also im Typ der Prämissen und der Konklusion. Diese allgemeine Schlussmethode benutzt er in zwei Bereichen. Zum Einen wendet er sie in seiner Theorie der Beweisführung an. Einen Beweis (apodeixis) charakterisiert er dadurch, dass er ein Schluss ist, der Wissen produziert. Eine wichtige Voraussetzung ist hierbei, dass die Prämissen wahr sind. Zum Zweiten wendet er sie in der Dialektik und Rhetorik an. Erstere ist die Theorie des (konsistenten) Argumentierens. Letztere ist die Theorie der Überzeugung; neben der glaubwürdigen Argumentation spielt die redetechnische Überzeugung, insbesondere die Fähigkeit, bei jeder Sache das möglicherweise überzeugende zu betrachten, hier hinein. In diesem Bereich wird von den Prämissen nur gefordert, dass sie allgemein akzeptiert sind. – Das Gedankengut der Antike hatte eine deutliche und oft starke Wirkung auf die Entwicklung der westlichen Gedankenwelt in den nächsten zwei Jahrtausenden. So gehörte zum Beispiel im Mittelalter die Logik (neben Rhetorik und Grammatik) zum Trivium des in der Antike entstandenen Kanons der Sieben Freien Künste.

In der Informatik erscheint die Logik in formaler, mathematischer Gestalt. Schon ab dem 17. Jahrhundert machen sich zunehmend mathematische Denkformen in der Logik bemerkbar, Gottfried Wilhelm Leibniz sei als Beispiel genannt (vergleiche [Schmidt 1982]). Dennoch muss man sie als eine Folge der Entwicklungen ansehen, die in der zweiten Hälfte des 19. Jahrhunderts einsetzten. Um dessen Mitte begann die Formalisierung der Logik. Beachtenswerte Arbeiten dieser Zeit stammen von George Boole und Augustus De Morgan. Ein zentrales Werk der zweiten Hälfte des 19. Jahrhunderts ist Gottlob Freges „Begriffsschrift“ aus dem Jahre 1879, in der die Logik axiomatisch entwickelt wird. Damit stellte zu diesem Zeitpunkt die fregesche Logik neben der Syllogistik das einzige formal aufgearbeitete Teilgebiet der Logik dar. Als sehr einflussreich erwies sich die in den Jahren 1910–1913 erschienene „Principia Mathematica“ von Bertrand Russell und Alfred North Whitehead. Dieser Stand der Entwicklung der Logik ist nicht nur geeignet, sondern auch ausreichend, um die konkrete Programmausführung zu beschreiben, was für den Fall der Heapbeschreibung im folgenden Abschnitt illustriert wird. Auf der Basis des in dieser Zeitperiode entstandenen Fundaments fand im 20. Jahrhundert eine rege

Weiterentwicklung der Logik statt, wir kommen in Abschnitt 3.4 kurz darauf zu sprechen.

3.3 Abstraktion von Heapzuständen

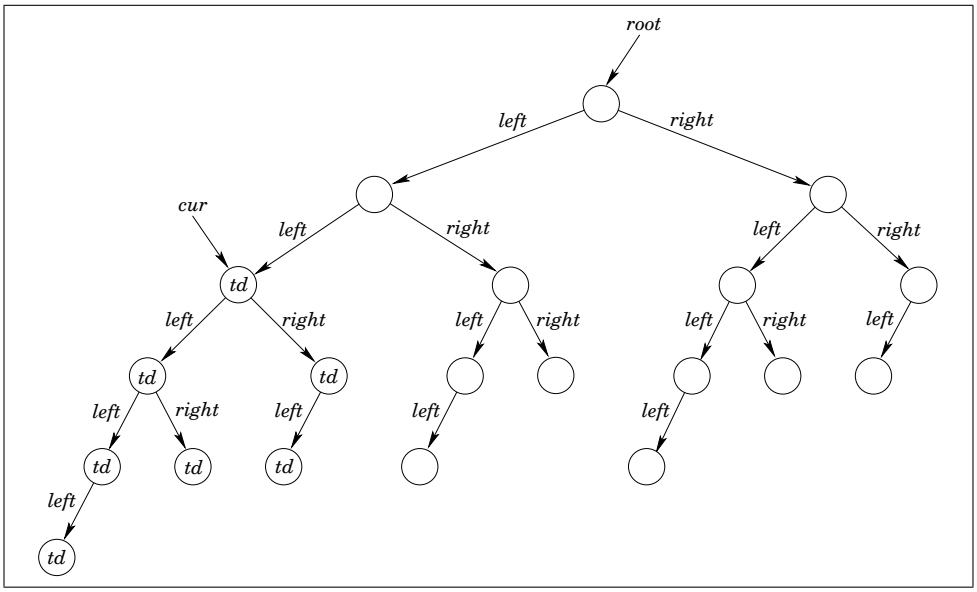

Abbildung 3.1: Konkrete Darstellung eines Suchbaums

Wir verwenden Logik zur Beschreibung von Heapzuständen, also von Eigenschaften der Datenstruktur im Heap. Shapeanalyse verwendet abstrakte Heapzustände, und die Abstraktion wird durch die logische Beschreibung bestimmt. Im Folgenden demonstrieren wir die Methode informal an einem Beispiel. In der Einleitung haben wir das Einfügen eines Elementes in einen Suchbaum betrachtet. Wir kehren hierauf zurück. Wir fokussieren auf den Suchalgorithmus und betrachten eine spezielle Eingabeinstanz. Der Baum und damit auch die Struktur der Datenstruktur im Heap sei wie in Abbildung 3.1. (Er unterscheidet sich von demjenigen in der Einleitung nur dadurch, dass das aktuelle Element, auf das der Zeiger cur zeigt, ein anderes ist.)

Als Erstes müssen alle relevanten Eigenschaft des Heaps mittels logischer Prädikate beschrieben werden. Zunächst muss die Zeigerstruktur erfasst werden. Für jede der vorkommenden Zeigervariablen führen wir ein einstelliges Prädikat ein. In unserem Fall sind dies root und cur. Eine Zeigervariable kann nur auf eine Adresse verweisen. Deshalb können wir für jedes dieser Prädikate p zusätzlich verlangen, dass es höchstens eine Heapzelle u geben darf, für die p zu wahr auswertet, für die $p(u) = 1$ gilt; diese ist die Heapzelle, auf die der Zeiger verweist. Um die Zeigerstruktur zwischen

Heapelementen zu codieren, führen wir für jede Zeigerkomponente des verwendeten Datentyps ein zweistelliges Prädikat ein. Ein Baumelement enthält zwei Zeiger, in unserem Beispiel führen wir die zwei Prädikate left und right ein. Für ein solches Prädikat q und zwei Heapzellen u_1 und u_2 bedeute $q(u_1, u_2) = 1$, dass u_2 von u_1 aus mittels des q-Zeigers erreichbar ist. Da eine Zeigerkomponente nur auf höchstens eine Heapzelle zeigen kann, können wir von diesen Prädikaten q ebenfalls zusätzlich verlangen, dass es zu jedem u_1 höchstens ein u_2 mit $q(u_1, u_2) = 1$ geben darf.

Anstelle von konkreten Heaps verwenden wir abstrahierte beziehungsweise abstrakte Heaps. Bei ihnen sind Heapzellen, die sich in bestimmten Eigenschaften nicht unterscheiden, zu Einheiten zusammengefasst. Eigenschaften von Heapzellen werden durch einstellige Prädikate ausgedrückt. Wir zeichnen eine Teilmenge P_a der einstelligen Prädikate als *Abstraktionsprädikate* aus. Zwei Heapzellen u_1 und u_2 werden zusammengefasst, wenn für jedes Abstraktionsprädikat $p_a \in P_a$ die Bedingung $p_a(u_1) = p_a(u_2)$ gilt. Diese Form der Abstraktion heißt *kanonische Abstraktion.* Wenn wir eine beliebige konkrete Heapinstanz auf diese Weise abstrahieren, dann hat sie höchstens $2^{|P_a|}$ viele abstrakte Heapzellen.

Im Suchbeispiel haben wir bisher root und cur als unäre Prädikate eingefügt. Um das Prinzip der Abstraktion klarer zu verdeutlichen, führen wir ein weiteres einstelliges Prädikat td („to do“) ein. Für ein Heapelement u gelte $\mathsf{td}(u) = 1$, wenn sich u in dem Teilbaum mit Wurzel cur befindet. Wir setzen $P_a = \{\mathsf{root}, \mathsf{cur}, \mathsf{td}\}$ und berechnen die kanonische Abstraktion bezüglich P_a von der Heapinstanz, die in Abbildung 3.1 dargestellt ist. Bei der Abstraktion entstehen höchstens acht abstrakte Heapelemente. Im betrachteten Beispiel treten jedoch nicht alle auf: Es gibt beispielsweise kein (konkretes) Heapelement u, das sowohl $\mathsf{root}(u) = 1$ als auch $\mathsf{td}(u) = 1$ oder alternativ $\mathsf{cur}(u) = 1$ erfüllt. Im Beispiel entstehen vier abstrakte Heapzellen: die Wurzel, das aktuelle Vergleichselement, eine, die aus den zwischen der Wurzel und dem aktuellen Element befindlichen Zellen samt der an ihnen herabhängenden Teilbäume besteht und eine, die für die Teilbäume steht, die am aktuellen Element herabhängen. Es sind dieselben, die in der rechten Darstellung von Abbildung 1.2 auf Seite 13 zu sehen sind.

Die Elemente des abstrakten Heapzustandes haben wir bestimmt. Es müssen noch die Prädikate übertragen werden. Im Grunde genommen ist schon klar, wie die Prädikate aus P_a beschaffen sind. Sie erhalten denjenigen Wahrheitswert, den alle zu dem abstrakten Heapelement zusammengefassten (abstrahierten) konkreten Heapzellen besitzen. In unserem Beispiel müssen noch die zweistelligen Prädikate left und right ins Abstrakte übertragen werden, oft existieren aber auch nullstellige Prädikate oder einstellige Prädikate, die keine Abstraktionsprädikate sind. Die generelle Idee ist wieder, als Wahrheitswert den „gemeinsamen“ Wert aller korrespondierenden konkreten Wahrheitswerte zu verwenden.

Da die Stelligkeit eines Prädikats keine Auswirkung auf die Methode hat, genügt es, wenn wir das Vorgehen an einem Prädikat als Beispiel demonstrieren. Wir be-

trachten left, und es bezeichne u die abstrakte Heapzelle, welche für die am aktuellen Element herabhängenden Teilbäume steht. Das ist das untere Heapelement in der rechten Darstellung von Abbildung 1.2 (Seite 13). Welchen Wahrheitswert soll $\mathsf{left}(u, u)$ erhalten? Betrachten wir dazu in Abbildung 3.1 die zu u abstrahierten konkreten Heapzellen, mit u_{k1} und u_{k2} seien zwei von ihnen bezeichnet. Unter diesen gibt es sowohl welche mit $\mathsf{left}(u_{k1}, u_{k2}) = 0$ als auch welche mit $\mathsf{left}(u_{k1}, u_{k2}) = 1$. Beide Festlegungen, $\mathsf{left}(u, u) = 0$ und $\mathsf{left}(u, u) = 1$ wären überaus problematisch: In beiden Fällen gäbe es zu u gehörende konkrete Heapzellen, für die left den konträren Wahrheitswert hätte.

Dieses Problem wird dadurch gelöst, dass man von einer zweiwertigen Logik, also einer Logik mit zwei Wahrheitswerten, zu einer dreiwertigen übergeht. Dort existiert neben den bisherigen, klassischen Wahrheitswerten 0 und 1 ein dritter Wahrheitswert. Wir schreiben ihn als $\frac{1}{2}$ und verstehen ihn als „unbestimmt“, „unklar“ oder „Wahrheitswert nicht bekannt“. In unserem Beispiel setzen wir $\mathsf{left}(u, u) = \frac{1}{2}$. Für zu u gehörende konkrete Heapzellen u_{k1} und u_{k2} kann dann entweder $\mathsf{left}(u_{k1}, u_{k2}) = 0$ oder $\mathsf{left}(u_1, u_2) = 1$ gelten.

Kommen wir noch einmal auf die Heapzellen zu sprechen. Eine abstrakte Heapzelle korrespondiert entweder zu einer einzelnen konkreten Heapzelle oder zu einer mehrelementigen Menge von konkreten Heapzellen. Für die Präzision der Shapeanalyse ist es vorteilhaft, diese Unterscheidung aufrechterhalten zu können. Zu diesem Zweck wird vom Shapeanalysealgorithmus automatisch zu jeder Anaylsespezifikation das einstelliges Prädikat sm (summary) hinzugefügt. Es repräsentiert die Eigenschaft, dass es sich bei einer abstrakten Heapzelle um mehr als eine konkrete Heapzelle handeln kann. Für eine (abstrakte) Heapzelle u bedeutet $\mathsf{sm}(u) = 0$, dass u eine einzelne konkrete Heapzelle bezeichnet; $\mathsf{sm}(u) = \frac{1}{2}$ bedeutet, dass u für mindestens eine konkrete Heapzelle steht. Der Fall $\mathsf{sm}(u) = 1$ ist nicht möglich und führt zu logischen Inkonsistenzen, vergleiche beispielsweise [Sagiv u. a. 2002, S. 244].

Wir repräsentieren die von der Shapeanalyse berechneten logischen Strukturen als (beschriftete) Graphen. Die Elemente des Universums sind seine Ecke. Deswegen nennen wir Individuen, die Elemente des Universums, gewöhnlich gemäß ihrer graphentheoretischen Entsprechung Ecken. Ein Individuum u mit $\mathsf{sm}(u) = \frac{1}{2}$ heißt *Summary-Ecke*. Ein Individuum u mit $\mathsf{sm}(u) = 0$ heißt *Individuenecke*, wobei das Wort Individuum hier auf seine Wortbedeutung (unteilbar) anspielt. Die Kanten des Graphen sind die zweistelligen Prädikate.

3.4 Erweiterung der klassischen Logik

Gegenstand der (formalen) Logik sind bestimmte Sprachen. Das Wort Sprache ist dabei, wie in der Informatik üblich, als formale Sprache zu verstehen. Damit ist eine

Logik zunächst einmal eine Menge von Wörtern[1] – also Zeichenketten – über einem vorgegebenen Alphabet. Eine Klasse dieser Sprachen sind die Sprachen der Prädikatenlogik über einer vorgegebenen Symbolmenge. Sie wurden erstmals von Frege in seiner Begriffsschrift eingeführt, wo er Prädikatenlogik zweiter Stufe betrachtete. Die Prädikatenlogik hat sich im letzten Jahrhundert zum vorherrschenden logischen System entwickelt. Zum einen erweist sie sich, was logische Fragestellungen anbetrifft, als eine reiche und fruchtbare Struktur. Zum anderen lassen sich viele in Anwendungen auftretende Fragestellungen durch sie formalisieren und logisch behandeln. Wegen dieser Universalität dürften sie wohl der am ausführlichsten untersuchte und der am meisten verwendete Typ von Logik sein.

Die Entwicklung der Logik bis hin zu Prädikatenlogik bezeichnen wir als *klassische Logik*. In anderen Darstellungen wird hierunter oft nur die vorformale Logik verstanden. Uns geht es mit der Begriffsfestsetzung jedoch mehr darum, die Abgrenzung zwischen Prädikatenlogik und den auf ihr fußenden Entwicklungen im 20. Jahrhundert zu betonen. – Allgemein wollen wir unter einer klassischer Logik eine formale Logik verstehen, die das Prinzip der Zweiwertigkeit und der Extensionalität erfüllt. Das Prinzip der Zweiwertigkeit bedeutet, dass es genau zwei Wahrheitswerte gibt, dass jeder logische Ausdruck (genau) einen von zwei Wahrheitswerten annimmt. (Hierin drückt sich auch der Satz vom ausgeschlossenen Dritten, tertium non datur, aus.) Das Prinzip der Extensionalität, das auch Kompositionalitätsprinzip genannt wird, bedeutet, dass sich der Wahrheitswert eines zusammengesetzten Ausdrucks funktional durch die Wahrheitswerte seiner primären Teilausdrücke bestimmt. Wenn wir zum Beispiel an den logischen Ausdruck $\varphi \vee \psi$ denken, dann bestimmt sich sein Wahrheitswert durch eine Funktion, die in diesem Fall Disjunktion genannt wird, deren Argumente die Wahrheitswerte von φ und ψ sind.

Oben haben wir das Prinzip der Abstaktion von Heapzuständen demonstriert. Dabei sind wir zunächst im Rahmen der klassischen Logik verblieben, am Ende sind wir aber auf Probleme gestoßen. Das Phänomen, dass die klassische, zweiwertige Logik ihre Grenzen hat und nicht immer ein adäquates Modell für die zu beschreibende Realität ist, stellt weder etwas Neues noch etwas Überraschendes dar. Schon Aristoteles war sich Grenzen bewusst. So betrachtet er in seiner Peri hermeneias (De interpretatione) Aussagen, die sich auf Zukünftiges beziehen. In Anlehnung an seine Ausführungen bilden wir den Satz „Morgen wird es zu einer Seeschlacht kommen." Was soll sein Wahrheitswert sein? Man könnte natürlich einen Tag abwarten und die Ereignisse beobachten. Aber was hat das Auftreten oder Nichtauftreten der Schlacht mit dem Wahrheitswert der Aussage zu tun? Nicht nur hat Aristoteles die Problematik der Aussagen über zukünftige Ereignisse aufgezeigt, er hat auch beschrieben, wie man in einem solchen Kontext schließen kann, in dem es neben Gewissheit und Ungewissheit um Notwendigkeit und Möglichkeit geht. (Bemerkenswerterweise haben diese Überlegungen keinen nachhaltigen Einfluss auf die Ausgestaltung seiner

1. Auch wenn viele deutschsprachige Logikbücher dem Leser anderes glauben machen, handelt es sich nur bei den allerwenigsten dieser Wörter um Worte.

Logik gehabt.)

Betrachten wir als Nächstes den Satz „Die größte ganze Zahl ist eine Primzahl.“ Wir sprechen über „die größte ganze Zahl“, also über ein Objekt, das nicht existiert. Kann eine Aussage, die sich auf etwas nicht Existierendes bezieht, wahr sein? Oder sollten wir mit der Tatsache, dass wir über die „größte ganze Zahl“ sprechen, ihre Existenz schon implizit postuliert haben? Wie kann andererseits der Satz falsch sein, wenn das Objekt, auf das sich die Aussage bezieht, nicht existiert? Wir tun uns offensichtlich schwer, Aussagen, die sich auf nichts beziehen, einen Wahrheitswert zuweisen. – Bleiben wir daher besser bei Existierendem. Betrachten wir den von Rudolf Carnap stammenden Beispielsatz „Cäsar ist eine Primzahl.“[2] Dabei bezeichne das Subjekt Cäsar die historische Person. Die Eigenschaft „Primzahl sein“ (oder „Primelement sein“) ist üblicherweise nur definiert für Elemente von Integritätsringen. In dem Satz wird fälschlicherweise versucht, die Eigenschaft auf ein Objekt einer anderen Kategorie anzuwenden. (Solcherart Fehler heißen üblicherweise Kategorienfehler, Carnap spricht von Sphärenvermengung). Wenn wir diesem Satz einen Wahrheitswert zuweisen wollen, so stehen wir im Grunde genommen vor den gleichen Problemen wie beim Beispielsatz zuvor. Wie kann eine Eigenschaft für ein Objekt wahr oder falsch sein, wenn sie sich auf das Objekt nicht anwenden lässt? – Des Weiteren können auch vage oder unscharfe Begriffe problematisch sein. Bei einem Satz wie „Die Antarktis ist bewohnt.“ muss zuerst geklärt werden, was „bewohnt“ bedeutet.

Auch die Mathematik ist vor solchen Problemen nicht gefeit. In der zweiten Hälfte des 19. Jahrhunderts entwickelte sich ein deutliches Unbehagen über die Grundlagen der Mathematik. Spätestens zum Ende des Jahrhunderts setzte ein beachtliches Interesse und Bemühen ein, sie neu zu ordnen. Diese Phase nennt man häufig die Grundlagenkrise der Mathematik, was ihre Bedeutung für die Mathematik deutlich genug betont. Bei der Fundierung der Grundlagen der Mathematik sind Logik und Mengenlehre zwei Grundpfeiler. Hier fanden sich schnell Paradoxien, die für beträchtliche Aufregung sorgten. Eine ist das Lügnerparadoxon, es bezeichnet die Aussage „Dieser Satz ist falsch.“ Wenn dieser Satz wahr ist, dann muss er nach der Aussage des Satzes falsch sein; wenn er hingegen falsch ist, dann besagt der Satz, das dies falsch sei, mithin der Satz wahr ist. Ein weiteres Beispiel ist die Russel'sche Antinomie. Wir betrachten die Menge aller Mengen, die sich nicht selbst als Element enthalten, also $A = \{M : M \notin M\}$, und bilden die Aussage $A \in A$. Wäre sie wahr, dann gälte nach Definition von A aber $A \notin A$; andererseits impliziert $A \notin A$ gemäß obiger Definition $A \in A$. Keiner der beiden Beispielaussagen können wir einen Wahrheitswert zuweisen, da stets ein Widerspruch entsteht. Eine Ursache der Problematik liegt darin begründet, dass beide Aussagen von Objekten handeln, die sich selbst referenzieren.

2. Rudolf Carnap: Überwindung der Metaphysik durch logische Analyse der Sprache. *Erkenntnis* 2 (1931), S. 219–241.

In der Vergangenheit wurde gelegentlich schon auf die Grenzen der (klassischen) Logik hingewiesen, aber eine tiefere Auseinandersetzung setzte erst in der 20er Jahren des 20. Jahrhunderts ein. Die dort untersuchten logischen Sprachen sind normalerweise Verallgemeinerungen der Prädikatenlogik. Es ist diese Art von Logik, die wir als *nichtklassische Logik* bezeichnen. Bei dieser Entwicklung geht es nicht darum, die „beste“ oder die „wahrste“ Logik zu finden und zu beschreiben, sondern darum, welche Logik für den beabsichtigten Anwendungsgegenstand gut geeignet ist. So haben sich mehrere Typen von Logiken herausgebildet, die alle ihre Berechtigung haben. Unter einem abstrahierenden Blickwinkel kristallisieren sich zwei Grundprinzipien der Erweiterung der klassischen Logik heraus.

Ein Ansatz beruht darauf, die klassische Logik so zu erweitern, dass Modalitäten berücksichtigt werden können. Modalität bezeichnet im philosophischen und logischem Sprachgebrauch das Wie des Seins oder Geschehens, den Grad der Bestimmtheit, also ob es sich beispielsweise um Wirklichkeit, um Möglichkeit oder um Notwendigkeit handelt. Der Aspekt der Wirklichkeit wird von der klassischen Logik abgedeckt, die angestrebte Erweiterung muss also Aussagen über Möglichkeiten und Notwendigkeiten handhaben können. Der so entstehende Typ von Logik heißt daher treffenderweise *Modallogik*. Modalitäten lassen sich auf unterschiedliche Weise interpretieren. Wenn man sie in Bezug auf Zeit interpretiert, dann kann man Notwendigkeit dahingehend verstehen, dass etwas (ab jetzt) immer gelten muss, Möglichkeit dahingehend, dass etwas zu irgendeinem Zeitpunkt gelten wird. Erweiterungen der Logik, die den Aspekt Zeit explizit berücksichtigen, heißen temporale Logiken. Eine modallogische Sicht impliziert also schon eine einfache temporale Logik. Modallogik finden unter anderem in der Informatik bei der Programmverifikation und in der künstlichen Intelligenz ein vielfältiges Anwendungsfeld.

Der zweite Ansatz besteht darin, die Anzahl der Wahrheitswerte, die klassische Logik kennt nur die Wahrheitswerte „wahr“ und „falsch“, zu vergrößern. Der auf diese Weise entstehende Typ von Logik heißt *mehrwertige Logik*. Die erste Arbeit zu diesem Thema stammt von dem polnischen Mathematiker (und Philosoph) Jan Łukasiewicz, der im Jahre 1920 eine Arbeit zur dreiwertigen Logik veröffentlichte. Praktisch gleichzeitig dazu führte der amerikanische Mathematiker Emil Post 1921 bei seinen Studium zur Repräsentierbarkeit von Funktionen zusätzliche Wahrheitswerte ein. Die Beweggründe für ihre Beschäftigung mit Erweiterungsmöglichkeiten der klassischen Logik waren verschiedenen. Während Łukasiewicz vornehmlich philosophische Motive hatte, waren es bei Post mathematische. Es ist diese Richtung der nichtklassischen Logik, welche die Grundlage von Shapeanalyse bildet. Sie verwendet eine Logik, die auf den amerikanischen Mathematiker Stephen Kleene zurückgeht. Er stellte 1938 eine dreiwertige Logik zur Behandlung von partiellen Funktionen vor. Der neue, zusätzliche Wahrheitswert erhielt dabei die Bedeutung „undefiniert“ oder „unbestimmt“. Auf die weitere geschichtliche Entwicklung brauchen wir nicht eingehen. Eine Zusammenfassung, auf die wir uns hier auch bezogen haben, findet sich beispielsweise in [Gottwald 2004]. Im Unterschied zur klassischen Logik gibt

die mehrwertige das Prinzip der Zweiwertigkeit auf, sie hält jedoch am Prinzip der Extensionalität fest.

3.5 Prädikatenlogik erster Stufe mit transitiver Hülle

Bisher haben wir Logik benutzt, um Heapzustände zu beschreiben. Sie wird bei der Shapeanalyse aber auch an anderer Stellen verwendet, beispielsweise bei der Beschreibung der Semantik von Programmaktionen. Aus diesen Gründen wäre es eigentlich sinnvoll mit der Semantik zu beginnen, zumal dies für Menschen vielleicht auch der intuitivere Zugang ist. Logik als Theorie trennt sehr deutlich zwischen Syntax und Semantik, wobei diese erst nach jener behandelt werden kann. Aus diesem Grunde folgen wir dem üblichen Vorgehen der Logik und beginnen mit der Syntax. Shapeanalyse verwendet für all ihre Belange nur eine logische Sprache: Prädikatenlogik erster Stufe mit transitiver Hülle. In dieser Arbeit geht es um Visualisierung. Daher genügt es, Shapeanalyse auf einem Niveau zu verstehen, welches es ermöglicht, die für die Visualisierung gewünschte Eingabe als Analyseausgabe zu erhalten beziehungsweise sie zu verstehen, sie „lesen zu können“. Für weitergehende detailliertere Ausführungen sei auf [Sagiv u. a. 1999, 2002] verwiesen. Bei der Notation weichen wir ab und folgen eher einer in der mathematische Logik, wie etwa in [Ebbinghaus u. a. 1996], gebräuchlichen.

Syntax der Prädikatenlogik erster Stufe mit transitiver Hülle

Zuerst legen wir das verwendete Vokabular, das Alphabet der Sprache fest. Es besteht wie üblich aus zwei Teilen. Zum einen enthält es Zeichen oder Symbole, die Bestandteil jeder Logik sind, und zum zweiten enthält es Zeichen für Prädikate (Relationen), die beliebig, also im Hinblick auf die beabsichtigte Anwendung, gewählt werden können. Zum ersten Bestandteil gehören die Zeichen

$$0, 1, \neg, \wedge, \vee,), (, \forall, \exists, =, \mathrm{TC},$$

wobei TC für transitive closure steht; außerdem gehören hierher eine Menge von Variablen, deren Elemente wir üblicherweise als $v_1, v_2, \ldots$ schreiben. Zum zweiten Bestandteil gehören Prädikate, die zur Beschreibung von Heapeigenschaften, der im Heap enthaltenen Datenstruktur und der Zeigervariablen verwendet werden. Für spätere Betrachtungen ist es hilfreich, wenn wir sie gemäß ihrer Stelligkeit n, $n \in \mathbb{N}$, in Mengen P_n zusammengefassen. Diese Menge dürfen leer sein. Des Weiteren setzen wir $P = \cup_{n \in \mathbb{N}} P_n$. Üblicherweise verlangt man in der Logik, dass diese (und alle ansonsten vorkommenden) Mengen abzählbar seien. Da es uns letztendlich um eine softwaremäßige Verarbeitung der auftretenden Objekte geht, verlangen wir gleich

einschränkender, dass all diese (und ansonsten vorkommenden) Mengen endlich seien und dass es nur endlich viele von ihnen gebe.

3.1 Vereinbarung. Alle in dieser Arbeit und insbesondere im Folgenden auftretenden Objekte seien, sofern nicht explizit etwas anderes vorausgesetzt wird, stets von endlicher Größe.

Üblicherweise werden bei der Definition der prädikatenlogischen Sprachen neben Prädikaten auch Konstantenzeichen und Funktionszeichen verwendet (beispielsweise in [Ebbinghaus u. a. 1996]). Das stellt keine Einschränkung dar, da Konstanten (Konstantensymbole) durch einstellige Prädikate (Prädikatssymbole) und n-stellige Funktionen durch $(n+1)$-stellige Prädikate ausgedrückt werden können. Auf der anderen Seite hat dieses Vorgehen den Vorteil, dass die folgende Definition der Wörter der Sprache einfacher ausfällt. (Terme brauchen nicht explizit definiert zu werden.) Außerdem vereinfacht sich dadurch auch die Formalisierung der Semantik.

Die Wörter einer logischen Sprache heißen Ausdrücke oder Formeln. Wir definieren sie induktiv. Gleichzeitig zeichnen wir eine Teilmenge der in ihnen vorkommenden Variablen als freie Variablen aus. Es sei P eine (endliche) Menge von Prädikaten. Die Ausdrücke (Formeln) bezüglich des Alphabets P sind genau diejenigen Wörter, die sich durch endlichmalige Anwendung der folgenden Regeln erzeugen lassen:

1. Die Zeichen 0 und 1 sind Ausdrücke ohne freie Variablen.
2. Für jedes Prädikatssymbol $p \in P$ von Stelligkeit n ist $p(v_1, \ldots, v_n)$ ein Ausdruck mit freien Variablen $\{v_1, \ldots, v_n\}$.
3. Sind v_1 und v_2 zwei Variablen, dann ist $(v_1 = v_2)$ ein Ausdruck mit freien Variablen $\{v_1, v_2\}$.
4. Ist φ ein Ausdruck mit freien Variablen V, dann ist $\neg\varphi$ ein Ausdruck mit freien Variablen V.
5. Ist φ ein Ausdruck mt freien Variablen V_φ und ψ ein Ausdruck mit freien Variablen V_ψ, dann sind $(\varphi \wedge \psi)$ und $(\varphi \vee \psi)$ Ausdrücke mit freien Variablen jeweils $V_\varphi \cup V_\psi$.
6. Ist φ ein Ausdruck mit freien Variablen V und $v \in V$, dann sind $(\forall v\colon \varphi)$ und $(\exists v\colon \varphi)$ Ausdrücke mit freien Variablen $V \setminus \{v\}$.
7. Ist φ ein Ausdruck mit freien Variablen V und $v_1, v_2 \in V$ sowie $v_3, v_4 \notin V$, dann ist $(\mathrm{TC}\ v_1, v_2 : \varphi)(v_3, v_4)$ ein Ausdruck mit freien Variablen $V \setminus \{v_1, v_2\} \cup \{v_3, v_4\}$.

Die Menge dieser Ausdrücke nennen wir die Sprache der *Prädikatenlogik erster Stufe mit transitiver Hülle* bezüglich des Alphabets P. – Neben den in der Definition der Ausdrücke benutzten Zeichen verwenden wir einige abkürzende Schreibweisen. So

stehe $(v_1 \neq v_2)$ für $\neg(v_1 = v_2)$, außerdem stehe $(\varphi \Rightarrow \psi)$ für $(\neg\varphi \vee \psi)$ und $(\varphi \Leftrightarrow \psi)$ für $(\neg\varphi \vee \psi) \wedge (\varphi \vee \neg\psi)$. Des Weiteren verzichten wir oft auf Klammern, wenn dies der intuitiven Erfassbarkeit des Ausdrucks dienlich erscheint.

Semantik der Prädikatenlogik erster Stufe mit transitiver Hülle

Logik beschäftigt sich in einem umfassenden Sinne mit dem Wahrheitsgehalt von Aussagen, also mit dem Wahrheitsgehalt von Ausdrücken. Ein weiterer Grundbaustein der Logik besteht folglich darin, jedem Ausdruck auf konsistente Weise einen Wahrheitswert zuzuweisen. Ausdrücke sind zunächst einmal nur Wörter, und für die Wahrheitswertzuordnung erweist sich die syntaktische Ebene in der Regel als nicht ausreichend. Dieses Phänomen wollen wir an einem Beispiel verdeutlichen. Dabei greifen wir allerdings voraus, wir verlassen die rein syntaktische Ebene und „interpretieren“ Ausdrücke in einem Kontext: Wir legen konkrete Objekte zugrunde, auf die sich ein Ausdruck bezieht, und wir assoziieren Prädikatssymbole mit realen Prädikaten über diesen Objekten. Wir betrachten den Ausdruck $p(v_1, v_2) \Leftrightarrow p(v_2, v_1)$ wobei p ein zweistelliges Prädikatssymbol, sowie v_1 und v_2 Variablen bezeichnen. Wir interpretieren im Folgenden die Variablen als (reelle) Zahlen. Wenn p die Gleichheitsrelation bezeichnet, dann ist der Ausdruck stets, also für jede Belegung der Variablen mit Zahlen, wahr. Die Gleichheitsrelation ist eine Äquivalenzrelation und folglich symmetrisch. Bezeichnet p hingegen die Kleinerrelation, dann ist der Ausdruck stets falsch. Eine Striktordnung ist asymmetrisch und kann somit den obigen Ausdruck für kein Paar von Zahlen erfüllen. Wiederum anders verhält es sich, wenn p die Kleinergleichrelation bedeutet. Dann gibt es Variablenbelegungen, für die der Ausdruck wahr ist (wenn beide Variablen gleiche Zahlen repräsentieren) und solche, für die der Ausdruck falsch ist (wenn beide Variablen verschiedene Zahlen repräsentieren). Der Wahrheitswert des Ausdrucks hängt demzufolge stark von dem realen Prädikat ab, das durch p bezeichnet wird, und üblicherweise auch von den Zahlen, die durch v_1 und v_2 bezeichnet werden.

Wenn wir also den Wahrheitswert eines Ausdrucks bestimmen beziehungsweise festlegen wollen, dann kann das nur geschehen 1. in Bezug auf einen Kontext, also bezüglich konkreter Objekte sowie konkreter Prädikate über diesen Objekten, und 2. in Bezug auf „konkrete Instanzen“ des Ausdrucks, also bezüglich einer Belegung der Variablen mit konkreten Werten. Das Teilgebiet der Logik, das sich damit beschäftigt, Ausdrücken einen Wahrheitswert zuzuordnen, ist die *Semantik*. In der klassischen Logik gibt es zwei Wahrheitswerte. Der eine heißt *wahr* und wir schreiben ihn als 1, der andere heißt *falsch* und wir schreiben ihn als 0. Es ist praktisch sie als (rationale) Zahlen zu aufzufassen. Auf diese Weise können auch Rechenoperationen auf ihnen durchgeführt werden, was gelegentlich elegantere Formulierungen ermöglicht.

Als Erstes werden Objekte benötigt, auf die sich ein Ausdruck beziehen soll. Dazu sei eine nicht leere Menge, ein Universum U gegeben. (Wir setzen zudem wieder voraus, dass es endlich sei.) Die Elemente von U nennen wir auch Individuen. – Des Weiteren muss festgelegt werden, was Prädikate im Realen sein sollen. Im obigen Beispiel haben wir nicht ganz korrekt Relationen verwendet und so vielleicht eine unterschwellige Assoziation hergestellt. Die realen Entsprechung eines Prädikatsymbols sei eine Wahrheitswertfunktionen, eine Funktion gleicher Stelligkeit über dem gewählten Universum, die in die Menge der Wahrheitswerte abbildet. – Ferner wird eine Abbildung υ benötigt, die jedem Prädikatssymbol eine Wahrheitswertfunktion gleicher Stelligkeit über U zuordnet. Das Paar $S = (U, \upsilon)$ nennen wir eine *Struktur*. Folgende Notation hat sich eingebürgert: Für ein Prädikatssymbol p schreibt man anstelle von $\upsilon(p)$ häufig p^S.

Nachdem der Kontext geschaffen ist, müssen auch einzelnen Instanzen eines Ausdrucks fixiert werden können. So wie der Wert einer Funktion von seinen Argumenten abhängt, so bestimmt sich der Wahrheitswert eines Ausdrucks funktional durch seine (freien) Variablen. Folglich müssen die Variablen konkreten Objekten zugeordnet werden. Eine Abbildung β von der Menge der Variablen in das Universum U heiße eine *Variablenbelegung* (in der Struktur S). Das Paar $\mathcal{I} = (S, \beta)$ nennen wir eine *Interpretation*. Aus technischen Gründen werden wir ebenfalls die Spezialisierung $\beta[\frac{v}{u}]$ benötigen: Sie bildet die Variable v auf das Element $u \in U$ ab und stimmt für alle anderen Variablen mit β überein.

Jetzt haben wir alle Hilfsmittel beisammen, um Ausdrücke bezüglich einer Interpretation auswerten zu können, ihnen einen Wahrheitswerte zuzuordnen. Die Zuordnung eines Wahrheitswertes zu einem Ausdruck bezüglich einer Interpretation $\mathcal{I} = (S, \beta)$ wird mit einer Funktion, der *Wahrheitswertzuordnefunktion* $W_{S,\beta}$ bewerkstelligt. Sie folgt in ihrer Definition dem syntaktischen, strukturellen Aufbau der Ausdrücke und sei erklärt durch

$$
\begin{aligned}
W_{S,\beta}(0) &:= 0 \\
W_{S,\beta}(1) &:= 1 \\
W_{S,\beta}\big(p(v_1 v_2 \dots v_n)\big) &:= p^S\big(\beta(v_1), \beta(v_2), \dots, \beta(v_n)\big) \\
W_{S,\beta}\big((v_1 = v_2)\big) &:= \begin{cases} 1, & \text{falls } \beta(v_1) = \beta(v_2) \\ 0, & \text{sonst} \end{cases} \\
W_{S,\beta}\big((\neg\, \varphi)\big) &:= \begin{cases} 1, & \text{falls } W_{S,\beta}(\varphi) = 0 \\ 0, & \text{sonst} \end{cases} \\
&= 1 - W_{S,\beta}(\varphi)
\end{aligned}
$$

$$W_{S,\beta}\big((\varphi \wedge \psi)\big) := \begin{cases} 1, & \text{falls sowohl } W_{S,\beta}(\varphi) = 1 \text{ als auch } W_{S,\beta}(\psi) = 1 \\ 0, & \text{sonst} \end{cases}$$
$$= \min\{W_{S,\beta}(\varphi), W_{S,\beta}(\psi)\}$$

$$W_{S,\beta}\big((\varphi \vee \psi)\big) := \begin{cases} 1, & \text{falls } W_{S,\beta}(\varphi) = 1 \text{ oder } W_{S,\beta}(\psi) = 1 \\ 0, & \text{sonst} \end{cases}$$
$$= \max\{W_{S,\beta}(\varphi), W_{S,\beta}(\psi)\}$$

$$W_{S,\beta}\big((\forall v\colon \varphi)\big) := \begin{cases} 1, & \text{falls } W_{S,\beta[\frac{v}{u}]}(\varphi) = 1 \text{ für alle } u \in U \text{ gilt} \\ 0, & \text{sonst} \end{cases}$$
$$= \min_{u \in U} \Big\{ W_{S,\beta[\frac{v}{u}]}(\varphi) \Big\}$$

$$W_{S,\beta}\big((\exists v\colon \varphi)\big) := \begin{cases} 1, & \text{falls ein } u \in U \text{ existiert mit } W_{S,\beta[\frac{v}{u}]}(\varphi) = 1 \\ 0, & \text{sonst} \end{cases}$$
$$= \max_{u \in U} \Big\{ W_{S,\beta[\frac{v}{u}]}(\varphi) \Big\}$$

$$W_{S,\beta}\big(\mathrm{TC}\ v_1, v_2 : \varphi)(v_3, v_4)\big) := \max_{\substack{(u_1,\ldots,u_n) \in U^n \text{ mit} \\ n \geq 2, \beta(v_3) = u_1, \beta(v_4) = u_n}} \min_{i \in \{1,\ldots,n-1\}} W_{S,\beta\left[\frac{v_1\ v_2}{u_i u_{i+1}}\right]}(\varphi)$$

3.6 Dreiwertige Prädikatenlogik und Einbettung

Bei der Berechnung der abstrakten Programmausführung werden Heapabstraktionen verwendet. Oben haben wir am Beispiel gesehen, dass sich für eine zweiwertige Logik hierbei schnell Grenzen auftun. Deshalb soll eine dreiwertige Logik verwendet werden. Neben den beiden klassischen Wahrheitswerten gibt es einen dritten, den wir als $\frac{1}{2}$ schreiben und als „undefiniert“ verstehen. Die Erweiterung der zweiwertigen Logik um einen weiteren Wahrheitswert erfordert nur einen geringen Aufwand. Im Syntaxteil muss zunächst das Alphabet um ein Zeichen $\frac{1}{2}$ erweitert werden. Fernen wird die Definition der Ausdrücke dahingehend erweitert, dass $\frac{1}{2}$ ein Ausdruck ohne freie Variablen ist. Die Semantik, die verwendet werden soll beruht auf Kleene. Bei der Wahrheitswertauswertung muss $W_{S,\beta}(\frac{1}{2}) := \frac{1}{2}$ ergänzt werden. Wenn man die Wahrheitswerte als (rationale) Zahlen ansieht, dann kann der Rest bestehen bleiben, wenn man jeweils nur die definierenden Formeln verwendet, die mittels Minimum und Maximum verwenden.

Auf der Menge der Wahrheitswerte ist in natürlicher Weise eine Ordnung durch den Grad der Wahrheit definiert. Wir verstehen Wahrheitswerte als (rationale) Zahlen, damit stimmt diese Ordnung mit der üblichen Ordnung auf den (rationalen) Zahlen

überein, weswegen wir dasselbe Zeichen verwenden können. Es gilt dann $0 \leq \frac{1}{2} \leq 1$. Die Menge der Wahrheitswerte lässt sich noch auf eine andere Weise ordnen. Die Wahrheitswerte 0 und 1 sind klassische Wahrheitswerte und haben einen bestimmten Wahrheitswert, sie heißen deshalb definite Wahrheitswerte. Den Wahrheitswert $\frac{1}{2}$ verstehen wir als undefiniert, als noch nicht bestimmt, er heißt deshalb indefinit. Der Grad an Bestimmtheit definiert die zweite Ordnung auf der Menge der Wahrheitswerte, die wir als $\sqsubseteq$ schreiben. Bezeichnet wird sie oft als Informationsordnung. Für sie gilt $\frac{1}{2} \sqsubseteq 0$ und $\frac{1}{2} \sqsubseteq 1$, die beiden definiten Wahrheitswerte 0 und 1 sind bezüglich $\sqsubseteq$ nicht vergleichbar.

Im Syntaxabschnitt haben wir beim Vokabular eine Menge P von Prädikatssymbolen eingeführt. In einer logischen Struktur S werden sie auf „reale Prädikate“ über dem Universum abgebildet, genauer auf Wahrheitswertfunktionen. Diese charakterisieren Eigenschaften für U, die wir zu beobachten wünschen. Wenn in einer Struktur S für jedes $p \in P$ die Wahrheitswertfunktion p^S als Wertebereich die Wahrheitswertemenge $\{0, 1\}$ hat, dann heiße S eine *zweiwertige (logische) Struktur*. Haben all diese Wahrheitswertfunktionen als Wertebereich die Wahrheitswertemenge $\{0, \frac{1}{2}, 1\}$, dann heiße S eine *dreiwertige (logische) Struktur*.

In dieser Arbeit haben wir stets von Shapegraphen geredet, ohne bisher explizit erklärt zu haben, was darunter zu verstehen sei. Das Wort wird in zwei Bedeutungen verwendet. In beiden Fällen geht es um die Beobachtung oder Beschreibung von Heapzuständen unter dem Blickwinkel vorgegebener Eigenschaften. Die erste Bedeutung von *Shapegraph* bezeichnet die allgemeine Idee, den Heap bezüglich dieser Beobachtungseigenschaften zu sehen beziehungsweise auszudrücken. Diese allgemeine Idee ist gänzlich unabhängig von irgendeiner speziellen Repräsentation. Jetzt haben wir mit den logischen Strukturen einen Begriff zur Beschreibung von Heapzuständen vorliegen. Als Basis dient ein Vokabular von Beobachtungsprädikaten, unter dessen Blickwinkel der Heap gesehen wird. Eine logische Struktur ist eine Realisierung der allgemeinen Idee des Ausdrücken. Der in dieser Arbeit beschriebene Ansatz zur Visualisierung stützt sich auf diese logische Strukturen, es werden keine anderen verwendet. Des Weiteren repräsentieren wir diese logische Strukturen als beschriftete gerichtete Graphen. Daher nennen wir auch eine logische Struktur, beziehungsweise seine Repräsentation als Graph, einen *Shapegraphen*. Das ist diejenige der beiden Bedeutungen, in der das Wort in dieser Arbeit hauptsächlich verwendet wird, als Synonym zu logische Struktur.

Mit den zweiwertigen und den dreiwertigen Strukturen haben wir zwei „Welten“ von Heapbeschreibungen. Wir benötigen einen Mechanismus um beide Welten miteinander in Beziehung zu setzen. Dazu definieren wir den Begriff der Einbettung. Formal stellt sie sich als eine binäre Relation auf der Menge der logischen Strukturen dar.

3.2 Definition. *Es seien $S = (U_S, \upsilon_S)$ und $T = (U_T, \upsilon_T)$ zwei logische Strukturen. Die Struktur S heiße* einbettbar *in T, wenn es eine Abbildung f von U_S auf U_T gibt*

– es handele sich bei f also um eine Surjektion –, so dass die folgenden Bedingungen erfüllt sind:

1. *Für jedes Prädikat (Prädikatssymbol) p, es habe Stelligkeit n, und für alle $u_1, \ldots, u_n \in U_S$ gelte*

$$p^S(u_1, \ldots, u_n) \sqsubseteq p^T\big(f(u_1), \ldots, f(u_n)\big),$$

   wobei $\sqsubseteq$ die Informationsordnung auf der Menge der Wahrheitswerte bezeichne.

2. *Für jedes $u_T \in U_T$ gelte*

$$(|\{u \in U_S \colon f(u) = u_T\}| > 1) \sqsubseteq sm^T(u_T).$$

Ist S in T einbettbar, so schreiben wir $S \sqsubseteq T$.

In der Definition wird die Surjektivität der die Einbettung vermittelnden Funktion verlangt. Hierauf können wir nicht verzichten. Wir werden in den späteren Untersuchungen in Abschnitt 7.1.2 auf die Einbettung zurückgreifen. Dort werden die Gründe genauer erläutert. Auch werden dort Beispiele zur Verdeutlichung angegeben und einige Eigenschaften der Einbettung herausgestellt. – Mittels des Begriffs der Einbettung lässt sich zeigen (Beweis in [Sagiv u. a. 2002]):

3.3 Satz. ***(Einbettungssatz).*** *Es seien $S = (U_S, \upsilon_S)$ und $T = (U_T, \upsilon_T)$ zwei dreiwertige Strukturen mit $S \sqsubseteq T$, wobei $f : U_S \to U_T$ die die Einbettung vermittelnde Surjektion sei. Dann gilt für jeden Ausdruck φ und jede vollständige Variablenbelegung β für φ die Ausage $W_{S,\beta}(\varphi) \sqsubseteq W_{T,f\circ\beta}(\varphi)$.*

Eine 2-wertige Struktur kann als 3-wertige aufgefasst werden, schließlich vergrößert sich nur der Wertebereich der Wahrheitsertefunktionen. Daher ist der Einbettungssatz auch richtig, wenn es sich bei der Struktur S um eine 2-wertige Struktur handelt. – Der Satz besagt, dass jede Information, die mittels einer Formel φ aus T herausextrahiert wird, eine konservative Approximation der Information ist, die mittels φ aus S herausextrahiert wird. Wertet φ in S zu einem definiten Wahrheitswert aus, dann wertet φ in T zu demselben definiten Wahrheitswert oder zu $\frac{1}{2}$ aus, wertet φ in S zu $\frac{1}{2}$ aus, dann auch in T.

Ein dreiwertiger Shapegraph ist eine Beschreibung eines abstrakten Heapzustandes, zweiwertige Shapegraphen beschreiben konkrete Heapzustände. Wir setzen:

3.4 Definition. *Es sei S ein Shapegraph. Es bezeichne $k(S)$ die Menge alle zweiwertigen Strukturen $S_{(2)}$ mit $S_{(2)} \sqsubseteq S$.*

Für einen Shapegraphen S kann die Menge $k(S)$ interpretiert werden als die Menge aller konkreten (zweiwertigen) Strukturen, die unter die Beschreibung von S fallen. Für einen zweiwertigen Shapegraphen S gilt $k(S) = \{S\}$. – Es seien S und

T zwei Shapegraphen mit $S \sqsubseteq T$. Wir werden in Abschnitt 7.1.2 zeigen, dass die Einbettungsrelation transitiv ist. Bei Ausnutzung der Transitivität gilt für jeden Shapegraphen $S_{(2)} \in k(S)$ damit $S_{(2)} \sqsubseteq T$, folglich besteht $k(S) \subseteq k(T)$. Diese Aussage lässt sich informal so interpretieren, dass die Beschreibung von T die Beschreibung von S umfasst.

3.7 Semantik von Programmaktionen

Bisher haben wir in erster Linie einzelne Heapzustände betrachtet. Bei der Visualisierung der Programmausführung hat man es mit einer durch das Programm bestimmten Folge von Heapzuständen zu tun. Für jedes Paar (h_1, h_2) aufeinanderfolgender Heapzustände ist der Übergang von h_1 zu h_2 durch eine Programmaktion bestimmt. Daher muss die Semantik von Programmaktionen ausgedrückt werden, wozu ebenfalls Logik verwendet wird. Betrachten wir zunächst die Situation für konkrete Heaps, also den zweiwertigen Fall. Die Heapzustände werden mittels logischer Strukturen beschrieben, für h_1 sei dies etwa S_1. Shapeanalyse arbeitet mit solchen logischen Strukturen, den Shapegraphen. Die Aktionen des Programms transformieren Shapegraphen in andere Shapegraphen. Der Shapegraph S_1 wird in einen Shapegraphen S_2 transformiert, der eine logische Beschreibung von h_2 ist.

Eine logische Struktur besteht neben dem Universum aus Prädikaten, wobei die Menge der Prädikate für alle Shapegraphen einer Analyse gleich ist. Die Transformation kann dadurch realisiert werden, dass man jedem Prädikat(ssymbol) p eine Funktion (gleicher Stelligkeit) zuordnet, die die Auswirkung der Aktion für das Prädikat p beschreibt. Diese Funktion wird durch eine logische Formel definiert und heißt *Prädikat-Update-Formel* für p. Die Situation im dreiwertigen Fall ist im Grunde genommen dieselbe wie im zweiwertigen. Es zeigt sich, dass dieselben Update-Formeln auch hier verwendet werden können. Um die Qualität der Analyse zu verbessern, um eine größere Präzision zu erhalten, ist es aber angeraten, die abstrakte Semantik zu verfeinern. Dies kann dazu führen, wie wir später sehen werden, dass die abstrakte Anwendung einer Aktion auf einen Shapegraphen eine Menge von Shapegraphen erzeugt.

Um eine Shapeanalyse durchführen, sind verschiedene Spezifikationen erforderlich: Zunächst benötigt man die Menge der Prädikatssymbole und für jede verwendete Aktion die Spezifikation der Transformation. Zweitens benötigt man eine Kodierung des Programms in Form eines Kontrollflussgraphen und drittens eine Menge von Startshapegraphen, die die Heapzustände bei Programmstart codieren. Shapeanalyse berechnet für jeden Programmpunkt die Menge, der an diesem Programmpunkt auftretenden Shapegraphen. Zusätzlich können die abstrakten Übergänge mit ausgegeben werden, so dass die abstrakte Programmausführung in Form eines Transitionsgraphen vorliegt. Bei Analysestart ist die zum Startprogrammpunkt gehörige Menge

Für jeden Programmpunkt n tue:
  $\mathcal{M}_n := \emptyset$
Iteriere bis sich die Familie der Mengen $\mathcal{M}_n$ nicht mehr verändert:
  Für jeden Programmpunkt n tue:
    Für jede TVLA-Aktion „n aktion n'“ tue:
      Für jeden Shapegraphen $S \in \mathcal{M}_n$ tue:
        – Wende den zu „aktion“ gehörenden Transformator auf S an, dieser erzeugt eine Menge $\mathcal{M}_S$ von Shapegraphen.
        – Für jedes $S' \in \mathcal{M}_S$ tue:
          Ist $S' \neq T$ bzw. $S' \not\simeq T$ für alle $T \in \mathcal{M}_{n'}$, dann:
            $\mathcal{M}_{n'} := \mathcal{M}_{n'} \cup \{S'\}$

Abbildung 3.2: Schematische Darstellung des Shapeanalyse-Algorithmus

mit den Startshapegraphen initialisiert, alle anderen Mengen sind leer. Die Berechnung lässt sich algorithmisch wie in Abbildung 3.2 darstellen.

Die einzige Art von Änderung, die der Algorithmus an den Shapegraphenmengen durchführt, ist das Hinzufügen von Shapegraphen. Ist man nicht nur an Shapegraphen, sondern an den abstrakten Übergängen interessiert, so speichere man jeweils (n_1, S) und (n_2, T). Wenn wir die äußere Schleife betrachten, dann wird in jeder Iteration wenigstens ein Shapegraph hinzugefügt. Der Algorithmus iteriert, bis keine Änderung mehr bei dem Mengensystem vorkommt, bis ein (kleinster) Fixpunkt erreicht ist. Die Terminierung der Analyse wird dadurch sichergestellt, dass bezüglich der Analysespezifikation nur eine endliche Anzahl von Shapegraphen existieren. Diese maximale Anzahl wird durch die Anzahl der Prädikate und durch die Abstraktion bestimmt. Sie ist insbesondere eine obere Schranke für die Anzahl der Shapegraphen an jedem Programmpunkt, also für die Größe jeder Shapegraphenmenge.

Für die abstrakte Ausführung einer Aktion auf einen Shapegraphen können im Prinzip dieselben Update-Formeln wie im konkreten Fall verwendet werden, also diejenigen, die in der Analysespezifikation angegeben sind. Jedoch führt dies in der Regel zu einem Qualitätsverlust: Beispielsweise können Wahrheitswerte ihre Definitheit verlieren, was einen geringeren Informationsgehalt (bezüglich der Informationsordnung) bedeutet. Aus diesem Grund wurden Mechanismen eingeführt, um die abstrakte Semantik präziser zu erfassen; die wichtigsten sind die Fokus- und die Coerce-Operation.

Fokus-Operation

Die zugrunde liegende Idee soll anhand eines Beispiels erläutert werden. Wir greifen auf das Einfügen eines Elements in einen Suchbaum zurück. Wir befinden uns in der Suchschleife und wollen vom aktuellen Element zu seinem linken Kind verzweigen. Als Erstes behandeln wir die Situation im konkreten, zweiwertigen Fall. Dazu

betrachten wir einen konkreten Suchbaum, etwa den in Abbildung 3.1 oder den in Abbildung 1.1 auf Seite 11. Der konkrete Heapzustand sei mittels einer zweiwertigen logischen Struktur S ausgedrückt. Das Individuum u_c sei dasjenige, für welches das Prädikat cur wahr ist, also diejenige Ecke, auf die der Zeiger cur zeigt; das linke Kind von u_c sei u_l. Die bei der Suche durch den Übergang zum linken Kind resultierende Heapstruktur unterscheided sich von der ursprünglichen nur dadurch, dass cur auf eine andere Heapzelle zeigt. Um dafür eine logischen Repräsentation T zu erhalten, braucht in S nur das Prädikat cur modifiziert zu werden. Dazu genügt es, die Wahrheitswerte für u_c und u_l so zu ändern, dass $W_{T,\beta[\frac{v}{u_c}]}\big(\mathsf{cur}(v)\big) = 0$ und $W_{T,\beta[\frac{v}{u_l}]}\big(\mathsf{cur}(v)\big) = 1$ gilt. Formal wird eine solche Modifikation durch eine Update-Formel (für das Prädikat cur) realisiert.

Shapeanalyse verwendet abstrakte Heapzustände. Behandeln wir daher jetzt den gleichen Vorgang im dreiwertigen Fall. Eine visuelle Darstellung des abstrakten Heaps könnte wie in Abbildung 1.2 aussehen. Der abstrakte Heapzustand sei durch eine dreiwertige logische Strukturen S repräsentiert. Die Ecke, auf die der Zeiger cur zeigt, sei wieder mit u_c bezeichnet, es handelt sich bei ihr um eine Individuenecke. Ihr linkes Kind ist in einer Summary-Ecke u_s „enthalten“. Genauso wie im konkreten Fall soll der bei der Suche durch den Übergang zum linken Kind resultierende Heapzustand T dadurch erhalten werden, dass in S die Wahrheitswerte von cur (für die beiden Ecken u_c und u_s) angepasst werden. Die Modifikation für u_c stellt kein Problem dar. Für die Summary-Ecke u_s ist aber nur $W_{T,\beta[\frac{v}{u_s}]}\big(\mathsf{cur}(v)\big) = \frac{1}{2}$ möglich, ein definiter Wahrheitswert wäre falsch. Die abstrakte Heapsituation T umfasst (im Sinne der Einbettung) alle konkreten Heapzustände, in denen eine Ecke existiert, die das linke Kind von u_c ist, also führte 0 zu einem Widerspruch. Andererseits kann es und in vielen dieser konkreten Heapzustände wird es weitere Ecken in dem Teilbaum geben, die natürlich nicht das linke Kind von u_c sind, also führte 1 zu einem Widerspruch. Die Anpassung der Wahrheitswerte für cur erfolgt durch eine Update-Formel, und das Problem liegt darin, dass sie für u_s zu $\frac{1}{2}$ auswertet.

Während im Shapegraphen S das Prädikat cur stets zu definiten Wahrheitswerten auswertet, würde jetzt in T ein undefiniter Wahrheitswert auftreten. In diesem Sinne entsteht ein Informations- oder Präzisionsverlust. Um einen solchen zu verhindern, muss dafür Sorge getragen werden, dass cur für alle Ecken zu definiten Wahrheitswerten auswertet. Die Fokus-Operation dient dazu, in solchen Fällen die Präzision aufrechtzuerhalten. Bei diesem Beispiel bewirkt sie bildlich gesprochen, dass das linke Kind aus dem Summary-Ecke herausmaterialisiert wird. Die ursprüngliche Summary-Ecke u_s wird aufgespalten und eine der entstehenden Ecken wird das linke Kind. Bei dem so entstehenden Shapegraphen lässt sich der Update der Prädikatswerte nun ohne Informationsverlust bewerkstelligen.

Weiter soll das Beispiel nicht vertieft werden, der interessierte Leser findet weiterführende, detaillierte Erklärungen in [Sagiv u. a. 2002] oder [Reps u. a. 2002]. Zu-

sammenfassend dient die Fokus-Operation also dazu, einen Update ohne Informationsverlust zu ermöglichen. Dies wird dadurch erreicht, dass man für eine angegebene Formel (für alle Variablenbelegungen) ihre Auswertung zu definiten Wahrheitswerten erzwingt. Wir erklären formal:

3.5 Definition. *Es sei $\mathcal{S}$ die Menge alle dreiwertigen Strukturen bezüglich einer zugrunde liegenden Analysespezifikation. Eine Funktion $f_{Fokus} : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ in die Potenzmenge von S heiße eine* Fokusoperation *für eine Formel φ, wenn*

1. *Der Shapegraph S und die Menge $f_{Fokus}(S)$ repräsentieren dieselben konkreten (zweiwertigen) Strukturen.*
2. *Für alle $T \in f_{Fokus}(S)$ gilt: Für jede Variablenbelegung β werte φ zu definiten Wahrheitswerten aus, also $W_{T,\beta}(\varphi) \in \{0, 1\}$.*

Sie heiße eine Fokusoperation für eine Menge F von Formeln, wenn sie für jede Formel $\varphi \in F$ eine Fokusoperation für φ ist.

Details des Algorithmus brauchen uns, die wir in Bezug auf Shapeanalyse eine Anwenderposition innehaben, nicht zu interessieren. Einzelheiten können in [Sagiv u. a. 2002] nachgesehen werden, ein verbesserter Fokusalgorithmus ist in [Lev-Ami 2000] beschrieben. Ein Punkt muss jedoch erwähnt werden, da er bei der Spezifikation von Shapeanalysen zu berücksichtigen ist. Wenn eine Fokusoperation auf einen Shapegraphen S angewendet wird, dann kann der Fall eintreten, dass die Bildmenge $f_{\text{Fokus}}(S)$ unendlich viele Shapegraphen enthalten kann. So etwas ist softwaremäßig nicht handhabbar. Daher wird die Fokusoperation auf eine Teilklasse der Formeln eingeschränkt, für die sichergestellt werden kann, dass die resultierenden Bildmengen endlich sind. Für die Anwendung bedeutet das, dass nicht jede Formel als Fokusformel verwendbar ist. Wenn wir später die Analysespezifikation von Listen und Bäumen entwickeln, werden wir diesem Problem begegnen. Oft lässt es sich durch das Umformulieren der Fokusformeln unter Hinzunahme geeigneter weiterer Prädikate zur Analysespezifikation umgehen. Wir werden dort aus diesem Grund beispielsweise für die Prädikate, welche die Zeiger der Heapelemente repräsentieren, explizit eine transitiv-reflexive Hülle als Prädikat formulieren.

Coerce-Operation, Instrumentationsprädikate

Bei den Ausführungen zu Beginn dieses Kapitels, vergleiche Abbildung 3.1, wollten wir den Teilbaum mit Wurzel cur auszeichnen. Dazu hatten wir ein zusätzliches Prädikat td („to do“) verwendet. Natürlich kann es jedesmal, wenn es benötigt wird, explizit als reflexiv-transitive Hülle der Formel $\mathsf{left}(v_1, v_2) \vee \mathsf{right}(v_1, v_2)$ formuliert werden. Es ist aber oft hilfreich, solche abgeleiteten Prädikate explizit in die Spezifikation aufzunehmen. Solche Prädikate zeichnen sich dadurch aus, dass sie durch eine Formel definiert sind, und sie heißen *Instrumentationsprädikate* (instrumentation predicates). Sie erlauben, Eigenschaften (des Heaps) in Form eines Prädikats prägnant zu erfassen. Zudem kann häufig verhindert werden, dass sie bei der Anwendung

einer Aktion an Präzision verlieren. Die Prädikate, die wir bisher verwendet haben, heißen zur Unterscheidung *Basisprädikate* (core predicates).

Allgemein gesprochen können Abhängigkeiten zwischen den im Shapegraphen gespeicherten Eigenschaften, also in den Prädikaten, vorliegen, die aber nicht notwendigerweise in den Update-Formeln für die Prädikate miteingeschlossen sind. Der Coerce-Mechanismus ist ein systematische Methode diese Abhängigkeiten zwischen Prädikaten zu erfassen und zu nutzen. Er beinhaltet unter anderem die Verwendung von Instrumentationsprädikaten, so wie sie oben vorgestellt wurden. Die Methode beruht auf dem folgenden

Verschärfungsprinzip. *Es sei S ein Shapegraph, p ein n-stelliges Instrumentationsprädikat und $u_1, \ldots, u_n$ Ecken von S.*

1. *Der in S gespeicherte Wahrheitswert $p^S(u_1, \ldots, u_n)$ sollte mindestens so präzise sein wie er sich durch Auswerten der p definierenden Formel φ ergibt, also wie $W_{S,\beta[\frac{v_1 \cdots v_n}{u_1 \cdots u_n}]}\big(\varphi(v_1, \ldots, v_n)\big)$.*
2. *Sind $p^S(u_1, \ldots, u_n)$ und $W_{S,\beta[\frac{v_1 \cdots v_n}{u_1 \cdots u_n}]}\big(\varphi(v_1, \ldots, v_n)\big)$ verschiedene definite Wahrheitswerte, dann werden durch S keine konkreten Strukturen repräsentiert.*

Tritt in einem Shapegraphen S für ein Prädikat p und Ecken $u_1, \ldots, u_n$ der erste Fall ein, dann kann der in S gespeicherte Wert $p^S(u_1, \ldots, u_n)$ durch den Wahrheitswert $W_{S,\beta[\frac{v_1 \cdots v_n}{u_1 \cdots u_n}]}\big(\varphi(v_1, \ldots, v_n)\big)$ verschärft werden. Tritt der zweite Fall ein, dann kann der gesamte Shapegraph S als unzulässig verworfen werden. Dieses Prinzip wird bei der Shapeanalyse systematisch verwendet. Zu Analysebeginn werden aus der Spezifikation automatisch Kompatibilitätsbedingungen („compatibility constraints“) abgeleitet, die gemäß dem Verschärfungsprinzip verwendet werden. Neben den definierenden Formeln von Instrumentationsprädikaten werden weitere spezifizierbare Charaktierisierungen berücksichtigt. Beispielsweise kann bei einem einstelligen Prädikat explizit festgelegt werden, dass es nur für höchstens eine Ecke wahr sein kann; dies korrespondiert zu der Tatsache, dass Zeiger auf höchstens eine Heapzelle zeigen können. Für weiterführende Details zum Coerce-Mechanismus sei der Leser auf [Sagiv u. a. 2002] verwiesen.

3.8 TVLA

Zur Durchführung von Shapeanalysen existiert die Software TVLA, siehe [TVLA 2007], die den Shapeanalysealgorithmus umsetzt. Bei der Realisierung dieses Softwareprodukts wurden einige Einschränkungen vorgenommen:

- Es werden nur Prädikate bis zur Stelligkeit zwei unterstützt.
- Alle Programme müssen intraprozedural sein.

Als *Eingabe* für die Analyse benötigt TVLA:

- eine Spezifikation, die die logische Beschreibung der Prädikate und Programmaktionen umfasst
- eine Codierung des Programms in Form eines Kontrollflussgraphen
- Startshapegraphen, welche die Heapzustände bei Programmstart angeben

Wir werden uns in Kapitel 4 dem Aspekt der Spezifikation ausführlicher widmen und dort auch Aspekte besprechen, die beim praktischen Umgang mit TVLA zu beachten sind.

Die *Ausgabe* von TVLA besteht standardmäßig aus einer Menge von Shapegraphen für jeden Programmpunkt. Für einen Programmpunkt n bezeichne $\mathcal{S}_n$ die Menge der zum Programmpunkt n berechneten Shapegraphen. Formal besteht die Ausgabe aus einer Liste von Paaren $(n, \mathcal{S}_n)$ so, dass zu jedem Programmpunkt n ein Paar $(n, \mathcal{S}_n)$ enthalten ist. Die Mengen $\mathcal{S}_n$ können gegebenenfalls leer sein. – Alternativ ist auch die zusätzliche Ausgabe der (abstrakten) Übergänge, der Transitionen, in Form eines Transitionsgraphen möglich. Bisher haben wir normalerweise Shapegraphen an einem festen Programmpunkt betrachtet. Wenn wir es mit Shapegraphen zu tun haben, die zu verschiedenen Programmpunkten gehören, dann muss sich die Zugehörigkeit zum Programmpunkt auch formal widerspiegeln. Wir setzen:

3.6 Definition. *Tritt bei der Shapeanalyse am Programmpunkt n der Shapegraph S auf, dann nennen wir das Paar (n, S) einen* abstrakten Zustand *der Programmausführung.*

Alternativ könnten wir die Standardausgabe auch als eine einzelne Menge von abstrakten Zuständen ansehen, beide Repräsentationen sind zueinander isomorph. – Wenn wir in der Analyseausgabe Transitionsinformationen benötigen, dann verwenden wir die Ausgabe in Form eines Transitionsgraphen.

3.7 Definition. *Der* Transitionsgraph *zu einer Shapeanalyse sei ein mit Kantenbeschriftungen versehener gerichteter Graph. Seine Ecken seien die bei der Analyse auftretenden abstrakte Zustände (n, S). Zwei abstrakte Zustände (n, S) und (n', S') seien durch eine gerichtete Kante miteinander verbunden, wenn es eine Programmaktion zwischen den Programmpunkten n und n' gibt und bei der abstrakten Programmausführung ein Übergang von (n, S) zu (n', S') stattfand. Die Kante sei mit der TVLA-Aktion beschriftet.*

Genauere technische Details, die über das Verständnis der Analysemethode und der Anwendung der Software hinausgehen, brauchen im Hinblick auf eine Visualisierung der Ausgabe nicht zu interessieren. Der interessierte Leser findet Startpunkte für weiterführende Darstellungen in [Lev-Ami u. a. 2007; Lev-Ami u. Sagiv 2000; Lev-Ami 2000]. – Es soll an dieser Stelle noch einmal die Anwendung einer Programmaktion auf einen Shapegraphen S angesprochen werden. Sie setzt

sich aus einer Reihe von Teilaktionen zusammen. Diese sind Fokus, Precondition, Update, Coerce und Blur, die defaultmäßig in dieser Reihenfolge angewendet werden.

Fokus: Zu jeder Aktion kann eine Menge von Fokusformeln angegeben werden. Ist diese Menge nicht leer, dann wird auf den Shapegraphen S die Fokusoperation angewendet. Es wird eine endliche Menge von Shapegraphen berechnet, die 1. die gleichen konkreten Heapzustände wie S beschreibt und in der 2. jede Fokusformel (für alle Variablenbelegungen) zu einem definiten Wahrheitswert auswertet.

Precondition: Sie verhält sich wie ein Filter. Es besteht die Möglichkeit, einer Aktion eine Formel als Bedingung mitzugeben. Handelt es sich um eine geschlossene Formel, die zu 0 auswertet, dann wird die Anwendung der Aktion auf den aktuellen Shapegraphen abgebrochen, bei 1 wird sie normal fortgesetzt. Hiermit können Verzweigungen realisiert werden.

Update: An dieser Stelle werden die Prädikatswerte geändert. In der Spezifikation (der Programmaktionen) ist zu jedem Prädikat(ssymbol) eine Update-Formel angegeben. Sie werden hier (für alle Variablenbelegungen) ausgewertet und als neue Prädikatswerte zugewiesen.

Coerce: Es werden Kompatibilitätsbedingungen ausgewertet. Damit können Wahrheitswerte verschärft werden. Auch werden nicht zulässige Shapegraphen verworfen, welche bei der Fokusoperation entstehen können.

Blur: Die Anwendung der Fokusoperation führt oft dazu, dass Summary-Ecken in mehrere Ecken aufgespalten werden, es entstehen also gewissermaßen „neue“ Ecken. Bei der Anwendung der Update- und Coerce-Operation können sich Prädikatswerte ändern. Dies kann dazu führen, dass der Shapegraph nicht mehr die korrekte Abstraktion besitzt. Bei der Erklärung der Abstraktion von Heapzuständen haben wir die kanonische Abstraktion vorgestellt. Sie ist auch diejenige, die wir im Rahmen der Visualisierung normalerweise verwenden, jedoch sind in TVLA zusätzlich noch weitere Abstraktionsoperatoren implementiert. Im Falle der kanonischen Abstraktion können nach Anwendung der Fokus-, Update- und Coerce-Operation jetzt also Ecken u_1 und u_2 existieren, so dass alle Abstraktionsprädikate für u_1 und u_2 zum gleichen Wahrheitswert auswerten. Die Blur-Operation stellt die korrekte Abstraktion durch Anwendung des Abstraktionsoperators wieder her.

Das Softwareprojekt TVLA wird stetig gepflegt und weiterentwickelt. Verbesserungen und Erweiterungen, sowohl der Analysemaschine als auch von Aspekten, die mit der Analyse einhergehen, wirken sich auch auf unsere Anwendung positiv aus. Eine Übersicht über den Stand der Dinge im Jahr 2004 gibt [Lev-Ami u. a. 2004], weitere und neuere Entwicklungen sind auf der Projektseite [TVLA 2007] vermerkt. So beschäftigt man sich beispielsweise mit der automatischen Generierung von In-

strumentationsprädikaten und mit der Analyse nebenläufiger Programme, auch hat man sich der interprozedurale Shapeanalyse angenommen.

4 Praktische Aspekte der Shapeanalyse

> Die Praxis sollte das Ergebnis des
> Nachdenkens sein, nicht umgekehrt.
>
> *(Hermann Hesse)*

Im vorangegangenen Kapitel 3 haben wir uns den logischen Aspekten der Shapeanalyse, insbesondere den logischen Grundlagen, gewidmet. In diesem Kapitel ergänzen wir die Erläuterungen zur Shapeanalyse um Aspekte ihrer Anwendung. Dabei konzentrieren wir uns auf solche, die im Hinblick auf die Visualisierung relevant sind. Zur Durchführung von Shapeanalysen existiert das Softwarepaket TVLA. Primär behandeln wir Aspekte, die um die Spezifikation von Shapeanalysen für TVLA zentriert sind. Unter einem anwendungsorientiertem Blickwinkel können diese Teile auch als praktische Erläuterung und ausführliches Beispiel zur Shapeanalyse gelesen werden. Dabei waren wir bestrebt, sie möglichst ohne allzu großen Rückgriff auf die logischen Grundlagen zu formulieren.

In dieser Arbeit betrachten wir in den Beispielen in erster Linie Algorithmen, die auf Bäumen operieren. Daher behandeln wir in Abschnitt 4.1 die Spezifikation der Datenstruktur Baum und als Vorstufe die der Datenstruktur Liste. Im darauf folgenden Abschnitt betrachten wir Analyseausgaben, die sich ergeben, wenn man Shapeanalysen mit den vorgestellten Spezifikationen durchführt, unter einem quantitativen Blickwinkel. Bei der Algorithmenvisualisierung geht es um die Visualisierung der Programmausführung, wobei diese durch das Programm bestimmt ist. Damit wirkt das Programm auch auf die Visualisierung. Die Kodierung des Programms beeinflusst, wie gut die mit der Visualisierung angestrebten Ziele erreicht werden können. Aus diesem Grund definieren wir in Abschnitt 4.3 eine Normalform von TVLA-Programmen.

4.1 Spezifikation von Shapeanalysen

Eine Analysespezifikation besteht aus mehreren Teilen. Sie besteht zum einen aus einer Menge von Prädikaten, die Eigenschaften des Heaps, der Heapzellen und zwischen Heapzellen beschreiben. Diese Prädikate bestimmen die verwendete Abstraktion der Datenstruktur, beziehungsweise deren konkreten Instanzen, und damit die Abstraktion des Heaps. Sie bestimmen, welche Eigenschaften in den Shapegraphen

ausgedrückt und unterschieden werden. Weiter enthält die Spezifikation Operationen – in TVLA-Terminologie *actions* –, sie transformieren Shapegraphen in andere und entsprechen den Anweisungen und Ausdrücken in Programmen.[1] Schließlich sind Startshapegraphen erforderlich, sie beschreiben die Heapsituationen zu Beginn der Programmausführung. Unser Hauptaugenmerk liegt auf der Abstraktion. Sie bestimmt, wie Shapegraphen aussehen, was für Visualisierungszwecke ein dominanter Aspekt ist.

Um uns im Folgenden auf die spezifizierenden Elemente für eine Shapeanalyse zu beziehen, verwenden wir die Wörter *Abstraktion* und *(Analyse-)Spezifikation*. Das Wort Abstraktion bezeichnet gemäß seiner durch die Wortherkunft gegebenen Bedeutung sowohl einen Vorgang für das Weglassen von Einzelheiten und der Überführung in etwas Allgemeine(re)s als auch das Resultat dieses Vorgangs. Die bei der Shapeanalyse berechneten Shapegraphen sind das Resultat eines solchen Prozesses, sie verstehen wir deshalb als Abstraktionen von Heapzuständen. Wie sich im vorangegangenen Absatz vielleicht schon andeutete, verwenden wir das Wortes Abstraktion in diesem Kapitel allgemeiner als es in der Shapeanalyseterminologie gewöhnlich üblich ist. Wir verstehen hier darunter auch die zugrunde liegende Beschreibung, welche die abstrakte Beschreibung der Heapsituationen bestimmt, also letztendlich die Prädikate. Dies beinhaltet sowohl die Eigenschaften, bezüglich derer der Heap betrachtete wird, als auch die Auswahl, bezüglich welcher Eigenschaften Heapzellen zusammengefasst werden (Abstraktionsprädikate). Gelegentlich sprechen wir auch von der Abstraktion einer Datenstruktur, um klarzustellen, dass wir uns auf alle Prädikate und nicht nur auf die Abstraktionsprädikate beziehen. Wenn wir im Folgenden von einer Spezifikation sprechen, dann meinen wir je nach Kontext die Gesamtheit der Elemente, die der Spezifikation der Datenstruktur dienen, also Prädikate und Aktionen, oder alle für eine Shapeanalyse benötigten Elemente. Dann gehören auch der Kontrollflussgraph des zu analysierenden Programms und die Startshapegraphen dazu. Wenn Unklarheiten zu befürchten sind, sprechen wir in ersterem Falle auch von der Spezifikation der Datenstruktur.

4.1.1 Listen

In diesem Abschnitt soll ein konkretes Verständnis für die Abstraktion von zeigerbasierten Datenstrukturen und der Spezifikation von Analysen für Programme, die auf ihnen operieren, gewonnen werden. Listen, und speziell einfach verkettete Listen,

1. Genaugenommen enthält eine Spezifikation auch Konsistenzregeln. Sie dienen bei der Coerce-Operation zum Verschärfen von Wahrheitswerten und zum Verwerfen von bei der Fokusoperation erzeugten unzulässigen Shapegraphen. Aus den zwischen den Prädikaten bestehenden Abhängigkeiten werden Konsistenzregeln automatisch abgeleitet. Es besteht darüber hinaus die Möglichkeit, von Hand weitere hinzuzufügen. Da sie auf die Qualität der Analyse und damit auf die Qualität der produzierten Shapegraphen wirken, nicht jedoch auf deren (durch die Prädikate bestimmte) Struktur, wollen wir sie hier unberücksichtigt lassen.

stellen die einfachsten zeigerbasierten Datenstrukturen dar. Kompliziertere zeigerbasierte Datenstrukturen lassen sich häufig als Erweiterung von Listen auffassen; bei Bäumen ist dies der Fall. In diesem Sinne stellen Listen eine (gemeinsame) Basis der zeigerbasierten Datenstrukturen dar. Für uns dient dieser Teilabschnitt also auch schon als Einführung in die Spezifikation von Shapeanalysen für Programme, die auf Bäumen operieren; ihnen werden wir uns im folgenden Teilabschnitt zuwenden.

Bei der Entwicklung der Shapeanalyse und von TVLA war die Untersuchung von Algorithmen, die auf Listen operieren, ein beliebter Forschungsgegenstand. Die Gründe sind aus dem einleitenden Absatz ersichtlich. Insbesondere ist es angeraten, Listen verstanden zu haben, bevor man sich komplizierteren, von Listen abgeleiteten Datenstrukturen (und Algorithmen auf ihnen) zuwendet. Die Untersuchung von Listen im Kontext der Shapeanalyse beginnt mit den Arbeiten [Sagiv u. a. 1996, 1998] und setzt sich, um nur einige Arbeiten beispielhaft anzuführen, über [Sagiv u. a. 1999; Reps u. a. 2002; Sagiv u. a. 2002; Reps u. a. 2004; Manevich u. a. 2005] fort. Diese Untersuchungen und damit auch die dort entwickelten und verwendeten Analysespezifikationen waren von theoretisch geprägten Zielsetzungen motiviert.

Es war ein grundlegendes Anliegen nachzuweisen, dass in einem Programm keine NULL-Zeiger dereferenziert werden und dass keine Löcher im Heap (memory leakage) entstehen, das heißt, dass keine Heapelemente von den Zeigervariablen unerreichbar werden. Ein weiteres Anliegen war festzustellen, ob mehrere Zeiger auf ein Heapelement zeigen, ob Heapelemente „shared“ sind. Später, siehe zum Beispiel [Lev-Ami u. a. 2000], begann man Prädikate zu studieren, die Ordnungseigenschaften der in den Heapelementen enthaltenen Datenwerte beschreiben. Damit ist es beispielsweise möglich nachzuweisen, dass die Ausgabe eines Algorithmus eine Liste ist, deren Datenelemente aufsteigend oder absteigend sortiert sind, und dass die Ausgabeliste eine Permutation der Eingabeliste ist. Die so gewonnenen Aussagen sind stark genug, um die partielle Korrektheit von Sortieralgorithmen, die einfach verkettete Listen verwenden, nachzuweisen.

Die Struktur der Heapzellen wird in einer Programmiersprache durch die verwendete Datenstruktur bestimmt. Wir setzen hier voraus, dass alle Listenelemente dieselbe Struktur besitzen. Ein *Listenelement* enthalte zum Ersten einen Datenwert eines festen Datentyps und zum Zweiten einen Zeiger auf ein Listenelement. Auf die Datenkomponente verweisen wir mit dem Wort(teil) *data* und auf den Zeiger mit *next*. Beim Typ des Datenwertes denken wir am besten an einen Zahlentyp, damit wir sofort eine intuitive Vorstellung haben, was eine Ordnungsrelation auf diesem Typ bedeuten soll. Anstelle von einer Heapzelle sprechen wir auch von einem Heapelement. Heapzellen stellen die feinsten Einheiten dar, die wir auf dem Heap unterscheiden.

Im Folgenden entwickeln wir eine Abstraktion für einfach verkette Listen. Bei den zu spezifizierenden Prädikaten lassen wir uns von einem Beispielalgorithmus inspirieren

```
cur := start                                  n1  Copy_Var_L(cur, start)        n2
while cur ≠ NULL do                           n2  Is_Null_Var(cur)              notfound
   if cur.data = el.data                      n2  Is_Not_Null_Var(cur)          n3
   then
      return cur                              n3  Equal_Data_L(cur, el)         found
   else                                       n3  Not_Equal_Data_L(cur, el)     n4
      cur := cur.next                         n4  Get_Next_L(cur, cur)          n2
od
return NULL
```

Abbildung 4.1: Algorithmus zur Suche eines Elements in einer einfach verketteten Liste: Pseudocode (links) und TVLA-Programm (rechts)

und von unseren Vorstellungen und Wünschen leiten, wie Shapegraphen in diesem Beispiel aussehen sollen. Wir betrachten dazu die Suche nach einem Element in einer (einfach verketteten) Liste. Der Algorithmus ist in Abbildung 4.1 abgebildet; links sehen wir ihn als Pseudocode, rechts als TVLA-Kontrollflussgraphen. Die Syntax des Kontrollflussgraphen kann in [Lev-Ami u. a. 2007] nachgesehen werden. Beim Startelement beginnend durchläuft der Zeiger cur den vom Startelement aus erreichbaren Teil des Heaps und vergleicht den Datenwert des aktuellen Heapelements mit dem eines Referenzelements, auf das der Zeiger el zeigt. Das Programm endet, wenn entweder ein Element mit gleichem Datenwert gefunden wurde oder wenn der vom Startelement aus erreichbare Teil des Heaps (erfolglos) durchlaufen wurde. Der Listensuchalgorithmus gehört zu den einfachsten Algorithmen (auf Listen), da die Struktur des Heaps nicht verändert wird. Ist der vom Startelement aus erreichbare Teil des Heaps zu Beginn des Programmlaufes eine Liste, dann ist er es auch bei Terminierung des Algorithmus. Solch ein Algorithmus stellt keine übermäßig hohen Anforderungen an eine Shapeanalyse. Für unsere Zwecke ist er aber ausreichend. Er erlaubt uns, auf Charakteristiken hinzuweisen, die im Zuge der Visualisierung auf der Basis von Shapegraphen zu berücksichtigen sind.

Wie wir oben dargestellt haben, war die Motivation für die Entwicklung der Heapabstraktionen und der Anaylsespezifikationen in den zuvor genannten Arbeiten eine theoretische. Unsere Prioritäten sind andere: Für uns steht die Aussagekraft der resultierenden Shapegraphen im Hinblick auf die Visualisierungsziele, also zum Beispiel im Hinblick auf lernunterstützende Prozesse, im Vordergrund. Häufig drücken Prädikate aus Spezifikationen, die aus theoretischen motivierten Zielsetzungen entwickelt wurden, starke und signifikante Eigenschaften aus, die auch im Hinblick auf die Visualisierungsziele eine starke Aussagekraft haben. Im Falle von Listen stimmt die aus theoretischen Motiven entwickelte Spezifikation gut mit der von uns für die Visualisierung gewünschten überein. Daher ist die hier präsentierte Abstraktion der in den Listenbeispielen von TVLA mitgelieferten „Standardspezifikation“ sehr ähnlich. Allerdings verwenden wir unter anderem zum Teil andere Bezeichnungen. Die Teile der Spezifikation, die wir in dieser Beschreibung übergehen, können aus den

oben genannten Arbeiten ergänzt werden.

Ein Anwender kann den in dieser Arbeit präsentierten Ansatz der Algorithmenvisualisierung mit ganz verschiedenen Zielsetzungen verwenden. Er kann zum Beispiel über keine oder nur geringe Vorkenntnisse in Datenstrukturen und Algorithmen verfügen und die Visualisierung zum Zwecke des Lernens, des Verstehenwollens eines Algorithmus verwenden. Er kann auch tiefe Vorkenntnisse haben und bei einem fehlerhaften Algorithmus den Fehler finden wollen. Oder er kann etwa eine neue Analysespezifikation entwickeln und austesten wollen. Sowohl der Stand der Vorkenntnisse als auch die konkrete Absicht kann variieren. Wir gehen in diesem und in den meisten Beispielen in dieser Arbeit davon aus, dass wir eine Spezifikation im Hinblick auf Lernvorgänge für einem Anwender mit höchstens leicht fortgeschrittenen Kenntnissen der zugrunde liegenden Datenstruktur und der Algorithmen wünschen. Ein fortgeschrittener Anwender, der spezielle Aspekte eines Algorithmus untersuchen will, mag ganz andere Abstraktionen wünschen. Für die intentionierte Zielgruppe und Anwendungszweck hat sich die folgende Eigenschaft als verheißungsvoller Ausgangspunkt erwiesen:

4.1 Vereinbarung (Strukturerhaltungsforderung). Eine Analysespezifikation werde *strukturerhaltend* genannt, wenn für jeden Shapegraphen bezüglich dieser Spezifikation gilt, dass die graphentheoretische Struktur des Shapegraphen, der zuvor auf die den Zeigern entsprechenden Prädikate eingeschränkt wird, die Zeigerstruktur des Heaps widerspiegelt.

In dieser Forderung haben wir keine präzise Eigenschaft formuliert, die eine zweifelsfreie Zuweisung der Strukturerhaltungseigenschaft, oder gar eine quantitative Beurteilung des Grades, erlaubte. Auch beinhaltet sie eine subjektive Komponente: Es hängt vom Kenntnisstand eines Betrachters ab, ob er in einem Shapegraphen die Zeigerstruktur eines konkreten Heaps angemessen repräsentiert sieht. Daher haben wir diese Forderung nicht als Definition notiert. Sie ist als qualitative Charakterisierung gemeint, die bei der Erstellung und Auswahl von Analysespezifikation leiten soll.

Ein Shapegraph besteht aus einem Universum sowie aus null-, ein- und zweistelligen Prädikaten (samt ihrer Belegung mit Wahrheitswerten).[2] Die Elemente des Universums, seine Ecken, repräsentieren einzelne konkrete Heapzellen oder im Falle von Summary-Ecken Mengen von konkreten Heapzellen. Zu welcher Ecke im Shapegraphen eine (konkrete) Heapzelle gehört, wird durch die Wahrheitswerte der einstelligen Prädikate für diese Ecke bestimmt. Wir verwenden bei der Shapeanalyse

2. Im Kapitel 3 haben wir logische Strukturen als Paare erklärt, die aus einem Universum und einer Funktion bestehen. Die Funktion bildet Prädikatssymbole auf Wahrheitswertfunktionen ab. Üblicherweise wird auf die explizite Angabe der Funktion verzichtet und es werden stattdessen die Wahrheitswertfunktionen direkt angegeben, die Funktion wird also mit ihren Bildern identifiziert.

üblicherweise die kanonische Abstraktion. In diesem Fall bestimmen die Wahrheitswerte der Abstraktionsprädikate die Ecke. Zu jeder Belegung der Abstraktionsprädikate mit (definiten) Wahrheitswerten gibt es in jedem Shapegraphen höchstens eine Ecke. Gibt es zu einer (potentiellen) Belegung keine konkrete Heapzelle, die sie erfüllt, dann existiert keine diese Belegung repräsentierende Ecke im Shapegraphen. Existiert genau eine Heapzelle, die sie erfüllt, dann handelt es sich im Shapegraphen um eine Individuenecke, sind es mehrere Heapzellen, dann ist sie eine Summary-Ecke.

Die nullstelligen Prädikate beschreiben Eigenschaften des ganzen Shapegraphen; in unserer Spezifikation werden wir sie nicht benötigen. Einstellige Prädikate beschreiben Eigenschaften von Heapzellen, hier also der Listenelemente. Defaultmäßig sind alle einstelligen Prädikate Abstraktionsprädikate. Das ist auch bei der hier dargestellten Spezifikation der Fall. Ob in einem Shapegraphen eine Ecke eine Individuen- oder eine Summary-Ecke ist, wird durch das unäres Prädikat sm (von summary oder summarisation) angezeigt: es ist für Individuenecken 0 und für Summary-Ecken $\frac{1}{2}$, es ist aber niemals 1. Die zweistelligen Prädikate beschreiben Eigenschaften zwischen Heapzellen, zwischen den Listenelementen. Diejenige (binäre) Eigenschaft, die wir in jedem Fall repräsentieren müssen, ist die Zeigerstruktur der Heapelemente wie sie durch die next-Komponenten der Listenelemente bestimmt ist. Daher führen wir ein binäres Prädikat next ein, das diese Zeigerstruktur ausdrückt. Andere zweistellige Prädikate benötigen wir im Prinzip nicht.

TVLA erlaubt in den Formeln die Verwendung eines Operators $*$, der die reflexiv-transitive Hülle eines Prädikats repräsentiert. So ist zum Beispiel in einem Shapegraphen $\mathsf{next}*(v_1, v_2)$ für zwei nicht notwendig verschiedene Ecken v_1 und v_2 wahr, wenn es in dem Shapegraphen einen (gerichteten) Weg von v_1 zu v_2 gibt, der nur aus next-Kanten besteht, oder wenn $v_1 = v_2$ gilt. Aus technischen Gründen ist es oft wünschenswert, hierfür ein eigenes Prädikat zur Verfügung zu haben. Wenn bei der Anwendung von TVLA-Aktionen während der Shapeanalyse auf Formeln, die den Hüllenoperator verwenden, fokussiert wird, dann führt dies sehr häufig zu einem Abbruch der Analyse. Daher definieren wir zusätzlich ein binäres Instrumentationsprädikat nextStar durch $\mathsf{nextStar}(v_1, v_2) := \mathsf{next}*(v_1, v_2)$ mit der beigefügten Eigenschaft „reflexive transitive“.

Zunächst ist es ausgehend von der Strukturerhaltungsforderung wünschenswert, dass die Heapzellen, auf die Zeiger deuten, Individuenecken sein sollen. Daher führen wir für jede im Programm auftretende Zeigervariable ein einstelliges Prädikat mit der zusätzlichen Eigenschaft „unique“, es kann nur für höchstens eine Ecke wahr sein, ein. Auf diese Weise erhalten wir die Prädikate start, cur und el[3]. Um die

3. Das Prädikat el benötigen wir nur aus technischen Gründen. Auf diese Weise benötigen wir nur einen Gleichheitstest für die Datenkomponenten von Listenelementen. Ansonsten müssten wir den Wert einer Komponente eines Listenelements mit einer Programmvariablen vergleichen, was insgesamt weniger konsistent ist und zusätzlichen Spezifikationsaufwand verlangt.

weiteren einstelligen Prädikate einzuführen, stellen wir uns vor, dass der Suchalgorithmus bereits einige (wenigstens drei) Iterationen der Schleife vollführt hat. Die (konkrete) Liste hat dann folgende Struktur: Der Zeiger start zeigt auf den Anfang der Liste; es folgen einige Listenelemente, danach folgt das Listenelement auf das der Zeiger cur zeigt und es folgt der Rest der Liste. Letzteres sind die Listenelemente, die noch nicht (von cur) besucht wurden. Durch diese Eigenschaft unterscheiden sie sich vom Anfang der Liste. Dort sind die Elemente, die (von cur) schon besucht wurden. In diesem Sinn repräsentieren sie die Geschichte der Programmausführung. Daher ist es wünschenswert, wenn schon besuchte und noch nicht besuchte Listenelemente im Shapegraphen durch verschiedene Ecken dargestellt werden. Es gibt für uns aber keine Gründe, die noch nicht besuchten Listenelemente voneinander zu unterscheiden. Sie können zu einer Summary-Ecke zusammengefasst werden. Ebenso brauchen wir die schon besuchten Listenelemente zwischen start und cur nicht zu unterscheiden.

Als aussagestarke Prädikate für diese Unterscheidung haben sich Prädikate herausgestellt, die Erreichbarkeitseigenschaften ausdrücken. Für jede Zeigervariable z führen wir generisch ein einstelliges Instrumentationsprädikat reach[z] ein, wir definieren $\mathsf{reach}[\mathsf{z}](v) := \exists v_1 \colon \mathsf{z}(v_1) \wedge \mathsf{nextStar}(v_1, v)$. Es ist genau für diejenigen Heapzellen v wahr, die von der Zeigervariablen z aus unter Verwendung der next-Zeiger erreichbar sind. Auf diese Weise erhalten wir die Prädikate reach[start], reach[cur] und reach[el]. In unserem Beispiel in von ihnen nur reach[cur] wirklich aussagestark. Alle Elemente der Liste sind von start aus erreichbar, aber nur das aktuelle Element und der Rest der Liste sind von cur aus erreichbar. Damit können der Anfangs- und der Endteil der Liste unterschieden werden.

Ändern wir nun unseren Blickwinkel. Wir haben eine Reihe von Prädikaten eingeführt und wollen überprüfen und untersuchen, welche Listenteile unterschieden werden. Dazu durchlaufen wir die Wahrheitswertbelegungen der Abstraktionsprädikate und untersuchen, welche Eigenschaften die ihnen entsprechenden Ecken der Shapegraphen haben. Wir interessieren uns dabei nur für die Liste, für die von start aus erreichbaren Heapzellen. Alle mit el zusammenhängenden Prädikate lassen wir unbeachtet. Des Weiteren ist reach[start] für alle diese Listenelemente wahr, weswegen wir hierauf auch verzichten. Es verbleiben die Prädikate start, cur und reach[cur]. In der folgenden Auflistung notieren wir die Wahrheitswertbelegungen als Tripel, wobei deren Komponenten in dieser Reihenfolge stehen. Für alle Belegungen, die nicht dargestellt sind, kann es keine (konkreten) Listenelemente geben, die zu diesen Wahrheitswerten auswerten.

$(1, -, -)$**:** das Listenelement, auf das der Zeiger start zeigt, der Listenanfang; es gibt zwei Fälle:

- $(1, 1, 1)$**:** der Zeiger cur zeigt ebenfalls auf dieses Listenelement
- $(1, 0, 0)$**:** der Zeiger cur zeigt nicht auf den Listenanfang

$(-, 1, 1)$**:** das Listenelement, auf das der Zeiger cur zeigt, das aktuelle Listenelement; es gibt zwei Fälle:

$(1, 1, 1)$**:** cur zeigt auf den Listenanfang, siehe oben

$(0, 1, 1)$**:** cur ist vom Listenanfang verschieden

$(0, 0, 1)$**:** die Listenelemente, die vom Zeiger cur aus erreichbar sind, aber nicht das Element, auf das cur zeigt; der noch nicht besuchte Teil der Liste

$(0, 0, 0)$**:** alle Listenelemente, auf die keiner der beiden Zeiger start und cur zeigt und die nicht von cur aus erreichbar sind; die Listenelemente zwischen start und cur

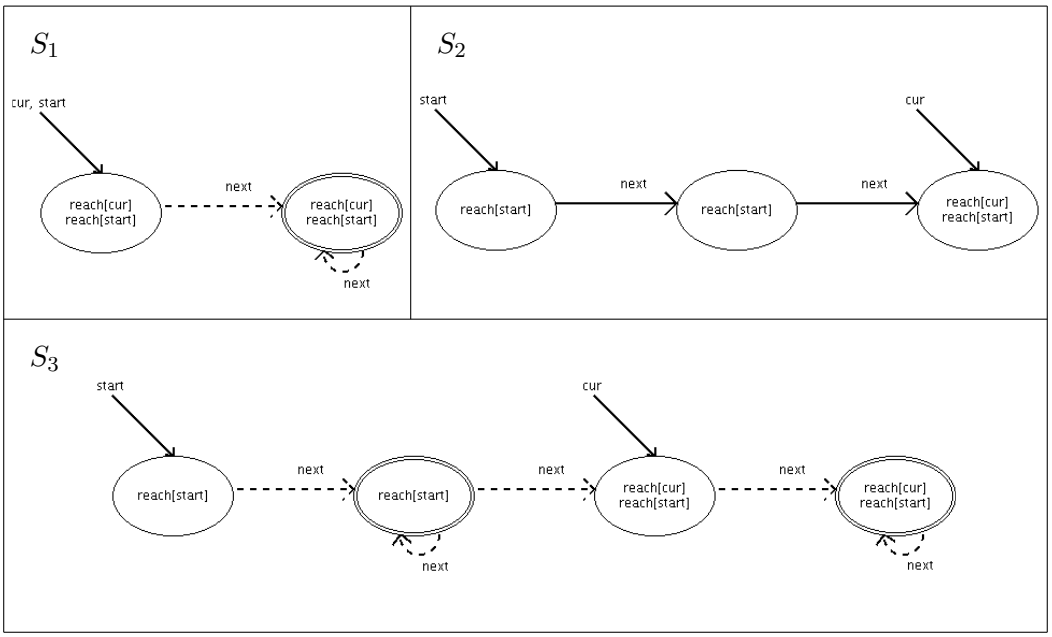

Abbildung 4.2: Drei Shapegraphen, die bei Listen auftreten

In Abbildung 4.2 sehen wir drei der Shapegraphen, die am Programmpunkt n2, das ist der Schleifeneingang, vergleiche hierzu Abbildung 4.1, auftreten. Genaugenommen sind die abgebildeten Shapegraphen unvollständig. Aus technischen Gründen wird ein Vergleich von Datenwerten auf die Weise realisiert, dass die Datenwerte zweier Heapelemente miteinander verglichen werden. Aus diesem Grund existiert zusätzlich stets eine einzelne Heapzelle für das Vergleichselement, auf sie zeigt die Zeigervariable el. Da es uns primär um die eigentliche Datenstruktur geht, lassen wir der Übersichtlichkeit halber solche Komponenten weg:

4.2 Vereinbarung. Bei allen Abbildungen von Shapegraphen wird auf die Darstellung von Komponenten verzichtet, die nicht zur Datenstruktur gehören. Insbesondere werden Vergleichselemente fortgelassen.

In den Darstellungen der Shapegraphen sind Summary-Ecken durch eine doppelte Berandung als solche gekennzeichnet. Der Wahrheitswert einstelliger Prädikate wird in der Ecke notiert: Ist er gleich 1, so steht nur der Name des Prädikats; ist er 0, so fehlt er. (Der Wahrheitswert $\frac{1}{2}$ wird in der Form „Prädikatsname $= \frac{1}{2}$“ dargestellt.) Der Wahrheitswert $p(u_1, u_2)$ eines zweistelligen Prädikats wird durch eine gerichtete Kanten $[u_1, u_2\rangle$ repräsentiert: Durchgezogene Kanten bedeuten 1, gestrichelte $\frac{1}{2}$ und fehlende 0. In S_3 sind alle in der vorangegangenen Auflistung genannten Listenteile als Ecken vorhanden, während in S_1 und S_2 einzelne Listenteile nicht vorkommen. In diesem Sinne beschreibt S_3 einen allgemeineren Fall als S_1 und S_2. In der Darstellung aller Shapegraphen haben wir das Prädikat nextStar der Übersicht halber weggelassen.

Eine Shapeanalyse benötigt als Eingabe eine Analysespezifikation, den Kontrollflussgraphen des zu untersuchenden Algorithmus und Startshapegraphen. Die Spezifikation setzt sich zusammen aus den Prädikaten, die die Abstraktion beschreiben, und Aktionen, also Transformationen von Shapegraphen in andere Shapegraphen, die Anweisungen und Ausdrücken im Programm entsprechen. Letztere haben wir hier nicht behandelt. Der Kontrollflussgraph unseres Beispielalgorithmus ist in Abbildung 4.1 rechts gezeigt. Als Startshapegraphen kommen drei Shapegraphen in Betracht. Als Erstes kann die Liste leer sein, der Shapegraph ist dann die leere Struktur. Als Zweites kann es sich um eine einelementige Liste handeln. Der Shapegraph ist dann eine Struktur mit einem Universum, das aus einem einzigen Individuum besteht. Die Prädikate start und reach[start] sind für dieses Individuum wahr. Als Drittes kann die Liste aus mehr als einem Element bestehen. Dieser Shapegraph sieht fast wie S_1 in Abbildung 4.2 aus, er unterscheidet sich von diesem nur dadurch, dass cur und reach[cur] stets 0 sind, also insbesondere cur nicht auf das Startelement zeigt. Er besteht aus zwei Ecken: eine repräsentiert das Listenelement, auf das start zeigt, während alle weiteren Listenelemente zu einer Summary-Ecke zusammengefasst sind. Dieser Startshapegraph beschreibt eine allgemeinere Liste als die beiden erstgenannten. (Des Weiteren enthält jeder dieser Shapegraphen eine zweite nur aus einer Ecke bestehende Komponente, auf die der Zeiger el zeigt.)

4.1.2 Bäume

In diesem Abschnitt entwickeln wir eine Shapeanalysespezifikation für Algorithmen, die auf binären Bäumen operieren. Dazu erweitern wir die zuvor vorgestellte Listenspezifikation. Wie zuvor konzentrieren wir uns auf die Abstraktion der Datenstruktur und besprechen die TVLA-Aktionen, die den Anweisungen und Ausdrücken im Programmcode entsprechen, nur am Rande. Für sie sei erneut auf die bei TVLA mitgelieferten Beispiele verwiesen, siehe [TVLA 2007]. Als Erstes behandeln wir Prädikate, die (in einem weiten Sinne) die Zeigerstruktur der Baumelemente betreffen. Danach ergänzen wir Prädikate, die Ordnungseigenschaften auf

den Datenwerten beschreiben. Beides zusammen ergibt die Abstraktion für (binäre) Suchbäume.

Neben doppelt verketteten Listen stellen (binäre) Bäume wohl die einfachste Erweiterung von einfach verketteten Listen dar. Des Weiteren sind Bäume eine sehr wichtige und universell einsetzbare Datenstruktur. Daher ist es natürlich und naheliegend, dass man bestrebt ist, Shapeanalyse auf sie anzuwenden. Jedoch sind Arbeiten rar, die sich ausschließlich mit der Spezifikation und Analyse von Baumalgorithmen beschäftigen, [Loginov u. a. 2006] ist eine der wenigen Ausnahme. In anderen Arbeiten wie zum Beispiel [Lee u. a. 2005] werden Bäume neben anderen Datenstrukturen angesprochen. In [Parduhn 2005] wird eine Spezifikation für AVL-Bäume vorgestellt und das Einfügen eines Elementes in einen AVL-Baum untersucht. In anderen Arbeiten, [Reineke 2005, 2006], geht es um abstrakte Datenstrukturen, die durch Bäume realisiert werden. Wieder andere Arbeiten, [Johannes u. a. 2005], behandeln Shapeanalyse von Baumalgorithmen im Hinblick auf Visualisierungszwecke.

Die Struktur der Heapzellen wird durch die Datenstruktur bestimmt. Im Falle von Bäumen sprechen wir von *Baumelementen*. Wir setzen wieder voraus, dass alle Baumelemente dieselbe Struktur haben. Sie enthalte drei Komponenten: einen Datenwert eines festen Typs und zwei Zeiger auf Baumelemente. Die Datenkomponente sprechen wir mit dem Wort(teil) *data* an, die beiden Zeiger nennen wir *left* und *right*. Das Baumelement, auf das der left-Zeiger zeigt, nennen wir das linke Kind, beim right-Zeiger entsprechend das rechte Kind. Bei dem Typ der Datenkomponente können wir wieder an einen Zahlentyp denken; allerdings ist für uns nur wichtig, dass wir eine Ordnungsrelation auf ihm definieren können. Die Baumelemente stellen die kleinste Einheit dar, die wir in der Analyse und (damit auch) in der Visualisierung im Heap unterscheiden.

```
cur := root
while cur ≠ NULL do
   if cur.data = el.data
   then
      return cur
   if el.data < cur.data
   then
      cur := cur.left
   else
      cur := cur.right
od
return NULL
```

n1	Copy_Var_T(cur, root)	n2
n2	Is_Null_Var(cur)	notfound
n2	Is_Not_Null_Var(cur)	n3
n3	Equal_Data_T(el, cur)	found
n3	Not_Equal_Data_T(el, cur)	n4
n4	Less_Data_T(el, cur)	n5
n4	Greater_Data_T(el, cur)	n6
n5	Get_Sel_T(cur, cur, left)	n2
n6	Get_Sel_T(cur, cur, right)	n2

Abbildung 4.3: Algorithmus zur Suche eines Elements in einem binären Suchbaum: Pseudocode (links) und TVLA-Programm (rechts)

Wir entwickeln die Abstraktion auf der Grundlage der schon bei Listen diskutierten

Aspekte. Wir haben wieder eine Spezifikation zum Ziel, die sich zur Unterstützung von Lernvorgängen bei Algorithmen, die auf Bäumen operieren, eignen. Die intentionierte Anwendergruppe habe wieder wenig bis leicht fortgeschrittene Kenntnisse in Algorithmen und Datenstrukturen. Deshalb sollen die entstehenden Shapegraphen wieder der Strukturerhaltungsforderung von Seite 61 genügen: Die graphentheoretische Struktur eines jeden Shapegraphen soll die Zeigerstruktur der Baumelemente auf dem Heap widerspiegeln. Des Weiteren lassen wir uns wieder von einem Beispielalgorithmus inspirieren. In Analogie zum Vorgehen bei Listen betrachten wir die Suche nach einem Element in einem binären (Such-)Baum. Dieser Suchalgorithmus ist in Abbildung 4.3 sowohl als Pseudocode als auch als TVLA-Kontrollflussgraph dargestellt. Ausgehend von der Wurzel, auf die der Zeiger root zeigt, durchlaufen wir mit cur Teile des Baumes solange, bis wir entweder das gesuchte Element gefunden haben (eines, das den gleichen Datenwert wie das Referenzelement el hat) oder wir über ein Blatt des Baumes hinausgelaufen sind.

Um eine bessere Vorstellung zu erhalten, welches Aussehen der Shapegraphen wir wünschen, betrachten wir, wie schon bei Listen, die Struktur des Baumes nach einigen (wenigstens drei) Schleifeniterationen. Bei Listen hatten wir uns von der Unterscheidung von schon besuchten und noch nicht besuchten Listenelementen leiten lassen. Wenn wir beim Baumsuchalgorithmus einige Schleifeniterationen ausführen, dann ist der besuchte Teil des Baumes ein Weg, diese Baumzellen bilden also eine Liste. Der nicht besuchte Teil besteht zum einen aus den Teilbäumen, die entlang des Suchweges herunterhängen und aus dem Teilbaum, dessen Wurzel das aktuelle Element ist. Dieser Teilbaum ohne seine Wurzel entspricht dem noch zu explorierenden Teil, während wir die herabhängenden Teilbäume gar nicht durchsuchen werden, wir sie sozusagen links (und rechts) liegen (oder besser hängen) lassen.

Die Unterscheidung des noch zu explorierenden Baumteils vom Rest ermöglichen wieder Prädikate, die Erreichbarkeitseigenschaften ausdrücken. Bei Listen haben wir sie mittels der reflexiv-transitive Hülle des next-Prädikats definiert. Bei Bäumen gehen wir analog vor. Im Gegensatz zu Listen hat jedes Heapelement aber zwei Zeiger, left und right, mit denen man in der Datenstruktur voranschreiten kann. Aus technischen Gründen ist es wieder hilfreich, die reflexiv-transitive Hülle von $\mathsf{left} \vee \mathsf{right}$ durch ein eigenes Prädikat auszudrücken. Zuerst erklären wir daher ein binäres Prädikat down durch $\mathsf{down}(v_1, v_2) := \mathsf{left}(v_1, v_2) \vee \mathsf{right}(v_1, v_2)$. Damit definieren wir das binäre Prädikat downStar und setzen $\mathsf{downStar}(v_1, v_2) := \mathsf{down}*(v_1, v_2)$. Darauf aufbauend erklären wir generisch für jede Zeigervariable z ein unäres Prädikat reach[z] durch $\mathsf{reach}[\mathsf{z}](v) := \exists v_1\, \mathsf{z}(v_1) \wedge \mathsf{downStar}(v_1, v)$. Sie werden als Abstraktionsprädikate ausgezeichnet.

Wir benötigen noch Prädikate, die eine Unterscheidung des Weges von der Wurzel zum aktuellen Element und den an diesem Weg herabhängenden Teilbäumen ermöglichen. Dazu führen wir Prädikate ein, die diesen Weg charakterisieren. Für jede Zeigervariable z führen wir generisch ein unäres Prädikat anc[z] (von ancestor)

ein, das definiert sei durch

$$\mathsf{anc[z]}(v) := \exists v_r, v_c\ v \neq v_c \wedge \mathsf{root}(v_r) \wedge \mathsf{cur}(v_c) \wedge \mathsf{downStar}(v_r, v) \wedge \mathsf{downStar}(v, v_c).$$

Es ist für diejenigen Baumzellen wahr, die Ahnen von z sind, die also auf dem Weg von der Wurzel zur Ecke von z liegen (aber nicht die Ecke ist, auf die z zeigt). Diese Prädikate sind Abstraktionsprädikate. Sie werden generisch für jede Zeigervariable eingeführt, für das Suchbeispiel hat aber nur anc[cur] eine wirkliche Aussagekraft.

Wir verifizieren, dass diese Prädikate schon das gewünschte Aussehen der Shapegraphen liefern. Dazu wechseln wir unseren Blickwinkel. Wir durchlaufen alle Belegungen der Abstraktionsprädikate mit (definiten) Wahrheitswerten und untersuchen, welche Baumelemente (Baumteile) diesen Belegungen entsprechen. Die für den Suchalgorithmus signifikanten Abstraktionsprädikate sind root, cur, anc[cur] und reach[cur]. Wir notieren die Wahrheitswertbelegungen dieser Prädikate in der folgenden Auflistung als Quadrupel, wobei die Reihenfolge der Wahrheitswerte die Reihenfolge der Prädikate im vorangegangenen Satz entspricht. Belegungen, die in der folgenden Auflistung nicht erwähnt werden, sind nicht möglich.

$(1, -, -, -)$**:** das eindeutige Baumelement, auf das der Zeiger root zeigt; die Wurzel des Baumes; es gibt zwei Fälle:

- $(1, 1, 0, 1)$**:** falls auch der Zeiger cur auf die Wurzel zeigt
- $(1, 0, 1, 0)$**:** falls cur nicht auf die Wurzel zeigt

$(-, 1, 0, 1)$**:** das eindeutige Baumelement, auf das cur zeigt; das aktuelle Element; es gibt zwei Fälle:

- $(1, 1, 0, 1)$**:** cur zeigt auf die Wurzel, siehe oben
- $(0, 1, 0, 1)$**:** cur zeigt auf ein von der Wurzel verschiedenes Baumelement

$(0, 0, 1, 0)$**:** Ahnen des aktuellen Elementes, aber weder die Wurzel noch das aktuelle Element; Baumelemente im Inneren des Suchweges von der Wurzel zum aktuellen Element

$(0, 0, 0, 1)$**:** wenn wir mit u dasjenige Baumelement bezeichnen, auf das der Zeiger cur zeigt, dann handelt es sich um diejenigen Baumelemente, die von u aus erreichbar sind (aber ohne u); der Teilbaum mit Wurzel u ohne die Ecke u

Ein *Suchbaum* ist ein Baum, bei dem eine Präordertraversierung eine sortierte Auflistung der Datenwerte ergibt. Dazu äquivalent gilt für jedes Baumelement: Der Datenwert seines linken Kindes ist kleiner oder gleich dem eigenen, und der eigene Datenwert ist kleiner oder gleich dem seines rechten Kindes. Um solche Charakterisierungen in der Analyse vornehmen zu können, benötigen wir Prädikate, die Ordnungseigenschaften der in den Baumelementen enthaltenen Datenwerte beschreiben. Darum ergänzen wir die Spezifikation als Erstes um ein binäres Prädikat (core predicate) dle, von data less or equal, mit der zusätzlichen Charakterisierung „reflexive

transitive“. Die Eigenschaft, dass der Datenwert eines Baumelements zwischen dem seines linken und rechten Kindes liegt, wird durch ein unäres Instrumentationsprädikat inOrder[dle] ausgedrückt. Es wird letztendlich durch

$$\mathsf{inOrder}[\mathsf{dle}](v) := \big(\forall v_l \colon \mathsf{left}(v, v_l) \Rightarrow \mathsf{dle}(v_l, v)\big) \wedge \big(\forall v_r \colon \mathsf{right}(v, v_r) \Rightarrow \mathsf{dle}(v, v_r)\big)$$

definiert. Die durch inOrder ausgedrückte Eigenschaft eines Baumelementes soll eine ergänzende Eigenschaft sein, eine Baumecke kann sie haben oder nicht. Die Eigenschaft soll aber keine Unterscheidung von Ecken im Shapegraphen herbeiführen. Deshalb definieren wir das Prädikat als Nichtabstraktionsprädikat. Es verwundert sicher nicht, wenn wir erwähnen, dass aus technischen Gründen und zum Zwecke einer „harmonischen“ Spezifikation gewöhnlich noch weitere Prädikate zur Beschreibung von Ordnungseigenschaften hinzugefügt werden – hierauf spielt das obenstehende „letztendlich“ an –; diese brauchen uns für unserer Beispiel aber nicht zu interessieren.

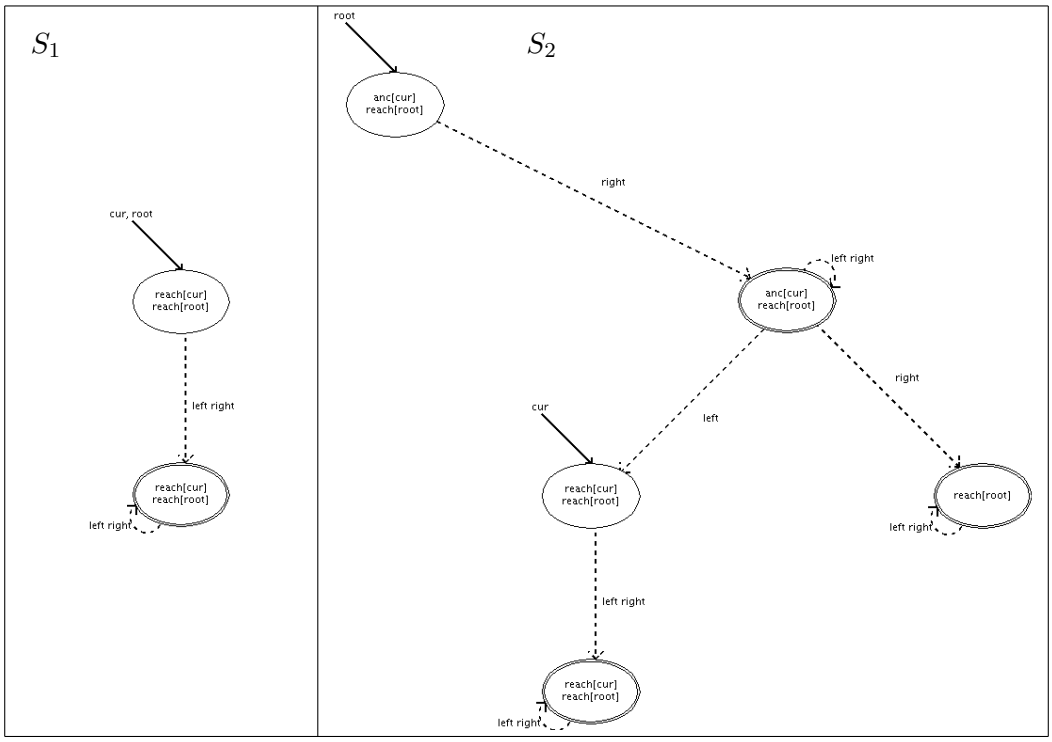

Abbildung 4.4: Zwei Shapegraphen, die bei Bäumen auftreten

In Abbildung 4.4 sehen wir zwei Shapegraphen bezüglich der hier vorgestellten Analysespezifikation, die am Programmpunkt n2, das ist der Eingangspunkt der Suchschleife, auftreten. Auf die Darstellung des Vergleichselements wurde, wie vereinbart, verzichtet. Der linke Shapegraph, S_1, entsteht, wenn der Programmfluss das erste Mal n2 erreicht. Der rechte Shapegraph, S_2, zeigt eine Heapsituation, wie sie

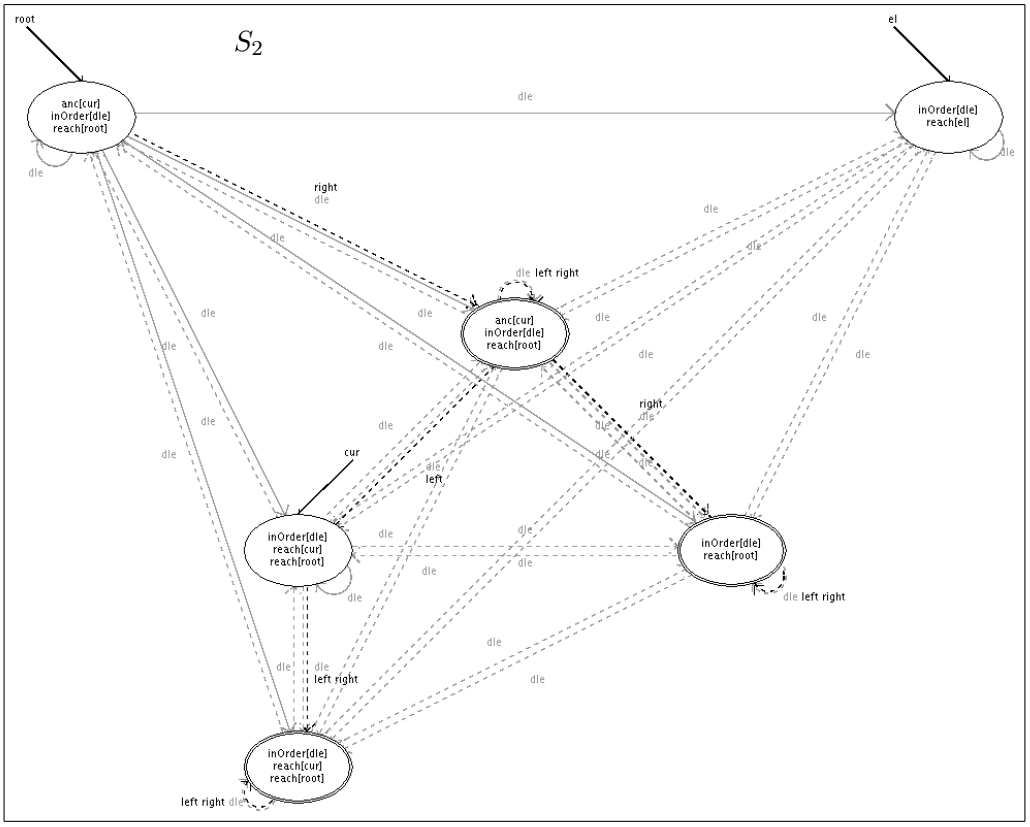

Abbildung 4.5: Shapegraph mit Prädikaten für Ordnungseigenschaften

nach einigen Schleifeniterationen auftritt. Prädikate, die Ordnungseigenschaften beschreiben, sowie Prädikate wie downStar, die vornehmlich aus technischen Gründen eingeführt wurden, haben wir der Übersichtlichkeit halber fortgelassen. In Abbildung 4.5 sehen wir S_2 erneut, diesmal ist auch das Prädikat dle mit dargestellt. Die Menge der Kanten erschwert die Erfassung der Struktur und der Eigenschaften des Graphen deutlich. Es erfordert nun keine große Fantasie sich auszumalen, wie der Shapegraph aussehen würde, wenn wir auch die anderen Prädikate wie down und downStar mit darstellen lassen. Daher verzichten wir in dieser Arbeit üblicherweise auf die Darstellung der „technischen“ Prädikate und der Prädikate, die Ordnungseigenschaften beschreiben.

Als Startshapegraphen kommen, wie schon bei Listen, drei Graphen in Betracht: zum Ersten ein eineckiger Graph, der nur aus der Wurzel des Baumes besteht; zum Zweiten ein Graph, der aus zwei Individuenecken besteht und zum Dritten ein zweieckiger Graph mit einer Summary-Ecke für den von der Wurzel verschiedenen Teil des Baumes. Dieser Graph ähnelt S_1 in Abbildung 4.4, lediglich das Prädikat cur und alle mit cur zusammenhängenden Prädikate sind 0.

In diesem Abschnitt haben wir die Spezifikation von Shapeanalysen von Algorithmen, die auf Listen oder Bäumen operieren, behandelt. Sowohl Listen als auch Bäume sind sehr universelle Datenstrukturen mit vielfältiger Anwendung. Bäume dienen in dieser Arbeit als Beispiel und Untersuchungsgegenstand. Uns ging es hier hauptsächlich um die Abstraktion der Datenstruktur. Sie soll dergestalt beschaffen sein, dass die von der Shapeanalyse berechneten Shapegraphen im Hinblick auf die Visualisierungszwecke aussagekräftig sind. Dazu haben wir eine Strukturerhaltungsforderung formuliert und die Abstraktion nach ihr ausgerichtet. Nachdem die Spezifikation behandelt ist, stellt sich die Frage nach der Beschaffenheit der sich ergebenden Analyseausgaben. Damit fahren wir im folgenden Abschnitt fort.

4.2 Quantitative Aspekte der Analyseausgaben

Im vorangegangenen Abschnitt haben wir uns mit der Spezifikation von Listen und Bäumen beschäftigt. Jetzt schreiten wir in der Reihe vom Algorithmus über die Shapeanalyse zur Visualisierung einen Schritt weiter und wenden uns den Analyseausgaben zu, wobei wir die vorgestellten Spezifikationen zugrunde legen. An dieser Stelle interessiert uns die Größe der erzeugten Analyseausgaben. Diesen quantitativen Aspekt beurteilen wir im Hinblick auf die Visualisierung.

Als Erstes betrachten wir Listen. Wir führen mit der in Teilabschnitt 4.1.1 behandelten Spezifikation und den dort angeführten Startshapegraphen eine Shapeanalyse für einige Listenalgorithmen durch und betrachten die Ausgabe. Wir untersuchen zunächst Algorithmen wie die Suche nach einem Element, das Einfügen eines Elementes, das Löschen eines Elementes oder das Invertieren einer Liste. Wir können sie in einem weiteren Sinn als Basisoperationen für Listen auffassen. Für jeden dieser Algorithmen entstehen für jeden Programmpunkt maximal 15 Shapegraphen. (Diese und die folgenden statistischen Angaben beziehen sich auf die bei TVLA mitgelieferten Listenbeispiele, siehe [TVLA 2007].) Das ist eine überschaubare und übersichtliche Ausgabe und jeder Shapegraph beschreibt einen signifikanten Heapzustand, der auch im Rahmen einer Visualisierung zu berücksichtigen ist. Die Analyseausgabe kann unmittelbar der Visualisierung übergeben werden.

Dieser Sachverhalt ändert sich, was wenig verwundert, wenn wir feinere Spezifikationen verwenden und kompliziertere Algorithmen (auf Listen) betrachten. Wir können der Spezifikation Prädikate hinzufügen, die Ordnungseigenschaften der in den Listenelementen enthaltenen Datenwerte beschreiben, und mit dieser erweiterten Spezifikation Sortieralgorithmen wie Bubble- oder Insertionsort untersuchen, vergleiche [Lev-Ami u. a. 2000] und [TVLA 2007]. In diesem Fall erhalten wir deutlich größere Analyseausgaben. Bei Bubblesort werden an den 31 Programmpunkten insgesamt

4153 Shapegraphen berechnet, das sind im Schnitt fast 134 Shapegraphen pro Programmpunkt, die maximale Anzahl Shapegraphen pro Programmpunkt liegt bei 541. Bei Insertionsort werden an den 29 Programmpunkten insgesamt 6938 Shapegraphen berechnet, das sind im Schnitt 239,2 Shapegraphen pro Programmpunkt, die maximale Anzahl pro Programmpunkt liegt bei 446. (Die Startshapegraphen repräsentieren unsortierte Listen.)

Mit einer gröberen Analyse (tvla.joinType = part) lassen sich diese Anzahlen verringern. Bei Bubblesort werden dann nur noch insgesamt 488 Shapegraphen berechnet, das sind im Schnitt 15,7 Shapegraphen pro Programmpunkt; bei Insertionsort erhalten wir insgesamt 689 Shapegraphen, was im Schnitt 23,8 Shapegraphen pro Programmpunkt sind. Diese gröberen Analysen sind noch stark genug, die partielle Korrektheit der Algorithmen zu zeigen. Jedoch genügen die resultierenden Shapegraphen der Strukturerhaltungsforderung, die wir auf Seite 61 formuliert haben, viel weniger als die der ursprünglichen Analyse. Deshalb erscheint die Verwendung solcher Analysen für den intentionierten Anwendungszweck und die intentionierte Anwendergruppe als nicht sehr vorteilhaft. Es scheint sich das Problem anzudeuten, dass feine(re) Abstraktionen und komplizierte(re) Algorithmen zu einer Analyseausgabe führen, die für eine direkte Visualisierung als zu groß anzusehen ist. Zusammenfassend kann aber festgehalten werden, dass im Großen und Ganzen nicht zu komplizierte Algorithmen auf Listen bei nicht zu feinen Abstraktionen eine kleine bis mäßig große Analyseausgabe erzeugen, die direkt visualisiert werden kann.

Gehen wir nun zu Bäumen über. Wir verwenden die in Teilabschnitt 4.1.2 eingeführte Spezifikation und führen eine Shapeanalyse für den Suchalgorithmus aus Abbildung 4.3 durch und betrachten die berechnete Ausgabe. Je nachdem wie der Gleichheitstest spezifiziert ist, variiert die (Größe der) Analyseausgabe. Bei einem „schwachen“ Gleichheitstest erhalten wir an den 8 Programmpunkten insgesamt 1339 Shapegraphen, das sind im Schnitt 167,4 Shapegraphen pro Programmpunkt, die maximale Anzahl Shapegraphen pro Programmpunkt liegt bei 235. Wenn wir den Gleichheitstest in der Spezifikation „stärker“ ausformulieren, dann erhalten wir insgesamt 1556 Shapegraphen, was einem Durchschnitt von 194,5 entspricht, das Maximum liegt bei 434 Shapegraphen. Zum Vergleich betrachten wir noch den Einfügealgorithmus für AVL-Bäume aus [Parduhn 2005]. Dort werden an 192 Programmpunkten insgesamt 6248 Shapegraphen berechnet, was einen Durchschnitt von 32,5 ergibt, das Maximum der Anzahl Shapegraphen pro Programmpunkt liegt bei 217. Die im Vergleich zum Suchalgorithmus insgesamt doch recht geringe Größe der Analyseausgabe erklärt sich daraus, dass beim Suchalgorithmus jeder Shapegraph, der am Schleifeneingang auftritt, im Prinzip auch die gesamte Suchschleife, und damit fast den gesamten Algorithmus, durchläuft, während der Algorithmus für das Einfügen eines Elementes in einen AVL-Baum eine ganze Reihe Fallunterscheidungen durchführt, wobei die den einzelnen Fällen entsprechenden Programmteile dann nur von einer Teilmenge der Shapegraphen durchlaufen werden.

Diese statistischen Angaben zeigen, dass selbst einfache Baumalgorithmen wie die Suche nach einem Element eine große Analyseausgabe erzeugen. Das gilt also insbesondere schon für Algorithmen, die Basisoperationen zur Verwaltung der Datenstruktur darstellen. Für Algorithmen, die auf ihnen aufbauen beziehungsweise sie verwenden, ist die Situation noch prägnanter. Bei einem erfahrenen Benutzerkreis mag man noch argumentieren können, dass eine Analyseausgabe im Umfang wie wir sie für den Suchalgorithmus erhalten, handhabbar ist. Jedoch sehen wir unseren Ansatz als ein universelles Mittel zur Unterstützung von Lernvorgängen. Insbesondere wollen wir auch die Nutzergruppe mit keinen oder wenig Vorkenntnissen ansprechen können. Für diese ist eine direkte Visualisierung einer Analyseausgabe dieser Größenordnung schwerlich geeignet. Und wenn wir an den Einfügealgorithmus für AVL-Bäume denken, dann erscheint allein wegen der Größe der Ausgabe eine direkte Visualisierung auch für einen erfahrenen Benutzerkreis als wenig geeignet. Wir stehen vor der Herausforderung und Notwendigkeit, die Analyseausgabe vor der eigentlichen Visualisierung in einer zweckdienlichen Form aufzubereiten. Aufbereitung kann dabei vielerlei bedeuten: Beispielsweise kann es bedeuten, dass man ähnliche Shapegraphen zusammenfasst oder dass man „interessantere“ Shapegraphen von „weniger interessanten“ unterscheidet. Dieser Themenkomplex der Aufbereitung der Analyseausgabe im Hinblick auf die Visualisierung ist der Hauptgegenstand dieser Arbeit.

In diesem Abschnitt haben wir quantitative Aspekte von Analyseausgaben besprochen. Eine Shapeanalyse von Listenalgorithmen, insbesondere von Algorithmen, die zur Verwaltung der Datenstruktur verwendet werden, erzeugt in der Regel eine kleine bis mäßig große Ausgabe. Diese Ausgabe kann direkt für eine Visualisierung genutzt werden. Dieser Sachverhalt ändert sich erst bei komplizierteren Algorithmen auf Listen. Bei Bäumen ist die Situation sogleich eine andere. Selbst bei einfachen Algorithmen, also auch bei Algorithmen, die Basisoperationen für Bäume darstellen, erzeugt eine Shapeanalyse eine große Ausgabe. Diese ist für eine direkte Visualisierung eher ungeeignet. Es entsteht das Bedürfnis und die Notwendigkeit, die Analyseausgabe vor der eigentlichen Visualisierung aufzubereiten, sie etwa weiter zu strukturieren oder Shapegraphen, die „interessantere“ Heapsituationen beschreiben, herauszufiltern.

4.3 Richtlinien für Analysespezifikationen: Normalform des Kontrollflussgraphen

Im einleitenden Abschnitt 4.1 dieses Kapitels haben wir uns mit der Spezifikation von Shapeanalysen beschäftigt. Die verwendeten Prädikate beschreiben Eigenschaften, unter deren Blickwinkel der Heapzustand betrachtet wird. Damit bestimmen sie auch dessen Abstraktion. Das wirkt sich letztendlich auch stark auf die Visualisierung aus. Sie erfolgt auf der Basis der von der Shapeanalyse berechneten Shape-

graphen. Damit kann auch nur das visualisiert werden, was in den Shapegraphen ausgedrückt ist. Wir sehen unseren Ansatz zur Visualisierung in einem weiten Sinne als ein lernunterstützendes Mittel. Mit dieser Zielsetzung muss die Spezifikation harmonisieren. Auch die Art und Weise wie das Programm spezifiziert ist, beinhaltet große Relevanz. Wenn beispielsweise eine Sequenz von Aktionen durch Sprünge unmotiviert auseinandergerissen wird, dann dürfte, im Vergleich zu einer Kodierung als Folge, die Lerneffizienz als beeinträchtigt anzusehen sein. Eine Bewertung von Analysespezifikationen im Hinblick auf den Grad, wie stark eine Spezifikation als lernunterstützend zu bewerten ist, wird sich sicherlich nur schwerlich formal und quantitativ fassen lassen. Jedoch ist eine qualitative Beurteilung bis zu einem gewissen Grad möglich. Im Folgenden sollen einige allgemeine Richtlinien für die Spezifikation von Programmen angegeben werden.

Eine Shapeanalyse erfordert verschiedene Eingaben: eine Analysespezifikation, einen Kontrollflussgraphen und Startshapegraphen. Der Kontrollflussgraph spezifiziert den Programmfluss des zu analysierenden Algorithmus. Da wir unseren Ansatz zur Algorithmenvisualisierung in einem Lernumfeld verstehen, können wir keine beliebigen, willkürlichen, „undurchsichtigen“, undidaktischen Kontrollflussgraphen erlauben. Sie sollen stattdessen sowohl die Struktur des eigentlichen Algorithmus klar darstellen als auch Programmen in Programmiersprachen, an ihre Struktur sind die meisten Anwender gewöhnt, nachempfunden sein. Daher zeichnen wir in diesem Abschnitt eine Klasse von wohlgeformten Kontrollflussgraphen aus, wir definieren eine Normalform.

Der *(TVLA-)Kontrollflussgraph* ist ein beschrifteter gerichteter Graph. Seine Ecken heißen *Programmpunkte*. Seine Kanten sind mit TVLA-Aktionen beschriftet, die den Anweisungen und Ausdrücken (beispielsweise in Vergleichen) in Programmiersprachen entsprechen. In Anlehnung an den üblichen Sprachgebrauch im Gebiet Programmiersprachen bezeichnen wir einen Kontrollflussgraphen auch als ein *(TVLA-) Programm*. – Die TVLA-Spezifikation des Kontrollflussgraphen erlaubt die Angabe eines beliebigen Graphen. Damit ist auch die Umsetzung beziehungsweise Wiedergabe einer völlig unstrukturierten, assemblernahen Programmierung möglich. Es können zum Beispiel Sprünge von jedem Programmpunkt zu jedem (anderen) Programmpunkt durchgeführt werden. Darüber hinaus können zwischen zwei Programmpunkten beliebig viele Kanten mit unterschiedlichen Beschriftungen (TVLA-Aktionen) existieren, es können nichtdeterministische Programme angegeben werden. Diese Vielfalt wollen wir einschränken.

Die zu spezifizierenden TVLA-Programme entstammen üblicherweise einer realen, deterministischen, Zeiger unterstützenden Programmiersprache, wie etwa Pascal oder C, oder sie entstammen einer Algorithmenbeschreibung in Pseudocode, die in ihrer Struktur solchen Programmiersprachen nachempfunden ist. Diese (den Ursprung bildenden) Programme sind also in einem durch die Syntax der jeweiligen Sprache beschriebenen Sinne „wohlgeformt“. Wir sehen unseren Ansatz zur Algorithmenvisualisierung als ein lernunterstützendes Mittel. Kontrollflussgraphen, die dem

geltenden Konsens von gutem oder wenigstens normalem Programmfluss widersprechen, dürften einem Lernprozess wenig förderlich sein. Wir werden uns deshalb darauf beschränken, nur solche TVLA-Kontrollflussgraphen zuzulassen, die Strukturen vorhandener und als gut strukturiert anerkannter Programmiersprachen nachempfinden.

TVLA unterstützt (zur Zeit) nur intraprozedurale Analysen. Die Programme, die wir nachempfinden wollen, sind also Prozeduren oder Funktionen. In den gängigen Programmiersprachen bestehen sie aus den folgenden drei Arten von Programmstrukturen:

Anweisungsfolge: eine einzelne Anweisung oder eine Folge von Anweisungen, deren Anfang und Ende gewöhnlich durch Schlüsselwörter (zum Beispiel durch BEGIN–END oder Schweifklammern) markiert sind

Verzweigung: eine Anweisung der Form IF–THEN–ELSE (oder ähnliche Schlüsselwörter), nach IF steht eine Bedingung, nach THEN und ELSE Anweisungsfolgen (allgemeiner: Programme), je nach Wert der Bedingung wird entweder der Teil nach THEN oder nach ELSE ausgeführt, der ELSE-Teil ist optional; manche Programmiersprachen unterstützten Mehrfachverzweigungen (hierunter fällt zum Beispiel die SWITCH-Anweisung in C)

Schleife: verschiedene Varianten: die Anzahl der Iterationen kann beim Schleifeneintritt feststehen oder nicht, Schleifenabburchbedingung kann am Anfang oder Ende stehen; gängige Programmiersprachen verwenden Schlüsselwörter wie FOR, WHILE, DO–UNTIL

Im Folgenden zeichnen wir eine Teilmenge der Menge aller möglichen Kontrollflussgraphen aus. Wir charakterisieren die dazugehörenden Graphen durch strukturelle Eigenschaften. Ein Programm, dass diesen Bedingungen genügt, nennen wir ein (TVLA-)Programm in Normalform. Die Spezifikation ist flexibel genug, um die oben genannten, in üblichen Programmiersprachen vorkommenden Strukturen getreu wiederzugeben. Andererseits ist sie einschränkend genug, um eine klare Struktur zu gewährleisten und uns, falls wir über den Kontrollfluss von Programmen argumentieren, vor allzu viel Sonderfällen und Spezielbetrachtungen zu bewahren.

Blöcke und ihre Konkatenation

Wir werden die Normalform von TVLA-Programmen, also von Kontrollflussgraphen, durch Zusammensetzen, durch Konkatenation von Teilprogrammen (Teilgraphen) bestimmter Form definieren. Da wir im Folgenden stets über (Teilgraphen eines) Kontrollflussgraphen reden, sind, auch wenn wir es einmal nicht explizit betonen, alle Graphen, die im Rest des Abschnitts auftreten, gerichtete Graphen, dessen Kanten mit TVLA-Aktionen beschriftet sind. Es gibt drei Formen von Teilprogrammen: solche, die linearen Sequenzen von Anweisungen entsprechen, solche,

die Verzweigungen entsprechen, und solche, die Schleifen entsprechen. Diese Teilgraphen sollen eine geregelte Verbindung zum Rest des Graphen haben: Alle Kanten aus einem Teilgraphen heraus starten in einer ausgezeichneten Ausgangsecke, alle Kanten in einen hinein münden in einer ausgezeichneten Eingangsecke. Wir formalisieren dies:

4.3 Definition. *Es sei B ein gerichteter Graph dessen Kanten mit TVLA-Aktionen beschriftet sind. Ferner seien e und a zwei nicht notwendig verschiedene Ecken von B. Das Tripel (B, e, a) heiße ein* Block*, e heiße* Eingangsecke *und a heiße* Ausgangsecke *des Blocks.*

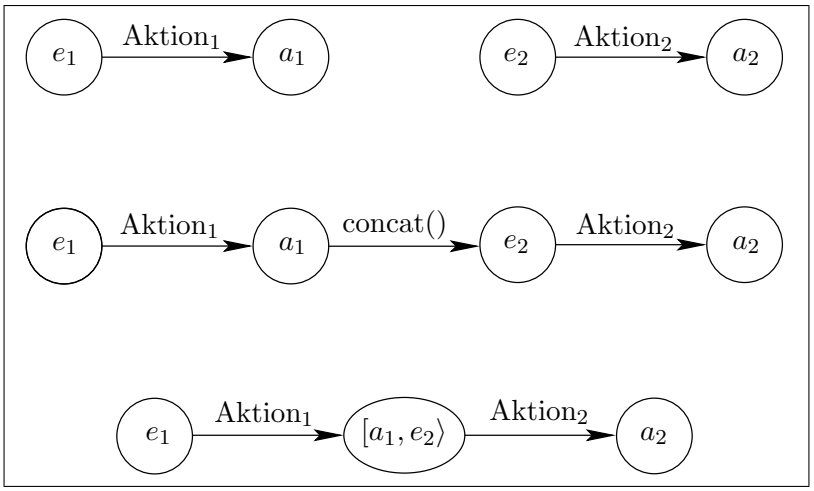

Abbildung 4.6: TVLA-Kontrollflussgraph: Konkatenation von Blöcken

Als Nächstes widmen wir uns dem geregelten Zusammensetzen von Blöcken. Von den beteiligten Blöcken setzen wir stets voraus, dass sie disjunkt sind, dass sie also keine gemeinsamen Ecken haben. Abbildung 4.6 zeigt ein Beispiel. Wir wollen die zwei Blöcke der ersten Zeile zusammenfügen. Der Einfachheit halber bestehen sie jeweils nur aus einer Aktion. Dazu verbinden wir die Ausgangsecke des ersten Blocks mit der Eingangsecke des zweiten und versehen diese Kante mit einer leeren TVLA-Aktion, die wir „concat“ nennen, als Beschriftung. Eine leere Aktion ist eine TVLA-Aktion, die jeden Shapegraphen, auf den sie angewendet wird, unverändert lässt, sie ist die Identität. Gewöhnlich enthält jede TVLA-Spezifikation bereits eine solche Aktion, wir verwenden im Grunde genommen also nur einen anderen Namen für unsere Darstellung. Auf diese Weise können wir anhand der Kantenbeschriftung erkennen, ob eine Kante zu einem Block gehörte oder beim Zusammenfügen entstanden ist. Diese Art der Konkatenation ist in der zweiten Zeile von Abbildung 4.6 dargestellt. Die neue Kante ist unter programmiertechnischen Gesichtspunkten redundant, kein Programmierer würde sie einfügen. Er würde stattdessen die beiden Endecken miteinander identifizieren, so wie es in der dritten Zeile von Abbildung 4.6 dargestellt ist. Diese Art der Konkatenation betrachten wir als zweite Art der Zu-

sammensetzung. Sie entsteht aus der ersten Art, indem die neue Kante kontrahiert wird: Der entstehende Graph verliert die Ecken a_1 und e_2 und erhält eine neue Ecke, die als $[a_1, e_2\rangle$ bezeichnet sei. Kanten, die vorher in a_1 und e_2 begannen oder endeten, tun dies nun in der neuen Ecke $[a_1, e_2\rangle$. Wir setzen:

4.4 Definition. *Das Wort „concat" werde in der Spezifikation einer TVLA-Analyse nicht verwendet. Wir ergänzen im Vorfeld die Analysespezifikation um die Aktion* concat*, welche die Identität auf der Menge aller Shapegraphen sei.*

Die TVLA-Aktion concat bildet als Identität jeden Shapegraphen auf sich ab. Ihre formale Definition besteht aus einem leeren Rumpf „{}", daran angelehnt nennen wir eine solche Aktion deshalb auch eine leere Aktion. – Wir formalisieren das Zusammensetzen von Blöcken:

4.5 Definition. *Es seien (B_1, e_1, a_1) und (B_2, e_2, a_2) zwei disjunkte Blöcke. Der Graph B sei die Vereinigung von B_1 und B_2 vermehrt um die (gerichtete) Kante $k = [a_1, e_2\rangle$ mit Beschriftung „concat()". Wir nennen den Block (B, e_1, a_2) die* kantenverbundene Konkatenation *von B_1 und B_2. Kontrahieren wir die neue Kante k, dann heiße $(B/k, e_1, a_2)$ die* verschmolzene Konkatenation *von B_1 und B_2.*

Wir werden Aktionenfolgen, Verzweigungen und Schleifen als Blöcke definieren und sie gemäß Definition 4.5 zu größeren Einheiten zusammensetzen. Programme in Normalform werden sich aus diesen drei Arten von Blöcken zusammensetzen. Jeder Block hat (genau) eine Eingangs- und eine Ausgangsecke. Verzweigungen und Schleifen werden wir aus Teilkomponenten zusammensetzen. Diese sind verallgemeinerte Blöcke, sie dürfen mehrere Ausgangsecken und in einem Fall auch mehrere Eingangsecken haben. Was die Art ihrer Konkatenation angeht, verhalten sie sich wie Blöcke, sie erfolgt in einer Definition 4.5 entsprechenden Weise; daher können wir auf eine Formalisierung verzichten. Für jeden der drei Blocktypen definieren wir, analog zur Konkatenation, zwei Formen: eine kantenverbundene und eine verschmolzene. Die kantenverbundene Variante erlaubt die Struktur der einzelnen Komponenten klarer zu formulieren, dafür enthält sie eine Reihe Verbindungskanten, die mit „concat()" beschriftet sind. In einem konkreten Programm wird man sie vermeiden. Die verschmolzene Variante ist diejenige, die (normalerweise) einem realen Programm entspricht.

Aktionenblöcke

Die einfachste Form, mehrere Anweisungen miteinander zu verbinden, ist die Anweisungsfolge. In ihr werden die Anweisungen in linearer Folge als Sequenz hintereinander angereiht. Wir setzen:

4.6 Definition. *Es sei W ein gerichtete Weg von e nach a, dessen Kanten mit TVLA-Aktionen beschriftet sind. Dann heiße (W, e, a) ein* Aktionenblock *in Normalform.*

Wir können in dieser Definition zulassen, das e und a gleich sind. Der Weg besteht dann nur aus einer Ecke. Dies entspricht einem leeren, nur aus einem Programmpunkt bestehenden TVLA-Programm. Ist umgekehrt (G, v, v) ein Aktionenblock, dann besteht G nur aus der Ecke v: Als Aktionenblock ist G ein Weg von v nach v, aber Ecken eines Weges sind paarweise verschieden. Im Allgemeinen, also wenn $e \neq a$ gilt, ist ein Aktionenblock ein Graph, so dass es eine Benennung der Ecken $e = v_1, v_2, \ldots, v_n = a$, $n \geq 2$, gibt mit $\delta_{\text{aus}}(v_i) = 1$, $\delta_{\text{ein}}(v_{i+1}) = 1$ und $[v_i, v_{i+1}\rangle \in W$ für alle $1 \leq i \leq n-1$. Hierbei bezeichnet $\delta_{\text{aus}}(v)$ den Ausgangsgrad einer Ecke v, also die Anzahl der gerichteten Kanten mit Anfangsecke v, analog bezeichnet $\delta_{\text{ein}}(v)$ ihren Eingangsgrad. – Trivialerweise gilt:

4.7 Satz. *Die kantenverbundene und die verschmolzene Konkatenation zweier Aktionenblöcke ist wieder ein Aktionenblock.*

Verzweigungen

Neben Aktionenblöcken stellen Verzweigungen einen wichtigen Hauptbestandteil von (imperativen) Programmiersprachen dar. Wenn wir Schlüsselwörter außer Acht lassen, dann beginnen sie mit einen Programmteil, in dem Ausdrücke ausgewertet werden. Je nach ausgewertetem Wert, beziehungsweise Werten, wird in verschiedene Blöcke verzweigt und das Programm dort fortgesetzt. Die verschiedenen Wege durch die Verzweigung werden abschließend wieder zusammengeführt. Ausgenommen sind solche Wege, die zu einer Beendigung der Prozedur führen.

Wir modellieren diese Struktur. Abbildung 4.7 zeigt das Blockdiagramm einer (einfachen) Verzweigung. Sie ist aus mehreren Teilblöcken aufgebaut, und diese werden in einer bestimmten Art und Weise miteinander verbunden. Diese Verbindungskanten erhalten die Beschriftung „concat()“, die wir in der Abbildung der Übersichtlichkeit halber aber fortgelassen haben. Im Einzelnen gelte:

- Der Teilgraph, welcher der Auswertung der Ausdrücke entspricht und die Verzweigung vorbereitet und einleitet, heiße der *Verzweigungsblock*, und wir bezeichnen ihn mit B_V. Er habe die Eingangsecke $e^{(V)}$, die auch die Eingangsecke der Verzweigung wird. Seine Ausgangsecken seien $a_1^{(V)}, a_2^{(V)}, \ldots, a_n^{(V)}$, $n \geq 2$. (Hätte er weniger als zwei Ausgangsecken, dann läge schwerlich eine Verzweigung vor.) Wir nennen ihn auch genauer einen Verzweigungsblock mit n Ausgängen. Der Verzweigungsblock selbst sei ein (binärer) Baum mit Wurzel $e^{(V)}$, seine Blätter seien die Ausgangsecken $a_1^{(V)}, a_2^{(V)}, \ldots, a_n^{(V)}$. Die Kanten von B_V sollen mit Aktionen beschriftet sein, die Auswertungen von Ausdrücken entsprechen und nicht mit solchen, die Anweisungen entsprechen. Das lässt sich kaum konkreter formalisieren, da TVLA keine solche Unterscheidung kennt, alles sind Aktionen. Wir können uns auch nicht damit behelfen, Aktionen, die Ausdrucksauswertungen entsprechen, dadurch charakterisieren zu wollen, dass

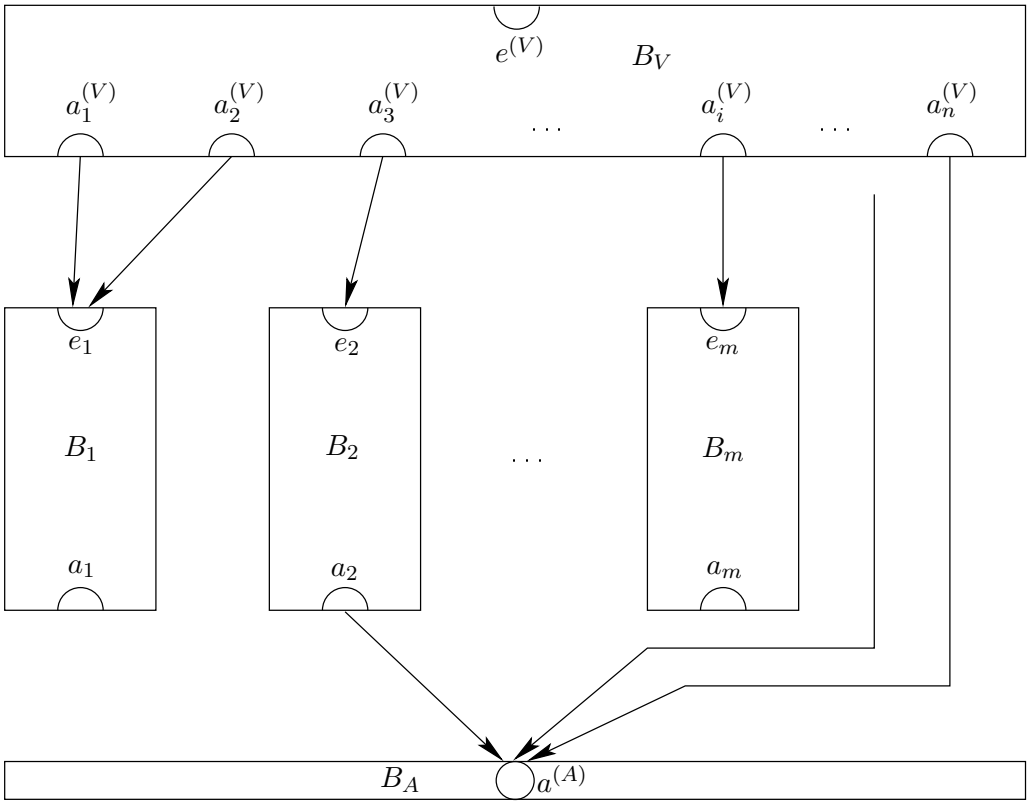

Abbildung 4.7: TVLA-Kontrollflussgraph: Blockdiagramm einer (einfachen) Verzweigung

sie Shapegraphen unverändert lassen. Häufig enthalten diese Aktionen Fokusformeln: eine Anwendung der Aktion auf einen Shapegraphen kann eine Menge (neuer) Shapegraphen produzieren. Andererseits gibt es auch Aktionen, die Anweisungen entsprechen, und (in einem bestimmten Kontext) Shapegraphen unverändert lassen.[4]

- Die Teilgraphen, in die verzweigt wird, seien $B_1, B_2, \ldots, B_m$, $1 \leq m \leq n$. Die Eingangsecke von B_j sei e_j und ihre Ausgangsecke a_j. Jeder dieser Blöcke sei ein Aktionenblock, weshalb wir die resultierende Verzweigung *einfach* nennen.
- Es gebe einen abschließenden Block, der die verschiedenen Wege durch die Verzweigung wieder zusammenführt. Wir nennen ihn den *Ausgangsblock* der Verzweigung und bezeichnen ihn mit B_A. Er bestehe nur aus einer Ecke $a^{(A)}$,

4. Wir haben schon die leere TVLA-Aktion kennen gelernt. Ein weiteres kurzes Beispiel soll den besagten Sachverhalt verdeutlichen. Wir betrachten das zweizeilige TVLA-Programm

n1 Copy_Var_T(*cur*, *root*) n2
n2 Copy_Var_T(*cur*, *root*) n3

Das zweite Aktion hat nach der ersten keine Auswirkung; an dieser Stelle verhält sie sich wie die Identität.

die auch die Ausgangsecke der Verzweigung bilden wird.

- Wir fügen einzelne, neue Kanten hinzu, jede mit der (leeren) TVLA-Aktion „concat()“ beschriftet. In Abbildung 4.7 haben wir diese Kantenbeschriftung der Übersichtlichkeit halber fortgelassen. Die Kanten gliedern sich in drei Arten:
  - Kanten der Art $[a_i^{(V)}, e_j\rangle$, wobei $1 \leq i \leq n$, $1 \leq j \leq m$ ist. Diese Kanten dienen dazu, die einzelnen Ausgänge des Verzweigungsbaumes mit den Blöcken, in die verzweigt werden soll, zu verbinden.
  - Kanten der Art $[a_i^{(V)}, a^{(A)}\rangle$, wobei $1 \leq i \leq n$ ist. Mittels dieser Kanten können wir ein IF-THEN-Konstrukt simulieren. Wertet die einleitende Bedingung zu falsch aus, dann müssen wir direkt zum Ende der Verzweigung springen können, ohne dass irgendein Block abgearbeitet werden muss.
  - Kanten der Art $[a_j, a^{(A)}\rangle$, wobei $1 \leq j \leq m$ ist. Nachdem ein Zweig einer Verzweigung abgearbeitet wurde, kann der Fall eintreten, dass ein Endpunkt des entstehenden Programms erreicht ist. Gewöhnlich soll das Programm aber fortgesetzt werden, daher muss es möglich sein, von den Ausgangsecken a_j der Blöcke die Ausgangsecke der Verzweigung $a^{(A)}$ zu erreichen.

  Insgesamt müssen nach Hinzufügen der neuen mit „concat()“ beschrifteten Kanten die folgenden Bedingungen erfüllt sein:
  - Für jede Ausgangsecke $a_i^{(V)}$ des Verzweigungsblockes sei der Ausgangsgrad $\delta_{\text{aus}}\left(a_i^{(V)}\right) = 1$. Nach der Auswertung der Ausdrücke im Verzweigungsblock wird die Programmausführung deterministisch fortgesetzt, daher kann es höchstens eine ausgehende Kante geben, also $\delta_{\text{aus}}\left(a_i^{(V)}\right) \leq 1$. Dieses Argument erklärt auch die Notwendigkeit für $m \leq n$. Die Möglichkeit $\delta_{\text{aus}}\left(a_i^{(V)}\right) = 0$ erlauben wir nicht. Soll das entstehende Programm an dieser Stelle einen Endprogrammpunkt erreichen, dann muss ein leerer Aktionenblock verwendet werden.
  - Für jede der Eingangsecken e_j der Aktionenblöcke sei der Eingangsgrad $\delta_{\text{ein}}(e_j) \geq 1$. Gälte $\delta_{\text{ein}}(e_j) = 0$ für eine solche Ecke, dann wäre sie von der Eingangsecke der Verzweigung $e^{(V)}$ nicht erreichbar. Der entstehende Kontrollflussgraph würde in Komponenten zerfallen, was nicht erwünscht ist. Die Möglichkeit $\delta_{\text{ein}}(e_j) > 1$ wollen wir nicht ausschließen.
  - Wir erlauben $\delta_{\text{aus}}(a_j) = 0$ für $1 \leq j \leq m$. Eine Verzweigeung darf in einem (inneren) Block beendet werden. Damit simulieren wir RETURN-Anweisungen innerhalb einer Verzweigung.[5]

5. Wenn man sich im Rahmen der Semantik von Programmen für Normalformen von Kontrollflussgraphen interessiert, dann wird man es üblicherweise vermeiden, eine solche Art der Terminierung zu erlauben. Die Beweggründe sind für uns nur bedingt relevant. Unser Absicht besteht primär darin, reale Programme möglichst getreu in einen Kontrollflussgraphen umsetzen zu können.

– Es gelte $\delta_{\text{ein}}\left(a^{(A)}\right) \geq 1$. Die Ausgangsecke der Verzweigung $a^{(A)}$ muss von ihrer Eingangsecke $e^{(V)}$ erreichbar sein. Andernfalls würde der entstehende Kontrollflussgraph wieder in Komponenten zerfallen, was wir nicht wünschen.

Wir fassen die vorstehende Konstruktion zusammen:

4.8 Definition. *Es sei B_V ein Verzweigungsblock mit $n \geq 2$ Ausgängen, ferner seien $B_1, B_2, \ldots, B_m$, $m \leq n$, Aktionenblöcke, und weiter sei B_A ein Ausgangsblock. Setzen wir die Blöcke gemäß den vorstehenden Angaben und Bedingungen zusammen, so heiße der entstehenden Block eine* einfache Verzweigung in kantenverbundener Normalform. *Seine Eingangsecke ist $e^{(V)}$ und seine Ausgangsecke $a^{(A)}$. Kontrahieren wir alle Kanten mit Beschriftung „*concat()*“, dann heiße der resultierende Graph eine* einfache Verzweigung in verschmolzener Normalform.

Schleifen

Der noch verbleibende Grundbaustein imperativer Programmiersprachen ist die Schleife. In Programmiersprachen tritt sie in unterschiedlichen Erscheinungsformen auf. Stets besteht sie aus zwei Komponenten: einem Schleifenrumpf, der die zu wiederholenden Anweisungen enthält, und einem Rahmen zur Verwaltung und Kontrolle des Wiederholvorgangs. Der Verwaltungsteil kann (nahezu) vollständig zu Beginn der Schleife stehen. In den gängigen Programmiersprachen werden solche Schleifen gewöhnlich mit dem Schlüsselwort WHILE eingeleitet. Danach folgt ein Ausdruck, der den Abbruch der Schleife bestimmt. Am Schleifenende erfolgt dann nur ein Rücksprung an den Schleifenanfang. Der Verwaltungsteil kann auch vollständig am Ende der Schleife stehen. Für diese Art von Schleifen verwenden Programmiersprachen oft die Schlüsselwörter DO–UNTIL, die den Schleifenrumpf umschließen. Nach WHILE folgt ein Ausdruck, der den Schleifenabbruch bestimmt. Der Verwaltungsteil kann sich auch auf beide Schleifenenden verteilen. Wenn beispielsweise ein Schleifenzähler verwaltet werden muss, dann erfolgt im vorderen Teil seine Initialisierung und es wird eine Bedingung für den Schleifenzähler überprüft, im hinteren Teil des Schleifenrahmens erfolgt oft eine Wertänderung des Schleifenzählers. In Programmiersprachen werden solche Schleifen üblicherweise durch das Schlüsselwort FOR eingeleitet.

Im Folgenden modellieren wir diese Struktur. Wir werden flexibel genug vorgehen, um die verschiedenen Arten von Schleifen erfassen zu können. Abbildung 4.8 zeigt das Blockdiagramm einer einfachen Schleife. Es ist wieder aus verschiedenen Teilblöcken aufgebaut, die in einer bestimmten Art und Weise miteinander verbunden sind. Diese Kanten erhalten wieder die Beschriftung „concat()“, auf die wir im Diagramm der Übersichtlichkeit halber verzichtet haben. Im Einzelnen gelten die folgende Bedingungen:

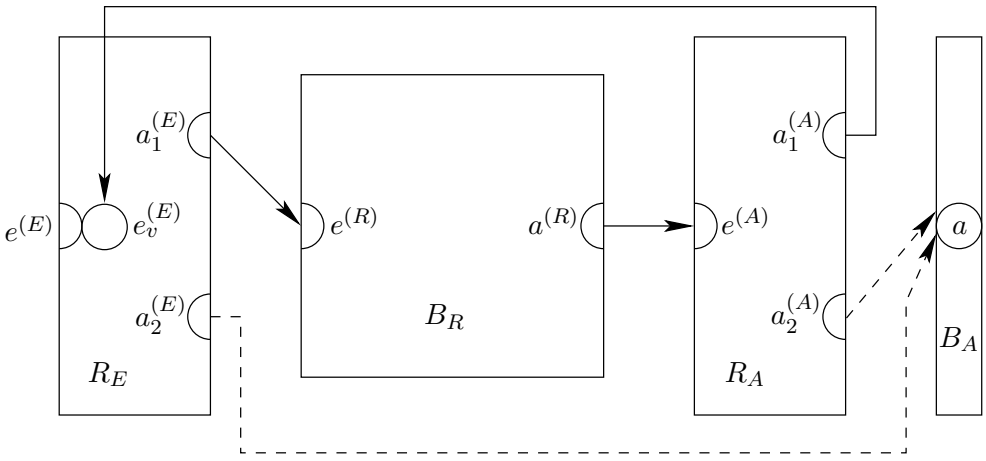

Abbildung 4.8: TVLA-Kontrollflussgraph: Blockdiagramm einer (einfachen) Schleife

- Die Schleife beginne mit einem Teilgraphen, den wir den *Eingangsblock des Schleifenrahmens* nennen und mit R_E bezeichnen. Er kann verschiedene Formen annehmen:
  - Der Eingangsblock ist eine Konkatenation eines Aktionenblockes mit Eingangsecke $e^{(E)}$ und eines Verzweigungsblockes mit Eingangsecke $e_v^{(E)}$ und den zwei Ausgangsecken $a_1^{(E)}$ und $a_2^{(E)}$. Dieser Fall ist in Abbildung 4.8 illustriert. (Der vorangestellte Aktionenblock kann zur Initialisierung eines Schleifenzählers verwendet werden.)
  - Der Eingangsblock ist ein Verzweigungsblock mit der Eingangsecke $e_v^{(E)}$ und den zwei Ausgangsecken $a_1^{(E)}$ und $a_2^{(E)}$. In diesem Fall setzen wir $e^{(E)} = e_v^{(E)}$.
  - Der Eingangsblock ist eine Aktionenfolge mit Eingangsecke $e^{(E)}$ und Ausgangsecke $a_1^{(E)}$. Wir setzen $e_v^{(E)} = a_1^{(E)}$. (Die Ecke $a_2^{(E)}$ existiert dann nicht.)
  - Der Eingangsblock besteht aus einer einzelnen Ecke. Wir setzen $e^{(E)} = e_v^{(E)} = a_1^{(E)}$. (Die Ecke $a_2^{(E)}$ existiert dann nicht.)
- Der Schleifenrumpf sei ein Aktionenblock $\left(B_R, e^{(R)}, a^{(R)}\right)$. Die resultierende Schleife nennen wir daher wieder einfach.
- Den Teilgraph hinter dem Schleifenrumpf nennen wir den *Ausgangsblock des Schleifenrahmens*, wir bezeichnen ihn mit R_A. Er kann dieselben Formen wie der Schleifenrahmeneingangsblock annehmen. Ist er ein Verzweigungsblock oder eine Konkatenation eines Aktionenblocks mit einem Verzweigungsblock, dann sei seine Eingangsecke $e^{(A)}$ und seine zwei Ausgangsecken seien $a_1^{(A)}$ und $a_2^{(A)}$. In den einzelnen Fällen gelten die analogen Eckengleichsetzungen wie oben. (Die Eingangsecke des Verzweigungsblockes brauchen wir nicht explizit

zu benennen.)

- Genau einer der Blöcke R_E und R_A des Schleifenrahmens enthalte einen Verzweigungsblock.[6]
- Danach folge der *Schleifenausgangsblock* (B_A, a, a). Er bestehe aus (genau) einer Ecke a. Er dient zum Zusammenführen der verschiedenen Wege durch die Schleife. Die Ecke a wird die Ausgangsecke der entstehenden Schleife werden.
- Wir fügen einzelne, neue Kanten hinzu. Jede wird mit der (leeren) TVLA-Aktion „concat()“ beschriftet. Im einzelnen sind dies:
  - die Kante $[a_1^{(E)}, e^{(R)}\rangle$. Sie verbindet den Eingangsblock des Schleifenrahmens mit dem Schleifenrumpf.
  - die Kante $[a^{(R)}, e^{(A)}\rangle$. Sie verbindet den Schleifenrumpf mit dem Ausgangsblock des Schleifenrahmens.
  - die Kante $[a_1^{(A)}, e_v^{(E)}\rangle$. Sie führt zum Schleifenrahmeneingangsblock zurück und erzeugt so die Wiederholung. (Ein eventuell vorhandener Aktionenblock im Schleifenrahmeneingangsblock wird übersprungen.)
  - die Kante $[a_2^{(E)}, a\rangle$, falls der Schleifenrahmeneingangsblock mit einem Verzweigungsblock endet. (Nur dann existiert die Ecke $a_2^{(E)}$.) Sie erlaubt durch einen Sprung zum Schleifenabschlussblock ein Beenden der Schleife.
  - die Kante $[a_2^{(A)}, a\rangle$, falls der Schleifenrahmenausgangsblock mit einem Verzweigungsblock endet. (Nur dann existiert die Ecke $a_2^{(A)}$.) Sie ermöglicht ein Beenden der Schleife.

    Da genau einer der Blöcke des Schleifenrahmens einen Verzweigungsblock enthält, existiert von den beiden Kanten $[a_2^{(E)}, a\rangle$ und $[a_2^{(A)}, a\rangle$ genau eine.

Wir fassen die vorstehende Konstruktion zusammen:

4.9 Definition. *Es sei R_E ein Schleifenrahmeneingangsblock und R_A ein Schleifenrahmenausgangsblock, so dass wenigstens einer von ihnen einen Verzweigungsblock enthalte. Des Weiteren sei B_R ein Aktionenblock und B_A ein Schleifenausgangsblock. Werden die Blöcke gemäß den vorstehenden Angaben konkateniert, so heiße der entstehende Block eine* einfache Schleife in kantenverbundener Normalform. *Werden alle Kanten mit Beschriftung „concat()“ kontrahiert, dann heiße der resultierende Graph eine* einfache Schleife in verschmolzener Normalform.

Programme in Normalform

Wir haben für jeden der drei Hauptbestandteile eines Programms, Aktionenfolge, Verzweigung und Schleife, eine Normalform definiert. Bei Verzweigungen und

6. In den gängigen Programmiersprachen sind die Schleifen derart, dass der Test auf das Schleifenende entweder am Anfang oder am Ende der Schleifeniteration erfolgt, aber er ist nicht auf beide Stellen verteilt.

Schleifen haben wir als Basisbestandteile nur Aktionenblöcke zugelassen und sie deshalb einfach genannt. Damit haben wir alle Voraussetzungen beisammen, um für (TVLA-)Programme, (TVLA-)Kontrollflussgraphen, eine Normalform definieren zu können.

4.10 Definition. *Ein* (TVLA-)Kontrollflussgraph – alternativ sprechen wir auch von einem *(TVLA-)Programm* – in kantenverbundener Normalform *entstehe durch endlichmalige Anwendung der folgenden Regeln:*

1. *Jeder Aktionenblock sei ein Programm in kantenverbundener Normalform.*
2. *Bilden wir eine Verzweigung in kantenverbundener Normalform gemäß Definition 4.8, wobei wir anstelle der Aktionenblöcke Programme in kantenverbundener Normalform erlauben, dann entstehe ein Programm in kantenverbundener Normalform.*
3. *Bilden wir eine Schleife in kantenverbundener Normalform gemäß Definition 4.9, wobei wir anstelle des Aktionenblocks für den Schleifenrumpf ein Programm in kantenverbundener Normalform erlauben, dann entstehe ein Programm in kantenverbundener Normalform.*
4. *Es seien P_1 und P_2 zwei Programme in kantenverbundener Normalform.*
   a) *Sind P_1 und P_2 disjunkt, dann sei ihre kankenverbundene Konkatenation gemäß Definition 4.5 ein Programm in kantenverbundener Normalform.*
   b) *Sind P_1 und P_2 nicht disjunkt, dann sei P_2' ein zu P_2 isomorpher Graph so, dass P_1 und P_2' disjunkt sind.*[7] *Die kantenverbundene Konkatenation von P_1 und P_2' gemäß Definition 4.5 sei ein Programm in kantenverbundener Normalform.*

Wenn wir in den obenstehenden Regeln stets die verschmolzenen Varianten verwenden, dann entstehe ein (TVLA-)Kontrollflussgraph in verschmolzener Normalform.

Damit haben wir zwei Normalformen für (TVLA-)Programme ausgezeichnet, bei einer wird immer die kantenverbundene Variante, bei der anderen stets die verschmolzene Variante verwendet. Wenn man einen Algorithmus als Kontrollflussgraph kodiert, dann wird man gewöhnlich die verschmolzene Variante wählen. Daher verstehen wir unter einem *(TVLA-)Programm in Normalform* gewöhnlich ein Programm in verschmolzener Normalform. – Aus dem Konstruktionsprozess der Blöcke ist der Zusammenhang zwischen kantenverbundener und verschmolzener Form schon hervorgetreten. Wir formulieren ihn abschließend noch einmal für Programme, wobei wir auf den formalen Beweis verzichten.

4.11 Satz. *Das TVLA-Programm G_k entstehe gemäß Definition 4.10, wobei stets die kantenverbundene Konkatenation verwendet wird. Das Programm G entstehe aus G_k indem alle mit „*concat()*“ beschrifteten Kanten kontrahiert werden. Das TVLA-*

7. Der Graph P_2' entsteht also aus P_2 durch Umbenennung der Ecken.

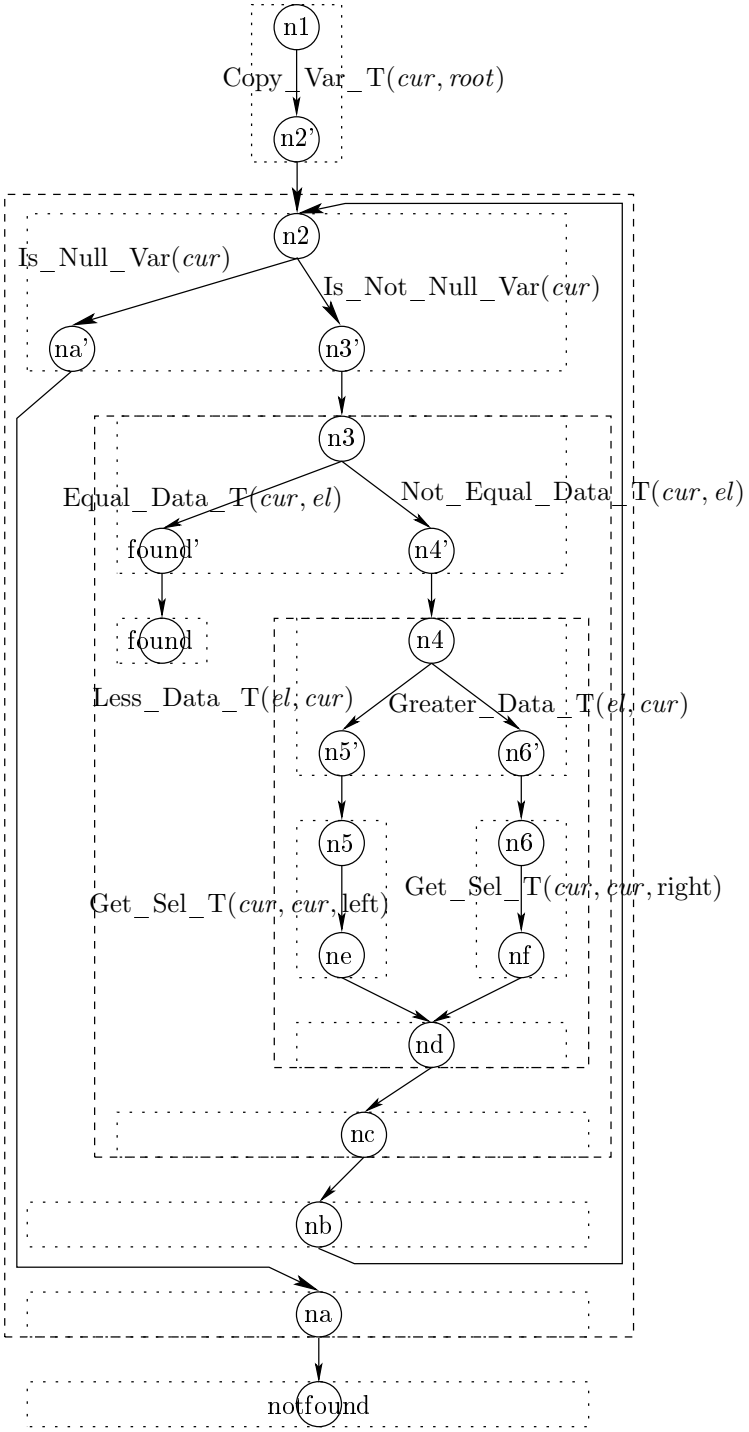

Abbildung 4.9: TVLA-Kontrollflussgraph in kantenverbundener Normalform zur Suche eines Elements in einem binären Suchbaum

Programm G_v entstehe durch denselben Konstruktionsprozess wie G_k, wobei aber stets die verschmolzene Version benutzt wird. Dann sind G und G_v isomorph.

Das Suchbeispiel in Normalform

Die Normalform soll an einem Beispiel illustriert werden. Wir betrachten wieder die Suche nach einem Element in einem binären Suchbaum. In Abbildung 4.3 auf Seite 66 haben wir den Suchalgorithmus als Pseudocode formuliert. Wir wollen daraus einen TVLA-Kontrollflussgraph in Normalform ableiten. In der Abbildung haben wir ebenfalls schon einen Kontrollflussgraphen präsentiert. Ist er vielleicht schon in Normalform? Er enthält keine Aktion concat. Wenn er in Normalform ist, dann muss er auf jeden Fall in verschmolzener Normalform sein.

Der Suchalgorithmus ist relativ einfach aufgebaut. Nach einem nur aus einer Operation (Kopieranweisung) bestehenden Aktionenblock folgt eine Schleife. Der Schleifenrumpf besteht aus zwei aufeinanderfolgenden Verzweigungen. Diese können auf zwei Arten realisiert werden: Es kann ein Verzweigungsblock konstruiert werden, in dem der Verzweigungsblock drei Ausgangsecke hat, oder es kann für die erste Verzweigung ein Verzweigungsblock konstruiert werden, der die zweite enthält. Wir entscheiden uns für die zweite Variante. Die Return-Anweisung hat keine Entsprechung als TVLA-Aktion. Wir simulieren diese Anweisung indem wir das Programm an diesen Stellen terminieren: Die beiden Programmpunkte, wir nennen sie „found“ und „notfound“ erhalten keine Nachfolger. (Alternativ könnte man an diesen Stellen auch einen Aktionenblock einfügen, der aus jeweils einer leeren TVLA-Aktion besteht, die der Return-Anweisung entsprechen soll.)

Wegen der Einfachheit des Algorithmus verzichten wir darauf, die einzelnen Bestandteile und den iterativen Zusammensetzungsprozesses explizit vorzuführen. Der resultierende Kontrollflussgraph in kantenverbundener Normalform ist in Abbildung 4.9 dargestellt. Alle unbeschrifteten Kanten müssen wir uns mit der (leeren) Aktion „concat()“ beschriftet denken, in der Abbildung haben wir der Übersichtlichkeit halber auf sie verzichtet. Die Namen der Ecken haben wir in Anlehnung an den TVLA-Kontrollflussgraphen aus Abbildung 4.3 gewählt. Wenn wir alle mit „concat()“ beschriftete Kanten kontrahieren, dann erhalten wir einen Kontrollflussgraphen in verschmolzene Normalform. Bis auf die Benennung der Ecken stimmt er mit dem Kontrollflussgraphen aus Abbildung 4.3 überein. Dieser ist also tatsächlich in (verschmolzener) Normalform.

5 Logische Hilfsmittel

> Wer gute Arbeit leisten will, schärfe zuerst das Werkzeug.
>
> *(Chinesisches Sprichwort)*

Der in dieser Arbeit beschriebene Ansatz zur Algorithmenvisualisierung basiert auf der Visualisierung von Shapegraphen. Es zeigt sich, dass die Analyseausgabe in der Regel zu groß ist, um sie direkt der Visualisierung zu übergeben. In dieser Arbeit beschäftigen wir uns mit Mechanismen, die Analyseausgabe vor der Visualisierung aufzubereiten. Ganz allgemein heißt dies, dass wir etwas mit den berechneten Shapegraphen machen. Da es sich bei ihnen um logische Strukturen handelt, ist es natürlich, wenn wir uns der Logik als Methode bedienen. In diesem Abschnitt haben wir logische Aspekte gesammelt, die wir später benötigen und auf die wir Bezug nehmen.

Wir haben den Gegenstand der Arbeit als eine Aufbereitung der Shapeanalyseausgabe im Hinblick auf die Visualisierung charakterisiert. Damit stehen wir also gewissermaßen zwischen der Shapeanalyse und der eigentlichen Visualisierung. Unsere Eingabe ist die berechnete Analyseausgabe und prinzipiell auch derjenige Teil der Analysespezifikation, der sich auf die Prädikate bezieht. Beides ist für uns fest. Wir können natürlich immer eine neue Analyse durchführen, aber es findet keine Interaktion zwischen unseren Mechanismen und der Shapeanalyse statt. Daher setzen wir in der ganzen Arbeit stets voraus, dass die Analysespezifikation fest ist.

5.1 Vereinbarung. Die Spezifikation einer Shapeanalyse sei fest und nicht Gegenstand von Veränderungen.

5.1 Shapegraphen mit kanonischem Universum

Ein Shapegraph ist eine logische Struktur bestehend aus einem Universum und einer Belegung aller Prädikate der Analysespezifikation mit Wahrheitswerten. Wir berechnen sie mittels einer Shapeanalyse. Die Elemente des Universums eines Shapegraphen, wir nennen sie auch Ecken oder Individuen, haben in der Regel beliebige Namen. Daher haben die Ecken zweier Shapegraphen syntaktisch oft nichts miteinander gemein. Zum einen wechseln die verwendeten Namen des Öfteren von TVLA-

Version zu TVLA-Version. Zum anderen haben Ecken in verschiedenen Shapegraphen oft verschiedene Name, selbst wenn sie gleiche Teile der Heapstruktur bezeichnen, wenn sie also für die unären (Abstraktions-)Prädikate zu gleichen Wahrheitswerten auswerten. Jedoch können wir Individuen mit ihren kanonischen Namen identifizieren.

Es sei eine Analysespezifikation festgelegt. Die Menge der einstelligen Prädikate sei mit P_1 bezeichnet. Das (einstellige) Prädikat sm ist Bestandteil jeder Spezifikation, es wird von TVLA automatisch hinzugefügt. Damit ist insbesondere P_1 nicht leer. Wir wollen eine Ecke mit den Wahrheitswerten ihrer einstelligen Prädikate identifizieren, dazu setzen wir:

5.2 Definition. *Es bezeichne P_1 die Menge der einstelligen Prädikate einer zuvor festgelegten Analysespezifikation. Ferner sei S ein Shapegraph und u eine seiner Ecken. Das Tripel*

$$\left(\left\{p \in P_1 \colon W_{S,\beta[\frac{v}{u}]}\big(p(v)\big) = 0\right\}, \left\{p \in P_1 \colon W_{S,\beta[\frac{v}{u}]}\big(p(v)\big) = \frac{1}{2}\right\}, \left\{p \in P_1 \colon W_{S,\beta[\frac{v}{u}]}\big(p(v)\big) = 1\right\}\right)$$

heiße der kanonische Name *von u in S.*

Diese Definition weicht leicht von der üblicherweise in der Shapeanalyseliteratur verwendeten ab, vergleiche zum Beispiel [Sagiv u. a. 2002]. – In den kanonischen Namen werden die einstelligen Prädikate gemäß ihrem Wahrheitswert für eine Ecke in Mengen eingeteilt. Da wir eine dreiwertige Logik verwenden, kann jedes Prädikat zu insgesamt drei Wahrheitswerten auswerten, es kann sich also in jeder der drei Mengen befinden. Folglich gibt es zu einer gegebenen Analysespezifikation $3^{|P_1|}$ viele kanonische Namen. Wir fassen alle bezüglich einer gegebenen Analysespezifikation möglichen kanonischen Namen in einer Menge zusammen, sie heiße das *kanonische Universum* bezüglich der zugrunde liegenden Analysespezifikation. In den von uns zum Zwecke der Visualisierung mittels TVLA berechneten Shapegraphen existieren keine Ecken mit gleichen kanonischen Namen, daher ist $3^{|P_1|}$ auch eine obere Schranke für die Anzahl der Ecken dieser Shapegraphen. (Genauer: Ist $A \subseteq P_1$ die Menge der (einstelligen) Abstraktionsprädikate, dann hat jeder Shapegraph höchstens $3^{|A|}$ viele Ecken.)

Mit Hilfe kanonischer Namen identifizieren wir Ecken in Shapegraphen mit den durch die Wahrheitswertbelegung beschriebenen Eigenschaften. Dies erlaubt eine Zuordnung zwischen Ecken und den durch die Eigenschaften beschriebenen Heapteilen. Auf diese Weise können wir Ecken in verschiedenen Shapegraphen miteinander in Beziehung setzen: Haben zwei Ecken gleiche kanonische Namen, dann bezeichnen sie gleiche Heapteile. Wir können auch davon sprechen, dass in einem Shapegraphen

eine Ecke fehlt. Wir meinen damit, dass keine Ecke (und damit auch keine konkreten Heapzelle) mit den durch den kanonischen Namen beschriebenen Eigenschaften existiert.

Wir benennen die Klasse der Shapegraphen, deren Ecken kanonische Namen tragen:

5.3 Definition. *Es sei eine Analysespezifikation gegeben. Wir nennen einen Shapegraphen einen* Shapegraphen mit kanonischem Universum*, wenn sein Universum eine Teilmenge der kanonischen Namen der Analysespezifikation ist.*

Im Hinblick auf die Visualisierung wird bei der Shapeanalyse stets die kanonische Abstraktion verwendet. Damit sind die von TVLA berechneten Shapegraphen dergestalt, dass es zu je zwei seiner Ecken ein (unäres) Prädikat gibt, das für diese Ecken zu verschiedenen Wahrheitswerten auswertet. Je zwei Ecken eines solchen Shapegraphen haben damit verschiedene kanonische Namen. Wenn wir die Individuen durch ihre kanonischen Namen ersetzen, dann entsteht ein zum ursprünglichen Shapegraphen isomorpher Shapegraph mit kanonischem Universum; auf den formalen Nachweis können wir verzichten. Folglich gilt:

5.4 Satz. *Eine Analysespezifikation sei gegeben. Es sei S ein Shapegraph (bezüglich dieser Spezifikation), so dass es für je zwei seiner Ecken ein einstelliges Prädikat gebe, das für diese Ecken zu verschiedenen Wahrheitswerten auswertet. Dann existiert ein zu S eindeutig bestimmter isomorpher Shapegraph mit kanonischem Universum (bezüglich dieser Spezifikation).*

5.2 Eine Beschreibungslogik für Shapegraphen

Gegenstand dieser Arbeit ist die Aufbereitung einer Shapeanalyseausgabe zum Zwecke einer Visualisierung. Dazu werden wir zum Beispiel in Kapitel 6 ein Ähnlichkeitskonzept für Shapegraphen einführen: Zwei Shapegraphen werden als ähnlich bezeichnet, wenn sie in bestimmten Merkmalen oder Eigenschaften übereinstimmen. Ähnliche Shapegraphen können dann als gleich angesehen werden und brauchen nicht gesondert visualisiert zu werden. Bei Shapegraphen handelt es sich um logische Strukturen. Daher ist die Logik eine probate Methode, Shapegraphen auf ihre Eigenschaften hin zu befragen. Für diesen Zweck definieren wir eine Logik, die wir Beschreibungslogik für Shapegraphen nennen. Sie bildet den Gegenstand dieses Abschnitts. Unser Vorgehen bei ihrer Definition ist dasselbe wie bei einer klassischen Logik, weswegen wir uns hier kurz fassen können.

Shapegraphen sind dreiwertige logische Strukturen. Daher benötigen wir eine dreiwertige Logik: In ihr wird es drei Wahrheitswerte geben und sie wird dem Extensionalitätsprinzip genügen. Als wir in Kapitel 3 Shapeanalyse und TVLA behan-

delten, haben wir die für die Shapeanalyse verwendete (dreiwertige) Logik kennengelernt. Ihre Syntax (in mathematischer Notation) und Semantik, beides soweit sie für die formale Entwicklung der Shapeanalyse relevant sind, werden in den in Kapitel 3 genannten Arbeiten beschrieben, beispielsweise in [Sagiv u. a. 2002]. Ihre (vollständige) Syntax in TVLA-Notation ist in [Lev-Ami u. a. 2007] beschrieben.

Die hier definierte Logik weicht in einer Reihe von Punkten ab. TVLA kennt für binäre Prädikate die Operatoren „+“ und „∗“ zur Berechnung der transitiven und der reflexiv-transitiven Hülle. Besonders die reflexiv-transitive Hülle ist häufig nützlich. Es hat sich aber herausgestellt, dass eine Analyse in der Regel verschärft werden kann, indem man eigens Instrumentationsprädikate für die (reflexiv-)transitiven Hüllen einführt. In Abschnitt 4.1 verwenden wir bei Listen das Prädikat nextStar und bei Bäumen das Prädikat downStar. Da wichtige, häufig verwendete reflexiv-transitive Hüllen als Prädikate formuliert sind, haben sich diese Hüllenoperatoren für unsere Zwecke als bisher nicht notwendig erwiesen, weswegen sie auch keinen Eingang in die Beschreibungslogik finden. (Sollten sie sich in der Zukunft doch als unverzichtbar herausstellen, kann die Definition der Beschreibungssprache leicht erweitert werden.) Des Weiteren erlaubt TVLA die Verwendung eines Operators für die Bildung der transitive Hülle einer (allgemeinen) binären Formel, dies benötigen wir ebenfalls nicht. Auf programmiertechnische Konstrukte wie $(\varphi_1 \ ? \ \varphi_2 : \varphi_3)$ für ein IF-THEN-ELSE-Konstrukt verzichten wir ebenso. Dafür führen wir als Erweiterung Vergleiche ein: Wir erlauben, den Wahrheitswert eines Prädikats (für eine bestimmte Variablenbelegung) mit einem Wahrheitswert zu vergleichen.

Syntax der Beschreibungslogik

Eine Logik ist zunächst eine formale Sprache, eine Menge von Wörtern über einem Alphabet aus Symbolen. Daher beginnen wir als Erstes mit der Definition des *Alphabets*. Es bestehe aus:

- Zeichen:
  - 0, $\frac{1}{2}$, 1
  - $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$,), (, $\forall$, $\exists$
  - $=$, $\neq$
  - $\leq$, $\geq$
- einer endlichen, eventuell leeren Menge von Variablen, die wir üblicherweise als v, v_1, v_2 schreiben
- Prädikatssymbolen:
  - eine endliche, eventuell leere Menge P_0 von nullstelligen Prädikatssymbolen

- eine endliche, eventuell leere Menge P_1 von einstelligen Prädikatssymbolen
- eine endliche, eventuell leere Menge P_2 von zweistelligen Prädikatssymbolen

Natürlich seien je zwei der in Liste genannten Zeichen verschieden, insbesondere seien die Prädikatsmengen paarweise disjunkt. Der Großteil der Zeichen stimmt mit den üblicherweise verwendeten überein. Wenn wir die Semantik der Beschreibungslogik erklären, werden wir sehen, dass die Symbole 0, $\frac{1}{2}$ und 1 Zeichen für die Wahrheitswerte sind. Die anderen neuen Zeichen werden sich als Vergleiche herausstellen. Da die für die Shapeanalyse verwendete Prädikatenlogik auf Konstantensymbole (für Individuen) und Funktionssymbole verzichtet, brauchen wir sie ebenfalls nicht einzuführen.

Mit diesen Zeichen setzen wir die Wörter der Logik zusammen, wir nennen sie *Formeln*. Wegen des Verzichts auf Konstanten- und Funktionssymbolen können wir sie direkt definieren und brauchen keinen Umweg über Terme zu beschreiten. Sie seien erklärt durch:

1. 0, $\frac{1}{2}$, 1 sind Formeln.
2. Für ein (nullstelliges) Prädikatssymbol $p \in P_0$ ist p eine Formel.
3. Für ein (einstelliges) Prädikatssymbol $p \in P_1$ und eine Variable v ist $p(v)$ eine Formel.
4. Für ein (zweistelliges) Prädikatssymbol $p \in P_2$ und nicht notwendig verschiedenen Variablen v_1 und v_2 ist $p(v_1, v_2)$ eine Formel.
5. Sind v_1 und v_2 nicht notwendig verschiedene Variablen, dann sind $(v_1 = v_2)$ und $(v_1 \neq v_2)$ Formeln.
6. Ist φ eine Formel, dann ist $\neg\varphi$ eine Formel.
7. Sind φ und ψ Formeln, dann sind $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \Rightarrow \psi)$ und $(\varphi \Leftrightarrow \psi)$ Formeln.
8. Ist φ eine Formel und v eine Variable, dann sind $(\forall v\colon \varphi)$ und $(\exists v\colon \varphi)$ Formeln.
9. Es sei α eine Formel gemäß Punkt 1, sie sei also von der Form 0, $\frac{1}{2}$ oder 1. Des Weiteren sei π eine Formel gemäß den Punkten 2–4, sie sei also von der Form p, $p(v)$ oder $p(v_1, v_2)$. Dann sind $(\pi = \alpha)$, $(\alpha = \pi)$, $(\pi \neq \alpha)$, $(\alpha \neq \pi)$, $(\pi \leq \alpha)$, $(\alpha \leq \pi)$, $(\pi \geq \alpha)$ und $(\alpha \geq \pi)$ Formeln.

Die Menge aller Formeln, die durch endlichmalige Anwendung dieser Regeln entstehen, heiße die *Beschreibungslogik* bezüglich des verwendeten Alphabets. Die unter dem Punkt „Zeichen“ genannten Symbole und im Prinzip auch die Menge der

Variablen[1] sind für jede Beschreibungslogik gleich. Nur die Prädikatssymbole sind variabel. Wir verwenden stets die Prädikatssymbole einer vorgegebenen Shapeanalysespezifikation, wobei wir sie gemäß ihrer Stelligkeit auf die Prädikatsmengen P_1, P_2 und P_3 verteilen. Insbesondere ist damit P_1 nicht leer, da stets $\mathsf{sm} \in P_1$ gilt. Wir nennen die Beschreibungslogik auch eine *Beschreibungslogik für Shapegraphen.*

Wir verwenden das Gleichheitszeichen einmal zum Vergleich von Variablen und zum anderen zum Vergleich von Wahrheitswerten. Im Hinblick auf die Verarbeitung solcher Formeln, beispielsweise bei der Implementierung eines Parsers, wird man es sicherlich bevorzugen, zwei verschiedene Symbole zu verwenden. Wir bevorzugen jedoch eine mathematisch intuitive Notation und verwenden deshalb in beiden Fällen das Gleichheitszeichen, Missverständnisse sind nicht zu erwarten. Wir werden später die Semantik der Implikation und der Äquivalenz durch andere Junktoren erklären, außerdem führen wir die meisten Vergleiche auf Gleichheit und Kleiner-Gleich zurück. In diesem Sinne sind viele Operatoren logisch redundant, ihre Verwendung ist eine Sache der Bequemlichkeit.

Bei der Definition der Syntax einer Logik hat man bei der Festlegung der Klammerung einige Wahlmöglichkeiten, sie unterliegt einer gewissen Willkür. Wir haben uns für die obenstehende Variante entschieden. Diese Entscheidung wird sich (in der Darstellung) nicht übermäßig stark auswirken. Wenn wir Formeln einer Beschreibungslogik angeben, dann werden wir die Klammerung nicht immer syntaktisch exakt einhalten. Wir folgen bei der Angabe von Formeln den gebräuchlichen mathematischen Konventionen und versprechen uns dadurch eine intuitivere Erfassbarkeit der Formeln. So verzichten wir generell bei alleinstehenden Formeln auf eventuell umschließende Klammern, wir schreiben $\varphi_1 \wedge \varphi_2$ anstelle von $(\varphi_1 \wedge \varphi_2)$. Auch vermeiden wir bei Junktoren gleicher Art die Paarbildung, anstelle von $((\varphi_1 \wedge \varphi_2) \wedge \varphi_3)$ schreiben wir $(\varphi_1 \wedge \varphi_2 \wedge \varphi_3)$ oder im Falle einer alleinstehende Formel ganz ohne Klammern.

5.5 Vereinbarung. Logische Formeln werden mit mathematisch gebräuchlicher Formatierung und Klammerung notiert.

Das Gleichheitszeichen darf in Formeln auftreten. Um Unklarheiten zu vermeiden, setzen wir:

5.6 Vereinbarung. Formeln werden mittels des Symbols „:=“ definiert.

1. Verschiedene Beschreibungslogiken können sich in der Anzahl der verwendeten Variablen unterscheiden. Wir hätten in der Definition des Alphabets alternativ eine höchstens abzählbare Menge von Variablen erlauben können, dann ist die Aussage korrekt. Es war jedoch unser Anliegen, die Endlichkeit des Alphabets zu betonen.

Semantik der Beschreibungslogik

Die Semantik beginnt mit der Erklärung, worauf sich eine Formel bezieht, in welchem Kontext sie zu interpretieren ist. Dazu definiert man ein Universum und eine Abbildung, welche die Prädikatszeichen auf (reale) Prädikate gleicher Stelligkeit über dem Universum abbildet. Beides zusammen heißt eine *logische Struktur*. Wir werten Formeln stets in, beziehungsweise für Shapegraphen aus. Diese sind die Strukturen. (Genaugenommen handelt es sich bei den von TVLA berechneten Shapegraphen bloß um Beschreibungen von Shapegraphen bezüglich einer vereinbarten Syntax, wir identifizieren jedoch diese Beschreibung eines Shapegraphen mit dem Shapegraph selbst.) Die besagte Abbildung bildet jedes Prädikatszeichen aus dem Alphabet der Beschreibungslogik auf ein Prädikat im Shapegraphen ab. Nun haben wir im Alphabet die Prädikatssymbole aus der Analysespezifikation übernommen. Sie bildet also ein Prädikatssymbol auf das gleichlautende Prädikat (Prädikatssymbol) im Shapegraphen ab. Wenn p ein Prädikatssymbol der Logik ist, dann bezeichnet p^S das Prädikat im Shapegraphen.

Da wir eine dreiwertige Logik definieren, benötigen wir drei Wahrheitswerte: Diese seien 0, $\frac{1}{2}$ und 1. Wir bezeichnen sie in dieser Reihenfolge als „falsch“, „unbestimmt“ oder „undefiniert“ und „wahr“. Wir fassen sie als Elemente des Intervalls $[0, 1]$ der reellen (beziehungsweise rationalen) Zahlen auf. Damit übertragen sich insbesondere alle Ordnungsrelationen auf die Wahrheitswerte. (Wollten wir die Wahrheitswerte nicht als reellen Zahlen ansehen, dann könnten wir natürlich die benötigte Ordnungsrelationen auf der Menge der Wahrheitswerte auch explizit definieren.)

Bei einer zweiwertigen Logik können wir ein Prädikat als eine Relation auffassen. Gehört ein Argumenttupel zur durch die Relation ausgezeichneten Teilmenge, dann wird der Wahrheitswert für dieses Argumenttupel als wahr angesehen, sonst als falsch. Im dreiwertigen Fall ist diese Sicht nicht mehr möglich. Stattdessen sehen wir Prädikate als (totale) Funktionen mit Wertebereich $\{0, \frac{1}{2}, 1\}$ an. Der Definitionsbereich eines Prädikates mit Stelligkeit n ist U^n, wobei U das Universum der Struktur (des Shapegraphen) bezeichne.

Wir wollen Formeln einen Wahrheitswert zuordnen. Dazu bezeichne β eine Variablenbelegung, also eine Abbildung von der Menge der Variablen in das Universum der Struktur. Des Weiteren sei $\beta[\frac{v}{u}]$ die Spezialisierung von β, sie bilde die Variable v auf das Individuum u ab und stimme ansonsten mit β überein. Für die Auswertung einer Formel φ in einem Shapegraphen mit der Variablenbelegung β definieren wir die *Wahrheitswertauswertefunktion* $W_{S,\beta}(\varphi)$. Sie folgt in ihrer Definition dem syntaktischen Aufbau der Formeln. Es gelte:

$$W_{S,\beta}(0) := 0$$

$$W_{S,\beta}\left(\frac{1}{2}\right) := \frac{1}{2}$$

$$W_{S,\beta}(1) := 1$$

Wir verwenden 0, $\frac{1}{2}$ und 1 einmal als Symbole der Logik und zum anderen als Wahrheitswerte. Es sind aber keine Missverständnisse zu befürchten, da aus dem Zusammenhang klar hervorgehen wird, worum es sich handelt.

$$\begin{aligned}
W_{S,\beta}(p) &:= p^S \\
W_{S,\beta}\big(p(v)\big) &:= p^S\big(\beta(v)\big) \\
W_{S,\beta}\big(p(v_1, v_2)\big) &:= p^S\big(\beta(v_1), \beta(v_2)\big) \\
W_{S,\beta}\big((v_1 = v_2)\big) &:= \begin{cases} 1, & \text{falls } \beta(v_1) = \beta(v_2) \\ 0, & \text{sonst} \end{cases} \\
W_{S,\beta}(\neg\,\varphi) &:= 1 - W_{S,\beta}(\varphi) \\
W_{S,\beta}\big((\varphi \wedge \psi)\big) &:= \min\big\{W_{S,\beta}(\varphi), W_{S,\beta}(\psi)\big\} \\
W_{S,\beta}\big((\varphi \vee \psi)\big) &:= \max\big\{W_{S,\beta}(\varphi), W_{S,\beta}(\psi)\big\} \\
W_{S,\beta}\big((\forall v\colon \varphi)\big) &:= \min_{u \in U}\Big\{W_{S,\beta[\frac{v}{u}]}(\varphi)\Big\} \\
W_{S,\beta}\big((\exists v\colon \varphi)\big) &:= \max_{u \in U}\Big\{W_{S,\beta[\frac{v}{u}]}(\varphi)\Big\} \\
W_{S,\beta}\big((\pi = \alpha)\big) &:= \begin{cases} 1, & \text{falls } W_{S,\beta}(\pi) = W_{S,\beta}(\alpha) \\ 0, & \text{sonst} \end{cases}
\end{aligned}$$

Das Gleichheitszeichen auf der linken Seite ist ein Symbol der Logik, das Gleichheitszeichen auf der rechten Seite ist ein Vergleich auf der Menge der Wahrheitswerte. Entsprechendes gilt für die folgenden beiden Zeilen.

$$\begin{aligned}
W_{S,\beta}\big((\pi \leq \alpha)\big) &:= \begin{cases} 1, & \text{falls } W_{S,\beta}(\pi) \leq W_{S,\beta}(\alpha) \\ 0, & \text{sonst} \end{cases} \\
W_{S,\beta}\big((\pi \geq \alpha)\big) &:= \begin{cases} 1, & \text{falls } W_{S,\beta}(\pi) \geq W_{S,\beta}(\alpha) \\ 0, & \text{sonst} \end{cases}
\end{aligned}$$

Die restlichen Formen von Formeln führen wir auf die oben verwendeten Formen zurück und setzen:

$$\begin{aligned}
W_{S,\beta}\big((v_1 \neq v_2)\big) &:= W_{S,\beta}\big(\neg(v_1 = v_2)\big) \\
W_{S,\beta}\big((\varphi \Rightarrow \psi)\big) &:= W_{S,\beta}\big((\neg\,\varphi \vee \psi)\big) \\
W_{S,\beta}\big((\varphi \Leftrightarrow \psi)\big) &:= W_{S,\beta}\Big(\big((\neg\,\varphi \vee \psi) \wedge (\neg\,\psi \vee \varphi)\big)\Big)
\end{aligned}$$

$$
\begin{aligned}
W_{S,\beta}\big((\alpha = \pi)\big) &:= W_{S,\beta}\big((\pi = \alpha)\big) \\
W_{S,\beta}\big((\pi \neq \alpha)\big) &:= W_{S,\beta}\big(\neg(\pi = \alpha)\big) \\
W_{S,\beta}\big((\alpha \neq \pi)\big) &:= W_{S,\beta}\big((\pi \neq \alpha)\big) \\
W_{S,\beta}\big((\alpha \leq \pi)\big) &:= W_{S,\beta}\big((\pi \geq \alpha)\big) \\
W_{S,\beta}\big((\alpha \geq \pi)\big) &:= W_{S,\beta}\big((\pi \leq \alpha)\big)
\end{aligned}
$$

Wertet eine Formel φ (in einem Shapegraphen bezüglich einer Variablenbelegung) zu 1 aus, dann sagen wir auch abkürzend, dass φ wahr ist. Wertet φ zu 0 aus, dann nennen wir φ falsch. Wertet sie zu $\frac{1}{2}$ aus, dann sagen wir dazu auch, dass sie potentiell wahr ist.

6 Ähnlichkeit von Shapegraphen

> This classification is not arbitrary like the grouping of the stars in constellations.
>
> *(Charles R. Darwin)*

6.1 Vom Nutzen einer Strukturierung

Der Gegenstand unseres Ansatzes zur Algorithmenvisualisierung sind Shapegraphen. Wir berechnen sie mittels einer statischen Programmanalyse. Sie erzeugt für jeden Programmpunkt eine Beschreibung der an diesem Programmpunkt möglichen Heapkonfigurationen, sie erzeugt für jeden Programmpunkt eine Menge von Shapegraphen. In der Regel erhalten wir große Mengen mit vielen Shapegraphen. Bei einer sofortigen, unmittelbaren Visualisierung führt dies schnell zu Unübersichtlichkeit, was unter didaktischen Gesichtspunkten als kritisch zu bewerten ist. Außer in den Fällen, dass die dem Algorithmus zugrunde liegende Datenstruktur sehr einfach ist, wie etwa eine einfach verkettete Liste, oder, dass eine grobe Analyse durchgeführt wurde, ist es normalerweise wünschenswert, die Analyseausgabe vor der eigentlichen Visualisierung in einer geeigneten Weise aufzubereiten.

Wir realisieren dies dadurch, dass wir die Shapegraphenmengen strukturieren, sie mit zusätzlicher Struktur versehen. Wir fassen „ähnliche“ Shapegraphen, also Shapegraphen, die sich durch bestimmte gemeinsame Eigenschaften auszeichnen, in Teilmengen zusammen. Dieses Vorgehen bietet eine zweifache Anwendungsmöglichkeit: Zum einen können wir auf diese Weise Shapegraphen in ausgewählten Teilmengen, also Shapegraphen mit bestimmten gemeinsamen Eigenschaften, gesondert betrachten. In diesem Sinne verhält sich die Methode wie ein Filter. Zum anderen können wir uns von der Detailsicht der einzelnen Shapegraphen lösen und stattdessen die Unterschiede zwischen „verschiedenen“ Shapegraphen, zwischen verschiedenen Teilmengen, zwischen Shapegraphen verschiedener Teilmengen studieren. In diesem Sinne verhält sich die Methode wie ein Abstraktionswerkzeug.

Für einen festen Programmpunkt wollen wir die von der Shapeanalyse zu diesem Programmpunkt berechnete Shapegraphenmenge in disjunkte Teilmengen zerlegen, sie also als disjunkte Vereinigung darstellen, und zwar in einer Weise, dass die Shapegraphen in jeder Teilmenge einander „ähnlich“ sind. Allerdings gehen wir dabei anders herum vor. Wir formalisieren den Begriff der Ähnlichkeit und benutzen ihn,

um ähnliche Shapegraphen in Teilmengen zusammenzufassen. Dies wird uns eine Partition der Shapegraphenmenge erzeugen.

Die Existenz einer absolut gültigen Definition von Ähnlichkeit bei Shapegraphen dürfen wir nicht erwarten. Was Ähnlichkeit bedeuten soll, kann von Visualisierung zu Visualisierung verschieden sein. In der Regel wird es von der Datenstruktur, vielleicht auch vom Algorithmus, vom Vorwissen des Benutzers und von der konkreten Zielsetzung der aktuellen Visualisierung abhängen. Gefragt sind also keine starren Ähnlichkeitsdefinitionen, sondern flexible Konzepte, die sich einfach und praktikabel den Wünschen und Anforderungen des Benutzers anpassen lassen. Wir verstehen unseren Ansatz zur Visualisierung in einem weiten Sinn als ein Mittel zum Lernen, zum Gewinnen von größerem Verständnis über Algorithmen. Mithin benötigen wir Konzepte von Ähnlichkeit, die im Hinblick auf Lernvorgänge sinnvolle und aussagekräftige Partitionen erlauben.

In den nächsten Abschnitten entwickeln wir zwei Ähnlichkeitskonzepte. Beide sind parametrisch, sie werden jeweils durch einen sich auf die zur Analyse verwendeten Prädikate stützenden Spezifikator definiert. Damit erfüllen sie das Kriterium größtmöglicher Flexibilität, die Ähnlichkeitsvorstellung lässt sich in weitem Maße den Wünschen und Anforderungen des Anwenders anpassen. Den beiden Konzepten liegt eine unterschiedliche Sicht der Shapegraphen zugrunde. Bei der ersten Methode sehen wir Shapegraphen als logische Strukturen an und stützen uns bei der Formalisierung der Ähnlichkeit stark auf diese logische Sicht. Bei der zweiten Methode verwenden wir einen stark visuellen Zugang. Wir sehen Shapegraphen als (beschriftete gerichtete) Graphen an und entwickeln eine Ähnlichkeitsvorstellung, die sich auf diese Sicht als Graphen stützt. Beide Methoden stimmen in bestimmten mathematischen Gesetzmäßigkeiten überein. Sie erlauben uns, die Partition auf jeweils dieselbe Art und Weise zu erzeugen, der Prozess der Herstellung der Partition ausgehend vom Ähnlichkeitsbegriff ist in beiden Fällen derselbe.

Bei der Entwicklung und Diskussion der Konzepte berücksichtigen wir sowohl theoretische als auch praktische Aspekte. Unter theoretischem Blickwinkel streben wir eine in sich geschlossene, homogene Diskussion der Begriffe an. Um nur ein Beispiel zu nennen: Uns interessiert stets die Mächtigkeit einer Partitionierungsmethode, also die Frage, ob sich jede, oder zumindest fast jede, Partition einer Shapegraphenmenge mit der vorgestellten Methode erzeugen lässt. Wir streben insgesamt aber keine alles erschlagende Tiefe an. Unter praktischem Blickwinkel untersuchen wir, wie sich die vorgestellten Methoden im Hinblick auf die Visualisierung eignen. Jede Sache hat ihre eigenen Charakteristiken, damit resultieren im Hinblick auf eine Anwendung üblicherweise bestimmte Vor- und Nachteile. Dieser Sachverhalt gilt auch für die vorgeschlagenen Ähnlichkeitsbegriffe. Anhand von ausführlichen Beispielen demonstrieren wir sowohl die jeweiligen Vorzüge und bevorzugten Anwendungsweisen als auch die Problembereiche der jeweiligen Methoden.

Eine Partition der Shapegraphenmenge eines Programmpunktes entspricht einem bestimmten Grad an Abstraktion und einem Fokus auf bestimmte Eigenschaften

oder Zusammenhänge der möglichen Heapsituationen dieses Programmpunktes. Ein Benutzer wird während einer Visualisierungssitzung nicht immer denselben Grad an Abstraktion betrachten und nicht immer denselben Fokus einstellen wollen. Zu Beginn möchte er vielleicht eine grobe Sicht ohne speziellen Fokus, um sich eine Übersicht zu verschaffen, dann möchte er vielleicht zu gradweise feineren Sichten übergehen, vielleicht möchte er eine bestimmte Eigenschaft genauer studieren und wünscht sich eine Sicht, die diese Eigenschaft in größerem Detailreichtum zeigt, vielleicht möchte er danach eine andere Eigenschaft untersuchen und wünscht eine Sicht, welche die neue Eigenschaft detailliert zeigt. Während einer typischen Visualisierungssitzung besteht also häufig der Wunsch, den Fokus und den Grad an Abstraktion zu verändern. Das bedeutet mit anderen Wörtern, dass ein Wechsel der Partition gewünscht wird. Daher ist es angeraten, wenn wir uns mit diesem Thema beschäftigen. Wenn wir zwei Partitionen vorliegen haben, von denen die zweite durch Veränderung aus der ersten hervorgegangen ist, dann untersuchen wir, in welchen Zusammenhang, in welchem Verhältnis, die beiden Partitionen und die sie erzeugenden Ähnlichkeitsbegriffe zueinander stehen. Unser Hauptaugenmerk gilt dabei einer bestimmten Art der Änderung von Partitionen, die wir der intuitiven Vorstellung folgend als Verfeinerung und Vergröberung bezeichnen.

In der einleitenden Motivation haben wir uns vorgestellt, dass die Shapegraphenmengen einer statischen Programmanalyse entstammen und es sich um die Mengen der Shapegraphen handelt, die an den einzelnen Programmpunkten des untersuchten Algorithmus auftreten. Von dieser Sicht können wir uns lösen. Für die folgende theoretische Entwicklung können wir allgemeiner eine beliebige nicht leere und endliche Menge von Shapegraphen voraussetzen. Dass wir die leere und unendliche Mengen ausschließen, ist durch die praktischen Gegebenheiten motiviert. Die leere Menge ist einfach genug, so dass wir sie ohne große Anstrengung visualisieren können, ganz davon zu schweigen, dass wir ihr kaum eine zusätzliche Struktur geben können. Unendliche Mengen lassen sich mit Computern nur schwer bearbeiten, zudem liefert die Shapeanalyse, deren Analyseausgabe die Eingabe der Visualisierung ist, natürlich stets eine endliche Ausgabe. Bei den Shapegraphen der betrachteten Menge soll es sich selbstverständlich um Shapegraphen bezüglich derselben Analysespezifikation handeln. Zur Erinnerung sei erneut auf die Vereinbarung 5.1 zu Beginn des Kapitels 5 verwiesen: Die Analysespezifikation ist fest, allen (in diesem Kapitel) auftretenden Shapegraphen und Ausdrücken liegt dieselbe logische Spezifikation zugrunde.

6.2 Partitionieren von Shapegraphenmengen mittels Formeln

Unser Ansatzes der Algorithmenvisualisierung beruht auf Shapegraphen, die wir mittels einer statischen Programmanalyse berechnen. Diese wird bezüglich einer

vorgegebenen Spezifikation, bezüglich einer vorgegebenen logischen Beschreibung, bezüglich vorgegebener Prädikate durchgeführt. Die resultierenden Shapegraphen sind logische Strukturen bezüglich ebendieser Prädikate. Etwas überspitzt formuliert sind es diese Prädikate, die visualisiert werden. Wir können sie in diesem Sinne als eine zugrunde liegende Basis auffassen. Daher ist es naheliegend, sich bei der Partitionierung von Shapegraphenmengen ebenfalls auf diese Prädikate zu stützen und den Prozess ihrer Herstellung auf sie zu beziehen.

6.2.1 Formelbasierte Ähnlichkeit

Als Ausgangspunkt sei eine endliche, nicht leere Menge von Shapegraphen gegeben. Unser Ziel besteht darin, sie in disjunkte Teilmengen zu zerlegen. Shapegraphen einer Teilmenge sollen einander ähnlich sein. Die Bedeutung von ähnlich spezifizieren wir durch Formeln. Dieses Vorgehen bietet in mehrfacher Hinsicht große Flexibilität. Durch Ändern der Formeln können unterschiedliche Ähnlichkeitsvorstellungen umgesetzt und somit unterschiedliche Partitionen erzeugt werden; wir sind damit in der Lage, die Partitionierung nicht nur der Datenstruktur, sondern auch dem gewünschten Betrachtungswinkel anzupassen. Auch lässt sich die Anzahl der Teilmengen der Zerlegung, die Größe der Partition, auf diese Weise steuern.

Wir gehen bei diesem Ansatz wie folgt vor: Wir spezifizieren eine endliche Menge F von Formeln. Wir sehen zwei Shapegraphen als ähnlich (bezüglich F) an, wenn wir sie bezüglich F nicht unterscheiden können, das heißt, wenn jede Formel aus F für beide Shapegraphen zum gleichen Wahrheitswert auswertet. Alle bezüglich F nicht unterscheidbaren Shapegraphen fassen wir in (Teil-)Mengen zusammen. Sie werden sich als disjunkt herausstellen und bilden damit eine Partition der gegebenen Shapegraphenmenge.

Für die Auswertung von Formeln, also für die Zuordnung von Formeln zu Wahrheitswerten, haben wir in Abschnitt 3.5 die Funktion $W_{S,\beta}$ eingeführt. Dabei bezeichnet $S = (U, \upsilon)$ eine logische Struktur. Sie besteht aus einem Universum U und einer Abbildung υ, die jedem Prädikatssymbol eine Wahrheitswertfunktion gleicher Stelligkeit über U zuordnet. Shapegraphen sind Strukturen in diesem Sinn, allerdings werden der Einfachheit halber anstelle von υ gleich die Wahrheitswertfunktionen angegeben. Des Weiteren bezeichnet β eine Variablenbelegung, also eine Abbildung der Variablensymbole in das Universum U.

Der Wahrheitswert einer Formel hängt normalerweise sowohl von der Struktur als auch von der Variablenbelegung (der freien Variablen) ab. Für unsere Zwecke ist es nicht sinnvoll, wenn die Auswertung einer Formel für einen Shapegraphen von der Variablenbelegung abhängt. Deshalb verlangen wir, dass die verwendeten Formeln keine freien Variablen enthalten, dass sie geschlossen sind. Es besteht dann keine Abhängigkeit mehr von der Variablenbelegung β. Für die Auswertung einer

geschlossenen Formel f bezüglich eines Shapegraphen S schreiben wir einfacher $W_S(f)$.

6.1 Definition. *Ist F eine endliche Menge von geschlossenen Formeln (bezüglich der verwendeten Analysespezifikation), dann nennen wir F einen* Formelpartitionierer.

Der Fall $F = \emptyset$ ist in dieser Definition mit eingeschlossen. – Aufbauend auf dem Begriff des Formelpartitionierers erklären wir, unter welchen Bedingungen zwei Shapegraphen einander ähnlich sein sollen:

6.2 Definition. *Es sei F ein Formelpartitionierer, ferner seien S und T zwei Shapegraphen (bezüglich derselben Analysespezifikation). Dann heißen S und T* ähnlich bezüglich F*, wenn jede Formel aus F für beide Shapegraphen zum gleichen Wahrheitswert auswertet, also wenn $\forall f \in F\colon W_S(f) = W_T(f)$ gilt. Wir schreiben in diesem Fall $S \sim_F T$.*

Ist $F = \emptyset$ die leere Menge, dann ist die in Definition 6.2 geforderte Bedingung trivialerweise erfüllt. Je zwei Shapegraphen sind bezüglich der leeren Menge ähnlich. In diesem Fall besteht die Zerlegung der Shapegraphenmenge nur aus einer einzigen (Teil-)Menge, der gegebenen. Aus praktischer Sicht kann man dies so interpretieren, als würde man dieses Ähnlichkeitskonzept nicht benutzen. Formal hat man es aber ursprünglich mit einer Menge von Shapegraphen und nun mit einer Menge, die als einziges Element die ursprüngliche Menge enthält, zu tun. Wenn man an eine Nichtnutzung dieses Ähnlichkeitskonzeptes denkt, dann erwartet man vermutlich eher eine Klasseneinteilung, bei der jede Klasse aus einem einzigen Shapegraphen besteht. Eine solche Partition lässt sich erzeugen, wir werden weiter unten sehen, wie dies bewerkstelligt werden kann. Die Methode wird sich aber als nicht übermäßig praktikabel erweisen. Zusammenfassend können wir aber festhalten, dass das formelbasierte Ähnlichkeitskonzept die Möglichkeit seiner Nichtnutzung mit einschließt.

Die in Definition 6.2 definierte Ähnlichkeit bezüglich einer Formelmenge ist formal eine binäre Relation (auf der Menge aller Shapegraphen über der zugrunde liegenden Spezifikation). Wir haben sie auf die Gleichheitsrelation in einem Wahrheitswerteraum zurückgeführt. Dort ist die Gleichheitsrelation eine Äquivalenzrelation. Diese Eigenschaft überträgt sich, was man leicht explizit nachrechnet, auf obige Ähnlichkeitsrelation. Damit erhalten wir:

6.3 Satz. *Für jeden Formelpartitionierer F ist die Relation Ähnlichkeit bezüglich F eine Äquivalenzrelation.*

Eine Äquivalenzrelation auf einer Menge induziert eine Partition dieser Menge. (Umgekehrt induziert auch jede Partition einer Menge eine Äquivalenzrelation auf dieser Menge.) Wie zuvor bezeichne F einen Formelpartitionierer, des Weiteren heiße die

(endliche und nicht leere) Menge der Shapegraphen $\mathcal{M}$. Für einen Shapegraphen $S \in \mathcal{M}$ definiert $[S]_{\sim_F} = \{T \in \mathcal{M} \colon S \sim_F T\}$ eine Teilmenge von $\mathcal{M}$, sie heißt die durch S repräsentierte *Klasse*. Wenn der Zusammenhang zur Relation klar ist, schreiben wir anstelle von $[S]_{\sim_F}$ kürzer $[S]$. Wir interessieren uns für die Familie $\{[S] \colon S \in \mathcal{M}\}$ der Klassen. Da stets $S \in [S]$ gilt, ist $\bigcup_{S \in \mathcal{M}} [S] = \mathcal{M}$, auch kann es keine leeren Klassen geben. Aus der Tatsache, dass $\sim_F$ eine Äquivalenzrelation ist, lässt sich leicht die Aussage $S \sim_F T \Leftrightarrow [S] = [T]$ zeigen. Daraus folgt insbesondere $[S] \neq [T] \Rightarrow [S] \cap [T] = \emptyset$. (Ein Element im Schnitt wäre sowohl zu S als auch zu T ähnlich, wegen der Transitivität von $\sim_F$, mit Verwendung der Symmetrieeigenschaft, wären dann auch S und T ähnlich, woraus $[S] = [T]$ folgte.) Zwei Klassen sind also entweder identisch oder sie sind disjunkt. Damit bildet die Familie $\{[S] \colon S \in \mathcal{M}\}$ der Klassen eine disjunkte Zerlegung von $\mathcal{M}$. Wir nennen sie die *durch F induzierte Partition* und schreiben für sie $\mathcal{M}/_{\sim_F}$.

Wir schätzen ab, aus wievielen Klassen eine solche Partition besteht. Wir betrachten dazu einen Formelpartitionierer F, der aus n Formeln bestehe. Unsere zugrunde liegende Logik ist dreiwertig, eine Formel kann zu den drei Wahrheitswerten 0, $\frac{1}{2}$ oder 1 auswerten. Wir denken uns F als geordnete Menge, als n-Tupel. Die Wahrheitswertzuordnungsfunktion W_S kann in natürlicher Weise auf n-Tupel von (geschlossenen) Formeln fortgesetzt werden und induziert so eine Abbildung von der Menge der n-Tupel aus (geschlossenen) Formeln in die Menge $\mathcal{Z} = \left\{0, \frac{1}{2}, 1\right\}^n$. Bezüglich F ähnliche Shapegraphen werden auf das gleiche Bild in $\mathcal{Z}$ abgebildet, nicht ähnliche Shapegraphen auf verschiedene Bilder. Es existiert also eine Injektion von den Klassen der Partition in $\mathcal{Z}$. Die Anzahl der Klassen ist identisch mit der Anzahl Bilder in $\mathcal{Z}$. Die Abbildung braucht nicht surjektiv sein; die Shapegraphenmenge kann so gewählt sein, dass nicht jedes Element von $\mathcal{Z}$ als Bild auftritt. Wegen der Existenz der Injektion kann es nicht mehr Klassen als Elemente in $\mathcal{Z}$ geben. Die ganze Überlegung ist zudem unabhängig von der Reihenfolge der Funktionen im n-Tupel. Damit erhalten wir:

6.4 Satz. *Wird eine (endliche) Menge von Shapegraphen bezüglich eines Formelpartitionierers F partitioniert, dann hat die resultierende Partition maximal $3^{|F|}$ Klassen.*

Unter der *Größe* einer Partition wollen wir die Anzahl ihrer Klassen verstehen. Ist $\mathcal{M}$ eine Menge von Shapegraphen und $\mathcal{P}$ eine Partition von $\mathcal{M}$, dann enthält jede Klasse von $\mathcal{P}$ wenigstens einen Shapegraphen. Damit ist die Größe von $\mathcal{P}$ auch durch $|\mathcal{M}|$ beschränkt. Betrachten wir Partitionen, deren Größe „deutlich kleiner“ als $|\mathcal{M}|$ ist. In diesem Fall wird, wie der vorangegangene Satz zeigt, die Größe der Partition in erster Linie durch die Anzahl der Formeln in F bestimmt. Wir können natürlich immer Formeln formulieren, die für (fast) alle Shapegraphen zum gleichen Wahrheitswert auswerten und somit keine oder nur eine sehr geringe Auswirkung auf die Anzahl der entstehenden Klassen haben. Jedoch kann das Hinzufügen einer weiteren Formel zu einem Formelpartitionierer die Anzahl der Klassen höchstens verdreifachen.

Als nächstes untersuchen wir die Mächtigkeit dieser Partitionierungsmethode. Lässt sich jede denkbare Partition auf diese Weise erzeugen? Gibt es (Arten von) Partitionen, die sich auf diese Weise nicht erzeugen lassen? Die Antwort hierauf gibt:

6.5 Satz. *Es sei $\mathcal{M}$ eine endliche und nicht leere Menge von Shapegraphen und es sei $\mathcal{P}$ eine Partition von $\mathcal{M}$, so dass isomorphe Shapegraphen zu derselben Klasse gehören. Dann gibt es einen Formelpartitionierer F, so dass $\mathcal{M}/_{\sim_F} = \mathcal{P}$ gilt, die von F induzierte Partition also gleich $\mathcal{P}$ ist.*

Beweis. Die Grundidee besteht darin, Shapegraphen, also logische Strukturen, durch Formeln bis auf Isomorphie eindeutig zu beschreiben. Aus diesen Formeln setzen wir dann den Formelpartitionierer zusammen. Unsere Konstruktion ist in den Details deutlich technischer Natur. Daher erlauben wir uns, an einigen Stellen auf den Formalismus zu verzichten und es dabei zu belassen, unser Vorgehen zu beschreiben.

Wie üblich sei die Analysespezifikation fest und alle im Folgenden auftretenden Objekte seinen Objekte bezüglich dieser Spezifikation. Wie in der Bedingung des Satzes sei $\mathcal{M}$ eine endliche und nicht leere Menge von Shapegraphen. (Die Aussage gilt selbstverständlich auch für $\mathcal{M} = \emptyset$, aber dieser Fall ist trivial.) Wir verwenden den folgenden

Hilfssatz. *Es sei S ein Shapegraph. Dann existiert eine geschlossene Formel f_S mit folgenden Eigenschaften: Es gilt $W_S(f_S) = 1$ und für jeden Shapegraphen T mit $W_T(f_S) = 1$ ist S isomorph zu T.*

Es sei $\mathcal{P}_\mathcal{M}$ eine Partition der Menge $\mathcal{M}$ so, dass isomorphe Shapegraphen zu derselben Klasse gehören, etwa

$$\mathcal{P}_\mathcal{M} = \left\{ \{S_1^1, S_2^1, \ldots, S_{n_1}^1\}, \{S_1^2, S_2^2, \ldots, S_{n_2}^2\}, \ldots, \{S_1^k, S_2^k, \ldots, S_{n_k}^k\} \right\}.$$

Mittels des Hilfssatzes konstruieren wir einen Formelpartitionierer aus k Formeln. Dabei drücken wir die Enthaltenseinrelation von Shapegraphen in Klassen durch Formeln aus. Jede dieser k Formeln entspricht einer Klasse. Sie sind jeweils Disjunktionen von Formeln f_S, wobei die Shapegraphen S die Shapegraphen der entsprechenden Klasse sind. Für jede Klasse i, $1 \leq i \leq k$, bilden wir die (geschlossene) Formel

$$f_i = \bigvee_{1 \leq j \leq n_i} f_{S_j^i}.$$

Die Formel f_i wertet für alle Shapegraphen der Klasse i zu wahr aus. Ebenso wertet sie für alle Shapegraphen zu wahr aus, die zu diesen isomorph sind; sie befinden sich nach Voraussetztung aber in der Klasse. Für alle anderen Shapegraphen, also für alle Shapegraphen, die nicht in Klasse i enthalten sind, wertet die Formel zu einem von wahr verschiedenen Wahrheitswert aus. Damit wertet die Formel f_i genau für die Shapegraphen der Klasse i zu wahr aus.

Anschließend setzen wir $F = \{f_1, f_2, \ldots, f_k\}$. Es ist nun leicht einsichtig, dass die von F induzierte Partition gleich $\mathcal{P}_{\mathcal{M}}$ ist.

Wir müssen noch den Hilfssatz beweisen. Dazu konstruieren wir zu einem Shapegraphen S eine Formel f_S, die (genau) für S und für zu S isomorphe Shapegraphen zu wahr auswertet. Die Idee ist, die Wahrheitswerte aller Prädikate für alle Argumente vorzuschreiben. Ein Shapegraph ist eine logische Struktur. Er besteht erstens aus einem Universum, das sei in unserem Falle für den Shapegraphen S die Menge $U = \{u_1, u_2, \ldots, u_n\}$. Zum Zweiten besteht er aus der Angabe der Wahrheitswerte aller Prädikate. Man beachte, dass in der TVLA-Notation üblicherweise nur die von 0 verschiedenen Wahrheitswerte notiert werden, wir spezifizieren alle.

Die Formel f_S ist zweiteilig und hat die Gestalt

$$f_S = \exists v_1, v_2, \ldots, v_n \colon \left(\bigwedge_{\substack{1 \le i,j \le n \\ i \neq j}} v_i \neq v_j \right) \wedge \left(\forall v \colon \bigvee_{1 \le i \le n} v = v_i \right) \wedge f'_S.$$

Als Erstes wird die Existenz von n Variablen (Ecken) gefordert. Die erste Klammer fordert, dass diesen Variablen paarweise verschiedene Ecken entsprechen, es sich also um n verschiedene Ecken handelt. (Mit der Formulierung „fordern“ meinen wir: Wertet die Formel zu wahr aus, dann impliziert der bisher beschriebene Teil der Formel, dass die genannte Aussage gilt.) Die zweite Klammer fordert, dass jede Ecke des Shapegraphen eine der bisherigen n ist. Damit wird sichergestellt, dass der Shapegraph aus genau n Ecken besteht.

Den zweiten Teil bildet die Formel f'_S. Sie wird so aufgebaut, dass sie für die Variablenbelegung $\beta\left[\frac{v_1 v_2 \ldots v_n}{u_1 u_2 \ldots u_n}\right]$ die Wahrheitswerte für alle Prädikate und alle Argumente fixiert. Wir verdeutlichen dies an einem Beispiel. Die Analysespezifikation enthalte das unäres Prädikat p. Die Wahrheitswerte von p seien: $p(u_1) = a_1$, $p(u_2) = a_2$, …, $p(u_n) = a_n$, wobei die a_i konkrete Wahrheitswerte meinen. Wir bilden zu diesem Prädikat die Teilformel $\bigwedge_{1 \le i \le n} p(v_i) = a_i$. Solche Teilformeln bilden wir für jedes Prädikat der Analysespezifikation. Die Formel f'_S entstehe aus der Konjunktion aller dieser Teilformeln.

Aus der Konstruktion der Formel f_S ist einsichtig, dass sie bezüglich der Struktur S zu wahr auswertet, dass also $W_S(f_S) = 1$ gilt. Die Variablenbelegung $\beta\left[\frac{v_1 v_2 \ldots v_n}{u_1 u_2 \ldots u_n}\right]$ ist dabei die einzig mögliche. Eine Ecke in einem Shapegraphen ist durch die Angabe der Werte der Abstraktionsprädikate eindeutig bestimmt. In jedem Shapegraphen gibt es zu jeder Belegung der Abstraktionsprädikate höchstens eine Ecke, die diese Belegung erfüllt. Und da wir die Belegung aus der Spezifikation von S übernommen haben, gibt es in S auch stets eine Ecke mit der vorgegebenen Belegung.

Es sei T ein Shapegraph für den die Formel f_S zu wahr auswertet. Wegen der Konstruktion von f_S hat T dann ebenfalls n Ecken. Des Weiteren gibt es genau eine Variablenbelegung β_T, für die f_S zu wahr auswertet. Die (eindeutige) Variablenbelegung, die die Formel in S erfüllt, sei β_S. Dann ist $\beta_T \circ \beta_S^{-1}$ eine Bijektion vom

Universum von S auf die Ecken von T. Man rechnet leicht explizit nach, dass sie die Prädikate invariant lässt. Die Shapegraphen S und T sind damit isomorph. □

Bei der Formulierung des vorangegangenen Satzes mussten wir auf Isomorphie Rücksicht nehmen. Es lässt sich also nicht jede Partition mittels dieser Methode erzeugen, es bestehen (leichte) Beschränkungen. Das ist von der Methode her auch insofern verständlich, als Formeln für isomorphe Shapegraphen zu gleichen Wahrheitswerten auswerten und diese Shapegraphen somit zwangsläufig derselben Klasse angehören werden. Für uns ist diese Einschränkung nicht relevant und wir können sie unberücksichtigt lassen. Zum einen beschreiben isomorphe Shapegraphen dieselbe Heapstruktur und wir werden sie deshalb schwerlich verschiedenen Klassen zugewiesen sehen wollen. Zum anderen produziert TVLA für jeden Programmpunkt sowieso eine Liste paarweise nicht isomorpher Shapegraphen. – Im Hinblick auf Visualisierung können wir mit der Formelpartitionierungsmethode jede wünschenswerte und gewünschte Partition erzeugen, sie erweist sich damit als eine mächtige und universelle Technik.

Im Anschluss an Definition 6.2 hatten wir folgende Frage aufgeworfen: Ist es mit dieser Partitionierungsmethode möglich, jede gegebene Menge von Shapegraphen so zu partitionieren, dass jede Klasse genau einen Shapegraphen enthält. Die vorangegangenen Erörterungen haben gezeigt, dass es für jede von TVLA erzeugte Shapegraphenmenge Formelpartitionierer gibt, die solche Partitionen induzieren. Die Frage kann also bejaht werden. Der obige Beweis liefert auch eine Methode, einen solchen Formelpartitionierer zu konstruieren. Der Konstruktionsprozess kann automatisiert werden und erfordert nicht zwangsläufig menschliche Interaktion. Der so entstehende Formelpartitionierer ist im Verhältnis zur Klassenanzahl sehr groß. Um eine Partition mit k Klassen zu erzeugen sind nach Satz 6.4 wenigstens $\lceil \log_3 k \rceil$ viele Formeln nötig. Obenstehende Konstruktion erzeugt einen Formelpartitionierer mit k Formeln.

6.2.2 Anwendungsbeispiele

Wie haben die formelbasierte Partitionierungsmethode theoretisch entwickelt. Jetzt sollen einige Beispiele zeigen, wie sie angewendet werden kann. Ihre Auswahl erfolgt dabei im Hinblick auf die Visualisierung. Anhand der Beispiele werden wir sowohl bevorzugte Anwendungsfälle als auch Beschränkungen dieses Ansatzes kennenlernen.

Als Rahmen für unsere Beispiele betrachten wir wieder die Suche nach einem Element in einem binären Suchbaum. Wir befinden uns an einem Programmpunkt in der Suchschleife, beispielsweise am Schleifenanfang. Bei der Visualisierung können wir an der Struktur des Weges von der Wurzel des Baumes zur aktuellen Ecke interessiert sein. Dieser Weg repräsentiert die bisherige Ausführung des Programms, gewissermaßen die Geschichte des aktuellen Programmzustandes. In diesem Fall

können wir an Partitionen interessiert sein, welche die verschiedenen Strukturen dieser Wege oder Eigenschaften dieser Wege widerspiegeln.

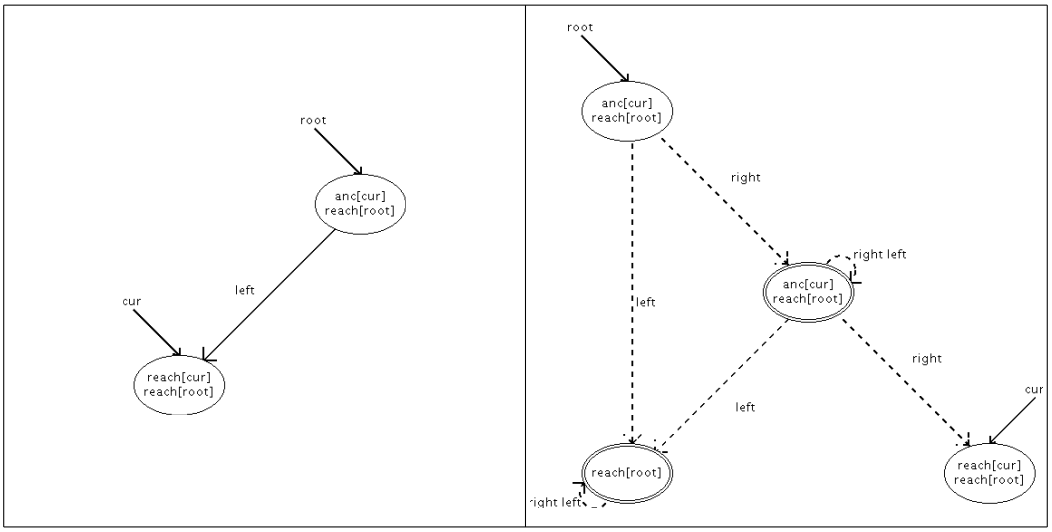

Abbildung 6.1: Zwei bezüglich $\exists v\colon \mathsf{cur}(v) \wedge \forall v_1\colon \neg\mathsf{down}(v, v_1)$ ähnliche Shapegraphen

Wir können daran interessiert sein, unser Augenmerk auf Situationen zu lenken, die zu einem Verlassen der Suchschleife führen. Sie wird in zwei Situationen beendet: Das gesuchte Element wurde (gerade) gefunden oder wir sind in einem Blatt des Baumes angekommen. Der erste Fall kann immer eintreten, er ist unabhängig von der (Zeiger-)Struktur des Baumes (Shapegraphen). Wir beschränken uns daher auf strukturbedingte Abbruchfälle. Wir können also an Partitionen interessiert sein, die Schleifenabbruchfälle von Nichtabbruchfällen unterscheiden. Wir bilden dazu einen Formelpartitionierer F mit der einzigen Formel

$$\exists v\colon \mathsf{cur}(v) \wedge \forall v_1\colon \neg\mathsf{down}(v, v_1).$$

Diese Formel wertet für die Shapegraphen zu wahr aus, in denen es eine Ecke v gibt, auf die der Zeiger cur zeigt und keine – insbesondere keine andere – Ecke ein Kind von v ist.[1] Dies sind die strukturbedingten Schleifenabbruchfälle, und sie werden in einer Klasse gesammelt. In Abbildung 6.1 sehen wir zwei bezüglich F ähnliche Shapegraphen. Bei der verwendeten Analysespezifikation werten alle erzeugten Shapegraphen für diese Formel zu einem definiten Wahrheitswert aus. Die entstehende Partition besteht also aus zwei Klassen.

Die normalerweise verwendeten Analysespezifikationen weisen ein deutliches Maß an Redundanz auf. Obige Eigenschaft lässt sich auch auf andere Weise formulieren:

1. Bisher haben wir Variablen syntaktisch immer von Heapzellen unterschieden: Für Erstere haben wir den Buchstaben v verwendet, für Letztere u. Der Einfachheit halber identifizieren wir in den Beispielen oft beides um nicht jedesmal anmerken zu müssen, dass wir der Diskussion eine Variablenbelegung $\beta[\frac{u}{u}]$ zugrunde legen.

Wir können beispielsweise die Teilformel $v_1 \neq v$ (per Konjunktion) hinzufügen oder das Prädikat down durch left und right oder alternativ durch reach[cur] ausdrücken. Dieses Beispiel zeigt, dass ein Formelpartitionierer durch die Angabe einer Partition nicht eindeutig bestimmt ist.

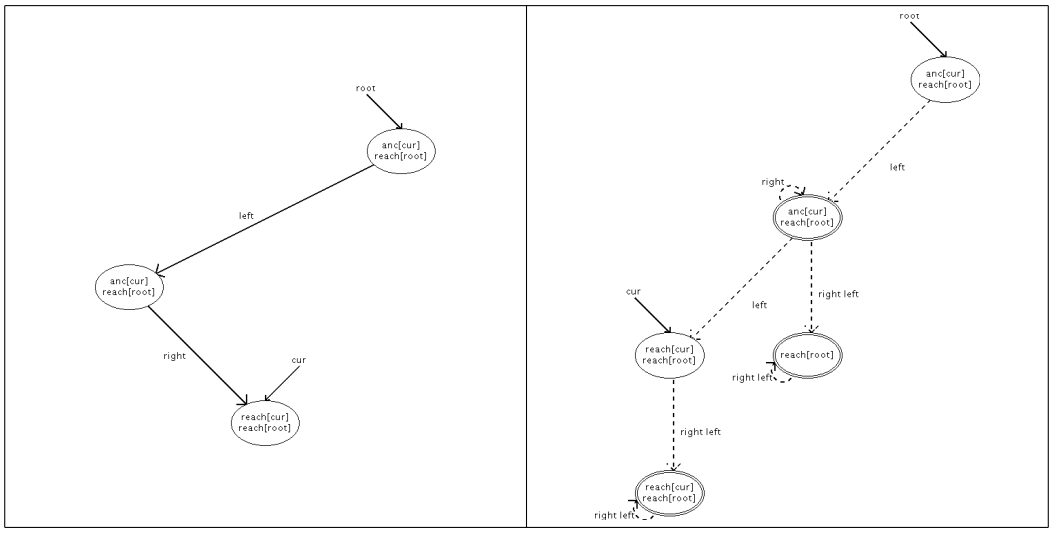

Abbildung 6.2: Zwei Shapegraphen, bei denen der Suchweg aus (wenigstens) drei Ecken besteht

Wir können an Wegen interessiert sein, deren Struktur einen allgemeinen Charakter hat. Das könnte zum Beispiel bedeuten, dass der Weg außer der Wurzel und der aktuellen Ecke noch weitere Ecken enthalten soll. Das lässt sich durch einen Formelpartitionierer bewerkstelligen, welcher die folgende Formel enthält:

$$\exists v_1, v_2, v_3 \colon v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge v_2 \neq v_3 \wedge \\ \mathsf{root}(v_1) \wedge \mathsf{anc}[\mathsf{cur}](v_1) \wedge \mathsf{anc}[\mathsf{cur}](v_2) \wedge \mathsf{cur}(v_3).$$

Sie schreibt die Existenz von drei Ecken auf dem Suchweg mittels der sie charakterisierenden Prädikate vor. Die im vorangehenden Beispiel genannten Bemerkungen hinsichtlich der Nichteindeutigkeit des Formelpartitionierers und der Anzahl der Klassen der Partition treffen hier selbstverständlich auch zu. Alternativ hätte man zum Beispiel auch folgende Formel verwenden können:

$$\exists v_1, v_2 \colon \mathsf{cur}(v_1) \wedge v_1 \neq v_2 \wedge \neg \mathsf{root}(v_2) \wedge \mathsf{downStar}(v_2, v_1).$$

Sie wertet zu wahr aus, wenn es eine Ecke v_1 gibt, auf die der Zeiger cur zeigt, und wenn es eine weitere Ecke gibt, die auf dem Weg von der Wurzel des Baumes zu v_1 liegt aber nicht die Wurzel ist.

Wir verfolgen diese Anwendung weiter. Wir können die Menge der Shapegraphen so in disjunkte Teilmengen zerlegen wollen, dass in jeder Teilmenge die Shapegraphen

mit gleicher Wegstruktur enthalten sind. Das Vorhaben lässt sich mit diesem Ansatz realisieren. Eine Möglichkeit besteht darin, die Formeln des Formelpartitionierers auf folgende Weise aufzubauen:

- $\exists v_1 : \mathsf{root}(v_1)$
- $\exists v_1, v_2 : \mathsf{root}(v_1) \wedge \mathsf{cur}(v_2)$
- $\exists v_1, v_2 : v_1 \neq v_2 \wedge \mathsf{root}(v_1) \wedge \mathsf{cur}(v_2)$
- $\exists v_1, v_2, v_3 : v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge v_2 \neq v_3 \wedge \mathsf{root}(v_1) \wedge \mathsf{cur}(v_2) \wedge \mathsf{reach}[\mathsf{root}](v_3)$
- ...

Die erste Formel spezifiziert, dass es ein Heapelement geben soll, auf das der Zeiger root zeigt, der Baum also eine Wurzel hat und damit nicht leer ist. Mit dieser Formulierung soll Folgendes ausgedrückt werden: Die Partition, die durch die erste Formel induziert wird, besitzt eine Klasse, in der alle Shapegraphen enthalten sind, die eine Ecke besitzen, für die das Prädikat root zu wahr auswertet. Sie besitzt eine Klasse, welche die Shapegraphen enthält, für die diese Bedingung (definitiv) falsch ist. (Normalerweise werten die Formeln zu einem definiten Wahrheitswert aus, es gibt also normalerweise keine Klasse, die dem Wahrheitswert $\frac{1}{2}$ entspricht.)

Die zweite Formel spezifiziert, dass es ein (nicht notwendig von der Wurzel verschiedenes) Element geben soll, auf das der Zeiger cur zeigt. Die dritte Formel spezifiziert, dass diese beiden Heapelemente außerdem verschieden sein sollen. Die vierte Formel spezifiziert, dass es ein weiteres, von den bisherigen verschiedenes Heapelement geben soll, dass von der Wurzel aus erreichbar ist. Die Formeln in diesem Beispiel sind so aufgebaut, dass in jeder Formel die Situation der Vorgängerformel wiederhergestellt und durch weitere Bedingungen ergänzt wird. Nach diesem Prinzip kann die Formelmenge erweitert werden, bis die durch sie induzierte Partition als Klassen die gewünschten Wegstrukturen aufweist.

Die Zusammenstellung eines solchen Formelpartitionierers erweist sich als recht umständlich und schwerfällig. Um eine geeignete Formelmenge zu spezifizieren, benötigt man klare Vorstellungen über das Aussehen der Wegstrukturen. Diese Arbeit kann der Visualisierungsinszenator im Vorfeld der Visualisierung erledigen, er wird über die nötigen Hintergrundkenntnisse verfügen. Einem wenig erfahrenen Benutzer wird es allerdings in der Regel schwerfallen, während der Visualisierung Änderungen an der Partitionierung vorzunehmen. In jedem Fall sollte man auch den Zeitbedarf für das Zusammenstellen des Formelpartitionierers nicht außer Acht lassen. Bei dieser Anwendung kann man der formelbasierten Partitioniermethode schwerlich das Prädikat praktisch zusprechen.

Unsere Absicht war, eine Menge von Shapegraphen so in disjunkte Teilmengen zu zerlegen, dass für jede Teilmenge die Shapegraphen dieser Teilmenge einander ähnlich sind. Dabei haben wir Ähnlichkeit über Formeln definiert und die Partitionierung auf diese Formeln gestützt. Wir haben Shapegraphen primär als logische

Strukturen angesehen und behandelt. Diese logische Sicht resultiert in einer flexiblen und mächtigen Partitionierungsmethode. Allerdings haben wir gesehen, dass es für Visualisierungszwecke wünschenswerte Partitionen gibt, die sich auf diese Weise nur umständlich spezifizieren lassen, unter diesem Blickwinkel offenbart der Ansatz Schwächen. Die formelbasierte Methode ist praktikabel, aber nicht immer praktisch.

6.3 Partitionieren von Shapegraphenmengen mittels Teilstrukturen

In diesem Kapitel werden Themen in Bezug auf die Partitionierung von Shapegraphenmengen behandelt. Im vorangegangenen Abschnitt 6.2 haben wir Shapegraphen als logische Strukturen, als Objekte der Logik, behandelt und bei der Partitionierung diese logische Sichtweise betont. Bei dem dort zuletzt betrachteten Anwendungsbeispiel wollten wir die Klasseneinteilung hinsichtlich der Wegstruktur, hinsichtlich der Gestalt des Suchweges von der Wurzel zur aktuellen Ecke, vornehmen. Die Klassen sollen also Shapegraphen mit „ähnlichen“ Suchwegen zusammenfassen. Hier ließ die Klasseneinteilung mittels Formelpartitionierer etwas an Praktikabilität zu wünschen übrig.

Bei diesem Anwendungsbeispiel ist es vorteilhafter, Shapegraphen stattdessen als (beschriftete) Graphen aufzufassen. Bei hinreichend präziser Analysespezifikation, die normalerweise gegeben ist, entsprechen diesen Wegen von der Wurzel zur aktuellen Ecke ebenfalls Wege im Shapegraphen. Ein Weg in einem Graphen ist ein Teilgraph dieses Graphen. Unser Anwendungsbeispiel bedeutet in graphentheoretischer Formulierung, dass wir die Klasseneinteilung hinsichtlich der Struktur von bestimmten Teilgraphen vornehmen wollen. In jeder Klasse sind also die Shapegraphen mit „gleichen“ beziehungsweise „ähnlichen“ Teilgraphen zusammengefasst.

6.3.1 Auszeichnung von Teilstrukturen

Die Teilgraphen zeichnen wir durch Bezugnahme auf die zur Analyse verwendeten Prädikate aus. Diese sind null-, ein- oder zweistellig. Nullstellige Prädikate beschreiben Eigenschaften von (des ganzen) Shapegraphen, für die Charakterisierung von Teilstrukturen sind sie daher nicht geeignet. Einstellige Prädikate beschreiben Eigenschaften von Heapzellen, also von Ecken des Shapegraphen. Zweistellige Prädikate beschreiben Beziehungen zwischen Heapzellen, somit zwischen Ecken des Shapegraphen. Folglich spiegeln sich in der Stelligkeit eines Prädikats strukturell verschiedene Eigenschaften eines Shapegraphen wider. Das wirkt sich auf die (Möglichkeiten zur) Definition der Teilstruktur aus. Bei der Auswahl der Ecken des auszuzeichnenden Teilgraphen wird man sich primär auf die unären Prädikate stützen

wollen, die Kanten wird man primär durch die binären Prädikate charakterisieren wollen.

Wenden wir uns als Erstes den Ecken zu. Wir wünschen eine Methode, die in Shapegraphen die Auswahl von Ecken erlaubt und sich dabei auf die unären Prädikate stützt. Wir bewerkstelligen dies mit Hilfe einer vom Anwender zu spezifizierenden Formel. Es mag hilfreich sein, wenn wir dabei zunächst an eine Formel denken, die nur einstellige Prädikate enthält. Für einen gegebenen Shapegraphen werten wir die Formel für jede Ecke aus, und wir nehmen eine Ecke in die Teilstruktur (des Shapegraphen) auf, wenn die Formel für diese Ecke zu wahr auswertet. Bevor wir uns mit der Auswahl der Kanten beschäftigen, diskutieren wir einige Aspekte des Themas Eckenauswahl und des vorgeschlagenen Verfahrens.

Unser Vorgehen impliziert eine Bedingung, die wir an die Eckenauswahlformel stellen müssen: Sie muss für jede Ecke (jedes Shapegraphen) auswertbar sein. Dazu verlangen wir, dass die Formel höchstens eine freie Variable enthalten soll. Wenn φ die Formel und v die freie Variable bezeichnen, dann schreiben wir $\varphi(v)$ um diesen Sachverhalt zu verdeutlichen. Enthält φ keine freie Variable, dann sei v ein Variablensymbol, dass in φ nicht auftritt. Die Teilstruktur, die wir in einem Shapegraphen S auszeichnen, besteht dann genau aus den Ecken u mit $W_{S,\beta[\frac{v}{u}]}(\varphi(v)) = 1$. Ist $\varphi(v)$ eine geschlossene Formel, dann ist die Auswertung unabhängig von v und damit unabhängig von der Ecke. In diesem Fall werden entweder alle Ecken oder keine in die Teilstruktur aufgenommen. Die Auswertbarkeit ist die einzige Bedingung, die wir an die Formel $\varphi(v)$ stellen. Wir brauchen nicht zu fordern, dass sie nur unäre Prädikate enthalten solle. Beispielsweise besteht keine Veranlassung, binäre Prädikate auszuschließen, denn eines (oder auch beide) der Argumente kann durch einen Quantor gebunden sein.

Es stellt sich die Frage nach der Mächtigkeit der hier vorgeschlagenen Auswahlmethode. Wir diskutieren diesen Aspekt sogleich unter einem eingeschränkten Blickwinkel. Kann man auf diese Art überhaupt jede Teilmenge von Ecken auswählen? Wir fragen dazu noch spezieller: Ist es damit überhaupt möglich, jede beliebige Ecke einzeln „anzusprechen“? Rufen wir uns zur Beantwortung der Fragen erneut ins Gedächtnis, wie die Ecken von Shapegraphen entstehen. Zum Zwecke der Visualisierung verwenden wir bei der Shapeanalyse kanonische Abstraktion. Ecken repräsentieren damit Äquivalenzklassen von Heapzellen. Alle Heapelemente einer Klasse werten für jedes Abstraktionsprädikat zum gleichen Wahrheitswert aus. Insbesondere existiert zu je zwei Ecken also ein (Abstraktions-)Prädikat, so dass das Prädikat für beide Ecken zu verschiedenen Wahrheitswerten auswertet. Eine Ecke ist somit durch Angabe der Wahrheitswerte auf den Abstraktionsprädikaten eindeutig bestimmt. Im Prinzip sind diese Wahrheitswerte der kanonische Name der Ecke. Um mittels einer Formel eine bestimmte Ecke auszuwählen, kann man wie folgt vorgehen: Man schreibt für jedes Abstraktionsprädikat p_{abs} seinen Wert mittels einer Teilformel der Gestalt $p_{abs}(v) = a$ vor, wobei a ein Wahrheitswert ist, und bildet anschließend deren Konjunktion. Eine Teilmenge von Ecken kann dann mittels einer Disjunktion

solcher Formeln charakterisiert werden. Auf die aufgeworfene Frage kann also eine positive Antwort gegeben werden.

Wir bewerkstelligen die Eckenauswahl durch eine einzige Formel. Man könnte anfangs versucht sein, diese Methode als (unnötig) restriktiv anzusehen. Man könnte sich stattdessen vorstellen, eine Menge von Formeln zu verwenden. Dann würde man sicher verlangen, dass eine Ecke zur Teilstruktur gehören soll, wenn wenigstens eine der Formeln für diese Ecke zu wahr auswertet Dass die Beschränkung auf eine einzige Formel keine Einschränkungen hinsichtlich der Möglichkeiten zur Auswahl von Eckenteilmengen mit sich bringt, haben wir oben bereits gesehen. Auch unter praktischen Gesichtspunkten entsteht keine Beeinträchtigung. Beide Methoden sind äquivalent. Dazu müssen wir nur zeigen, dass sich jede Instanz der einen Möglichkeit in eine Instanz der anderen transformieren lässt. Wollen wir die Eckenauswahl mittels einer Menge von Formeln vornehmen, dann können wir alternativ die Disjunktion dieser Formeln bilden. Gegebenenfalls ist vorher eine Variablenumbenennung vorzunehmen, so dass in allen Formeln das gleiche Symbol für die freie Variable steht (und die übrigen Variablensymbole alle verschieden sind). So erhalten wir eine Instanz der Auswahlmethode mittels einer einzigen Formel. Die Umkehrung ist trivial.

Die Ecken der Teilstruktur – in logischer Terminologie sind das die Elemente des Universums – sind jetzt ausgewählt. Es müssen noch die Prädikate hinzugefügt werden. Dazu verwenden wir eine Teilmenge der Prädikate der Analysespezifikation. Die Teilstruktur übernimmt alle Prädikate des Shapegraphen, die in der Teilmenge enthalten sind; dabei werden sie auf die ausgewählte Eckenmenge, auf das neue (Teil-)Universum, eingeschränkt. Bei nullstelligen Prädikaten besteht natürlich keine Veranlassung, sie auf das neue Universum einzuschränken. Für die binären Prädikate bedeutet das in graphentheoretischer Terminologie, dass wir den von den ausgewählten Ecken bezüglich der binären Prädikate aufgespannten Untergraphen bilden.

Unser Vorgehen bei der Auswahl der Prädikate der Teilstruktur lässt sich durchaus allgemeiner gestalten, auf jeden Fall für die ein- und zweistelligen Prädikate. Man könnte zum Beispiel nur die Werte von unären Prädikaten für bestimmte Ecken, sowie die Werte von binären Prädikaten für bestimmte Paare von Ecken, in der Teilstruktur vorschreiben wollen. Für diese Ecken, beziehungsweise Paare von Ecken, übernimmt man die Werte aus dem Shapegraphen, und für die anderen Ecken, beziehungsweise Paare von Ecken, setzt man den Wert des Prädikats auf $\frac{1}{2}$, also gewissermaßen auf „nicht relevant“. In graphentheoretischer Sichtweise bedeutet das für ein binäres Prädikat, dass wir nicht die Kanten des von den ausgewählten Ecken (bezüglich des binären Prädikats) aufgespannten Untergraphen, sondern nur bestimmte Kanten daraus aufnehmen wollen. Die Ecken, beziehungsweise die Eckenpaare, für die wir die Prädikatswerte übernehmen wollen, können dabei natürlich von Prädikat zu Prädikat verschieden sein. Man könnte dazu jedem Argument eines unären und binären Prädikat eine Formel mit höchstens einer freien Variablen zuordnen. Der

Prädikatswert eines unären Prädikats wird für eine Ecken genau dann in die Teilstruktur übernommen, wenn die zugeordnete Formel für die Ecke zu wahr auswertet. Der Prädikatswert eines binären Prädikats wird für ein Eckenpaar genau dann in die Teilstruktur übernommen, wenn die zugeordneten Formeln für die Ecken zu wahr auswerten. Mit dieser Verallgemeinerung lassen sich sowohl mehr als auch feinere Teilstrukturen realisieren, es entstehen mehr Möglichkeiten mit diesem Ansatz Unterscheidungen herbeizuführen. Ob dieses Mehr an Möglichkeit und Flexibilität zu einem Mehr – im Sinne von Mehrwert – bei der Visualisierung führt, ist ungewiss. Unsere bisherige Erfahrung deutet darauf hin, dass diese zusätzlichen Möglichkeiten den Mehraufwand bei der Spezifikation nicht rechtfertigen. Daher verzichten wir auf eine Formalisierung dieser Verallgemeinerung.

Nachdem wir unser Vorgehen erläutert und einige damit zusammenhängende Aspekte besprochen haben, formalisieren wir den Ansatz. Wie zuvor sei die Analysespezifikation fest, und alle Objekte, die im Folgenden vorkommen, seien Objekte bezüglich dieser Spezifikation. Wir beginnen mit:

6.6 Definition. *Es sei $\varphi(v)$ eine Formel, die höchstens eine freie Variable enthalte. Ferner sei P eine Teilmenge der Analyseprädikate. Dann nennen wir das Paar $(\varphi(v), P)$ einen* Teilstrukturdefinierer.

Im vorangegangenen Abschnitt haben wir Formeln, die mit Formelpartitionierern in Verbindung stehen, mit lateinischen Buchstaben bezeichnet. Für Formeln in Teilstrukturdefinieren verwenden wir griechische Buchstaben. Auf diese Weise ist durch den visuellen Eindruck des Formelnamens sogleich eine verwendungsbezogene Zuordnung vorgegeben. Wie wir zuvor bereits erläutert haben, erlauben wir in der Formel $\varphi(v)$ eines Teilstrukturdefinierer Prädikate beliebiger Stelligkeit. In der Praxis wird man sie aber häufig so aufbauen, dass alle vorkommende Prädikatssymbole einstellig sind.

Mit Hilfe von Teilstrukturdefinierern zeichnen wir in Shapegraphen Teilstrukturen aus:

6.7 Definition. *Es sei S ein Shapegraph und $(\varphi(v), P)$ ein Teilstrukturdefinierer. Die durch $(\varphi(v), P)$ in S definierte* Teilstruktur $S_{(\varphi(v),P)}$ *hat als Universum alle Individuen u aus S mit $W_{S,\beta[\frac{v}{u}]}(\varphi(v)) = 1$. Sie übernimmt von S alle Prädikate, die in P enthalten sind, wobei sie auf das Universum der Teilstruktur eingeschränkt werden.*

Alternativ sprechen wir von $S_{(\varphi(v),P)}$ auch als der durch $(\varphi(v), P)$ in S *induzierten* Teilstruktur oder nennen sie die bezüglich $(\varphi(v), P)$ *reduzierte* Teilstruktur. – Ist $\varphi(v) = \varphi$ eine geschlossene Formel, dann enthält die Teilstruktur eines Shapegraphen entweder alle seinen Ecken oder keine. Im letzteren Fall entsteht, falls P keine nullstelligen Prädikate enthält, die leere (Teil-)Struktur.

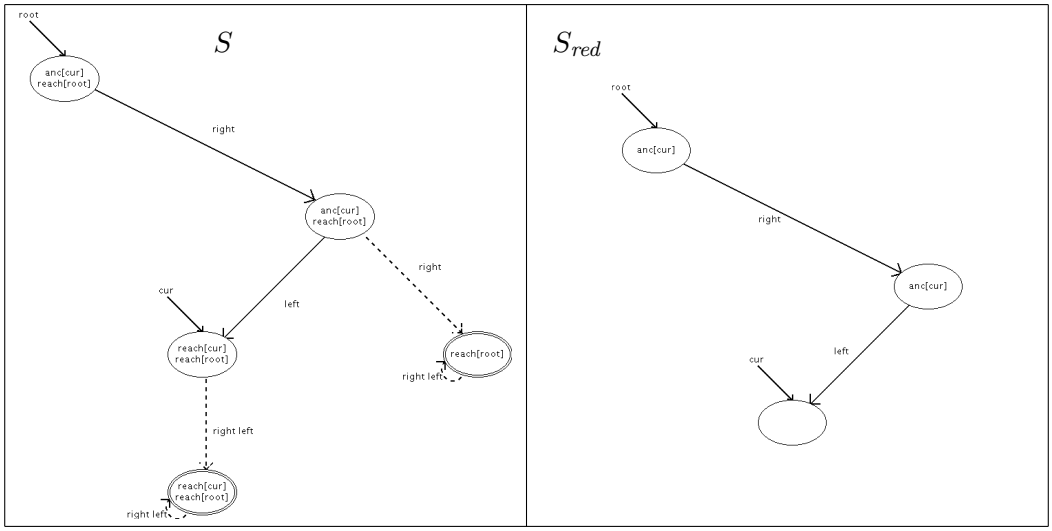

Abbildung 6.3: Ein Shapegraph und seine durch den Teilstrukturdefinierer $(\mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v), \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}, \mathsf{left}, \mathsf{right}\})$ bestimmte Teilstruktur

Wir betrachten ein Beispiel. In Abbildung 6.3 sehen wir links einen Shapegraphen S, wie er bei der Suche nach einem Element in einem Suchbaum auftritt. Wir wollen den Suchweg, den Weg von der Wurzel des Baumes zur aktuellen Ecke, als Teilgraph auszeichnen. Für die Ecken auswählende Formel verwenden wir $\varphi(v) := \mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v)$. Sie wertet für die aktuelle Ecke und für alle ihre Vorfahren zu wahr aus. Von den Prädikaten übernehmen wir $P = \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}, \mathsf{left}, \mathsf{right}\}$. Der Graph S_{red} in Abbildung 6.3 zeigt die reduzierte Teilstruktur.

Das Verfahren, Mengen von Shapegraphen unter Bezugnahme auf einen Begriff von Ähnlichkeit in disjunkte Teilmengen zu zerlegen, wurde in [Johannes u. a. 2005] vorgeschlagen. Auch dort basiert der Ähnlichkeitsbegriff auf einer Ähnlichkeit hinsichtlich der Gestalt von (Teilen eines) Shapegraphen. Das Konzept der Teilstruktur wird dort ebenfalls eingeführt. Ihre Definition erfolgt in ähnlicher Weise, die Auswahl der Ecken wird allerdings restriktiver mittels einer (eventuell leeren) Menge von einstelligen Prädikatssymbolen vorgenommen. Eine Ecke u wird in die Teilstruktur (eines Shapegraphen) aufgenommen, wenn es wenigstens ein Prädikat p dieser Menge gibt, so dass p für u potentiell wahr ist, so dass also $W_{S,\beta[\frac{v}{u}]}\big(p(v)\big) \geq \frac{1}{2}$ gilt. Die Angabe der Menge $\{p_1, p_2, \ldots, p_n\}$, $n \geq 1$, korrespondiert zur Wahl der Eckenauswahlformel

$$\varphi(v) := \left(p_1(v) \geq \frac{1}{2}\right) \vee \left(p_2(v) \geq \frac{1}{2}\right) \vee \cdots \vee \left(p_n(v) \geq \frac{1}{2}\right),$$

zur leeren Menge korrespondiert die Formal $\varphi(v) := 0$.

6.3.2 Teilstrukturbasierte Ähnlichkeit

Das Ziel dieses Abschnitts ist, eine Menge von Shapegraphen in Klassen einzuteilen. Die Einteilung soll auf der Gestalt der Shapegraphen, auf der Gestalt ausgewählter Bestandteile, beruhen. Shapegraphen innerhalb einer Klasse sollen in der Struktur von Teilgraphen übereinstimmen oder sich darin ähneln. Wir haben dazu den Begriff der Teilstruktur eingeführt, um dies formal zu konkretisieren. Nun werden wir daraus einen Ähnlichkeitsbegriff ableiten. Ein naheliegendes Vorgehen ist, zwei Shapegraphen ähnlich (bezüglich eines Formelpartitionierers) zu nennen, wenn ihre (durch den Formelpartitionierer definierten) Teilstrukturen isomorph sind. Aber ist das wirklich, was wir wollen?

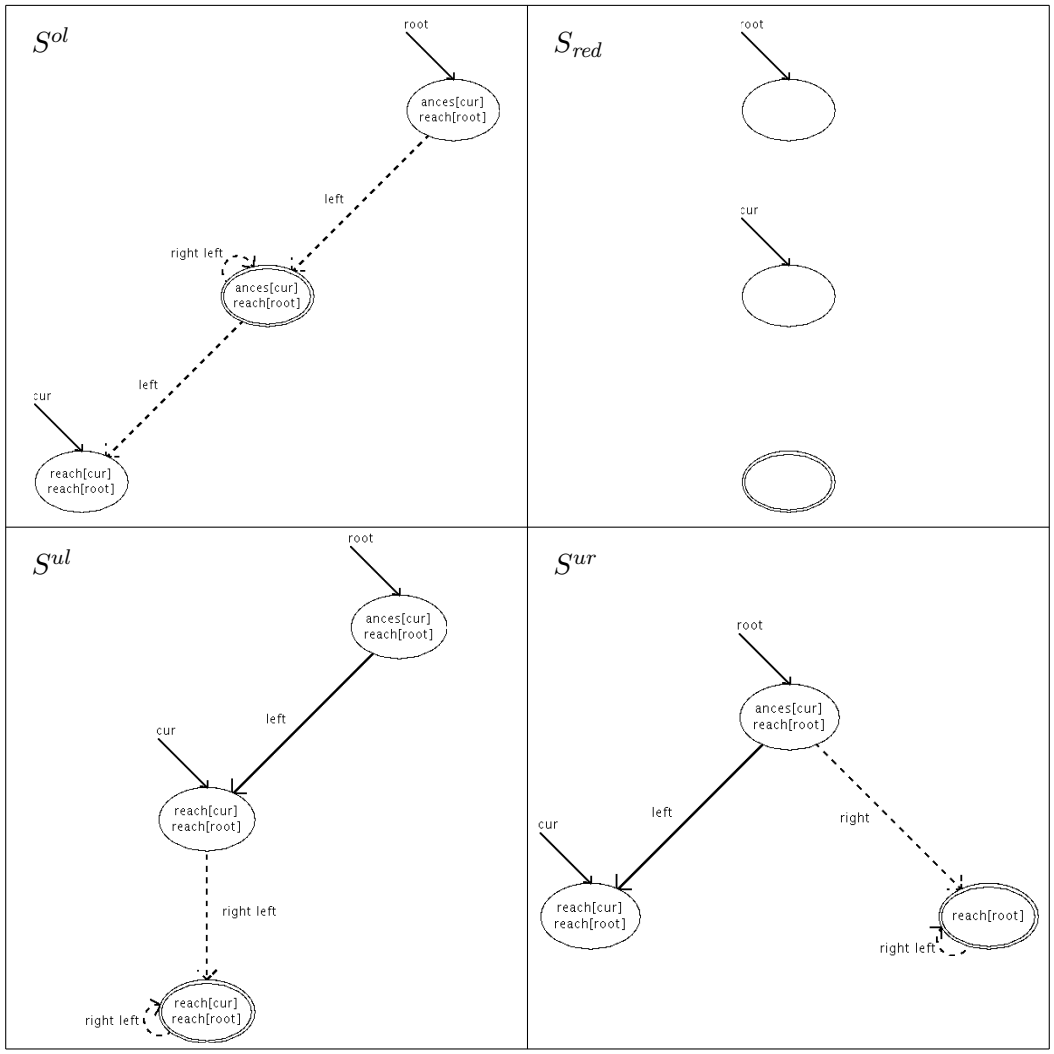

Abbildung 6.4: Drei Shapegraphen (S^{ol}, S^{ul}, S^{ur}) mit bezüglich $(1, \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}\})$ isomorphen Teilstrukturen (S_{red})

Wir vergegenwärtigen uns die Situation an einem Beispiel. Dazu greifen wir wieder das Suchbeispiel auf, wobei wir uns für die Struktur des Weges von der Wurzel zur aktuellen Ecke interessieren. Wir befinden uns an einem Programmpunkt in der Suchschleife. Der Shapegraphen S^{ol}, der oben links in Abbildung 6.4 gezeigt ist, tritt dort auf. Wir wählen als Teilstrukturdefinierer $(1, \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}\})$. Unter praktischen Gesichtspunkten wirkt er recht gekünstelt, aber er führt uns direkt zum Kern des Problems. Bestimmen wir die durch den Teilstrukturdefinierer in S^{ol} ausgezeichnete Teilstruktur. Die Ecken auswählende Formel ist 1, sie wertet für jede Ecke zu wahr aus; damit enthält die Teilstruktur alle Ecken des Shapegraphen. Von den Prädikaten werden nur root und cur übernommen. Die Teilstruktur S_{red} ist oben rechts in Abbildung 6.4 gezeigt. In der unteren Zeile in Abbildung 6.4 sind zwei weitere Shapegraphen, S^{ul} und S^{ur}, gezeigt, die ebenfalls am selben Programmpunkt auftreten. Ihre durch den Teilstrukturdefinierer bestimmten Teilstrukturen sind ebenfalls (isomorph zu) S_{red}, folglich gilt

$$S^{ol}_{(1,\{\mathsf{root},\mathsf{cur}\})} \simeq S^{ul}_{(1,\{\mathsf{root},\mathsf{cur}\})} \simeq S^{ur}_{(1,\{\mathsf{root},\mathsf{cur}\})} \simeq S_{red}.$$

In allen drei Shapegraphen gibt es (genau) eine Summary-Ecke. In jedem Graphen repräsentiert sie einen anderen Teil eines Baumes: in S^{ol} ist sie ein Teil des Suchweges, in S^{ul} ist sie der Teil des Baumes, der noch nicht traversiert wurde und in S^{ur} der Teilbaum der Wurzel, in dem der Suchweg nicht verläuft. In der/den Teilstruktur(en) ist dieser Ursprung verloren gegangen. Alle drei Shapegraphen haben isomorphe Teilstrukturen, aber wollen wir sie als „ähnlich“ ansehen? Die Struktur des Suchweges ist jedenfalls in allen drei Graphen verschieden, die Antwort ist somit ein klares Nein. Ganz allgemein dürfte in den meisten (Anwendungs-)Fällen die Antwort negativ ausfallen.

Wenn wir die ursprünglichen Shapegraphen miteinander vergleichen, dann sind die Summary-Ecken durchaus voneinander unterscheidbar. Das lässt sich auf mehrere Arten zeigen. Wir können die Wurzel und die aktuelle Ecke als Referenzecken verwenden und anhand der Zeigerstruktur (mittels der Prädikate left und right) argumentieren, dass es sich bei den Summary-Ecken um verschiedene Teile des Baumes handelt. Alternativ können wir die Wahrheitswerte der (Abstraktions-)Prädikate anc[cur] und reach[cur] für die Summary-Ecken betrachten. In S^{ol} sind sie, wenn wir sie in dieser Reihenfolge als Tupel schreiben, $(1, 0)$, für die Summary-Ecke in S^{ul} sind sie $(0, 1)$ und in S^{ur} sind sie $(0, 0)$. Die kanonischen Namen der drei Ecken (in ihren Ursprungsshapegraphen) sind also verschieden, sie repräsentieren unterschiedliche Teile des Baumes, unterschiedliche Teile in Relation zum Suchweg. Bei der Spezifikation des Teilstrukturdefinierers haben wir diejenigen unären Prädikate weggelassen, für die sich die kanonischen Namen der Summary-Ecken unterscheiden. Ebenso haben wir die Prädikate fortgelassen, die die Zeigerstruktur beschreiben. Deshalb können die Summary-Ecken in den Teilstrukturen nicht mehr voneinander unterschieden werden. In den meisten (Anwendungs-)Fällen wünschen wir aber, dass die Shapegraphen nicht nur isomorph sind, sondern die Isomorphie auch kanonische Namen invariant lässt.

Wir verwenden das Konzept der Teilstruktur, um auf Mengen von Shapegraphen eine (Ähnlichkeits-)Relation zu definieren:

6.8 Definition. *Es sei $(\varphi(v), P)$ ein Teilstrukturdefinierer, ferner seien S und T zwei Shapegraphen.*

1. *Wir nennen S und T* schwach ähnlich bezüglich $(\varphi(v), P)$*, wenn ihre durch $(\varphi(v), P)$ definierten Teilstrukturen isomorph sind, wenn $S_{(\varphi(v),P)} \simeq T_{(\varphi(v),P)}$ gilt.*
2. *Wir nennen S und T* ähnlich bezüglich $(\varphi(v), P)$*, wenn:*
   a) *Die Shapegraphen S und T sind schwach ähnlich bezüglich $(\varphi(v), P)$.*
   b) *Es existiert ein Isomorphismus f von $S_{(\varphi(v),P)}$ auf $T_{(\varphi(v),P)}$ so, dass für jede Ecke v von $S_{(\varphi(v),P)}$ der kanonische Name von v in S derselbe wie der kanonische Name von $f(v)$ in T ist.*

   Sind S und T ähnlich bezüglich $(\varphi(v), P)$, dann schreiben wir $S \sim_{(\varphi(v),P)} T$.

Ähnliche Shapegraphen sind auch schwach ähnlich, die Forderung nach Ähnlichkeit ist eine stärkere, eine einschränkendere. Im Hinblick auf die Visualisierung von Shapegraphen erweist sie sich als nützlicher. Daher werden wir uns im Folgenden auf diesen Ähnlichkeitsbegriff beschränken.

Wir betrachten wieder unser Suchbeispiel, wir befinden uns an einem Programmpunkt in der Suchschleife. In Abbildung 6.5 sehen wir vier Shapegraphen, die dort auftreten. Der Weg von der Wurzel zur aktuellen Ecke hat in allen Graphen dieselbe Struktur, sie ist als S^{or} oben rechts gezeigt. (Wir kennen diesen Shapegraphen bereits aus Abbildung 6.4.) Wir wollen einen Formelpartitionierer spezifizieren, der genau diesen Weg als Teilstruktur in den Shapegraphen auszeichnet. Für die Auswahl der Ecken verwenden wir $\varphi(v) := \mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v)$, und von den Prädikaten übernehmen wir diejenigen aus $P = \{\mathsf{root}, \mathsf{cur}, \mathsf{reach}[\mathsf{root}], \mathsf{reach}[\mathsf{cur}], \mathsf{sm}, \mathsf{anc}[\mathsf{cur}], \mathsf{left}, \mathsf{right}\}$. Die von diesem Teilstrukturdefinierer in den vier Shapegraphen ausgezeichneten Teilgraphen sind jeweils (isomorph zu) S^{or}. (Beim Shapegraphen S^{or} stimmt der ausgezeichnete Teilgraph also mit dem Graphen überein.) Alle hierzu ähnlichen Shapegraphen haben dieselbe Suchwegstruktur. Sie unterscheiden sich (nur) in den anderen Baumteilen: Es kann wie in S^{ol}, S^{ul} und S^{ur} eine Summary-Ecke geben, die die am Suchweg herabhängenden Teilbäume repräsentiert. Hierbei gibt es verschiedene Möglichkeiten: Es kann sich um Teilbäume der Wurzel, um Teilbäume der inneren Ecke des Weges (S^{ol}, S^{ul}) oder um Teilbäume von beiden Ecken (S^{ur}) handeln, im zweiten (und dritten) Fall kann es sich um linke, rechte (S^{ol}) oder sowohl um linke und rechte Teilbäume (S^{ul}, S^{ur}) handeln. Weiterhin kann es wie in S^{ol} und S^{ur} eine Summary-Ecke geben, die den noch nicht traversierten Teil des Baumes, die Teilbäume der Kinder von cur, repräsentiert. – Der angegebene Teilstrukturdefinierer ist mitnichten eindeutig. Wir hätten dasselbe Resultat auch anderes erhalten können, was einige Beispiele zeigen sollen. Wir hätten $\mathsf{root}(v)$ per

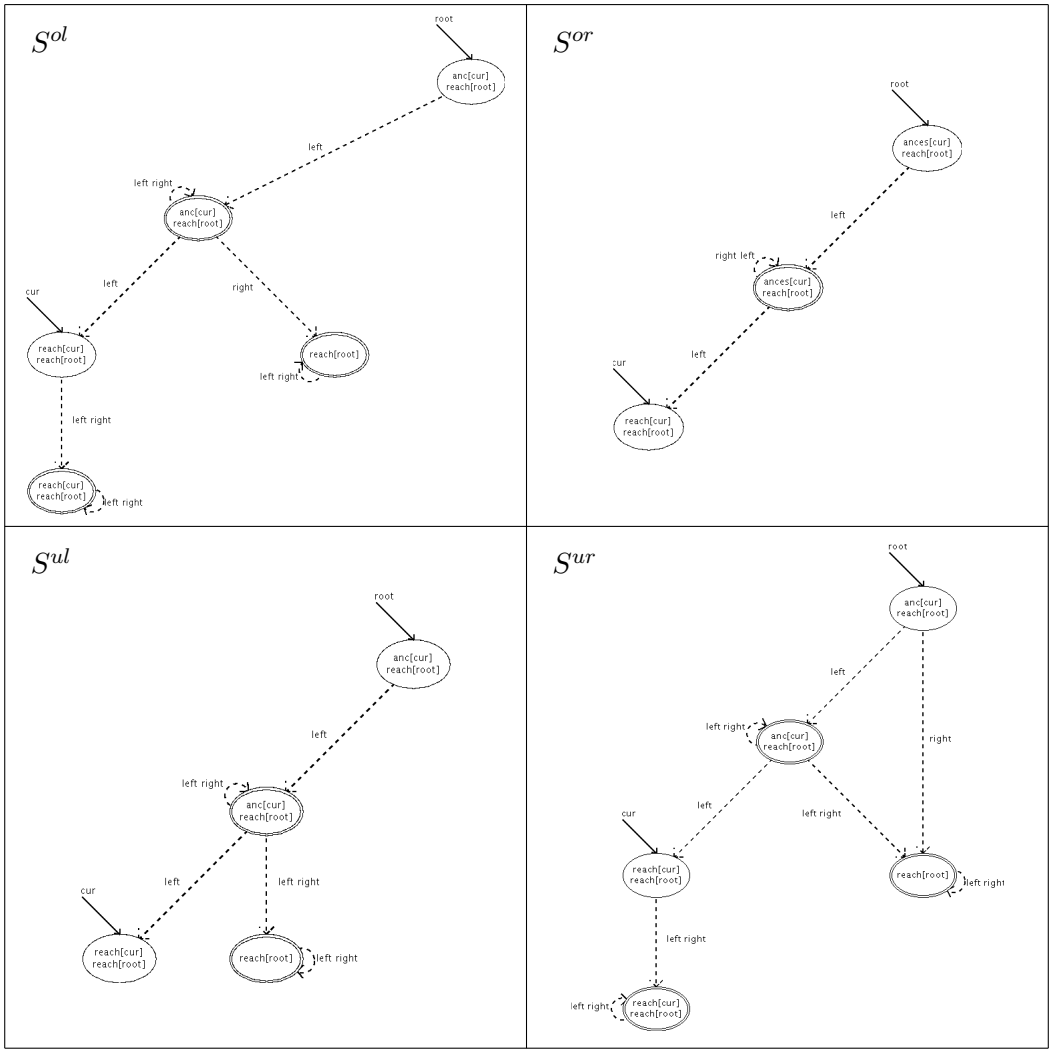

Abbildung 6.5: Vier bezüglich des Teilstrukturdefinierers $(\mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v), \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}, \mathsf{reach}[\mathsf{root}], \mathsf{reach}[\mathsf{cur}], \mathsf{anc}[\mathsf{cur}], \mathsf{left}, \mathsf{right}\})$ ähnliche Shapegraphen

Disjunktion in $\varphi(v)$ aufnehmen können. Wir hätten $\varphi(v)$ auf andere Art formulieren können, beispielsweise $\varphi(v) := \exists v_1 \mathsf{cur}(v_1) \wedge \mathsf{downStar}(v, v_1) \geq \frac{1}{2}$ (die Ecke, auf die cur zeigt, ist von v aus erreichbar). Wir hätten in P das Prädikat reach[root] fortlassen oder anc[root] hinzufügen können.

Unsere Vorstellung von Ähnlichkeit von Shapegraphen haben wir letztendlich unter Zuhilfenahme des Isomorphiekonzepts formalisiert. Den Begriff der schwachen Ähnlichkeit haben wir durch Isomorphie von ausgezeichneten Teilstrukturen ausgedrückt. Isomorphie (von logischen Strukturen) ist eine Äquivalenzrelation. Diese Eigenschaft überträgt sich unmittelbar auf die Relation der schwachen Ähnlichkeit.

Bei dem Begriff der Ähnlichkeit tritt eine weitere Bedingung hinzu. Sie hat aber, wie man leicht nachrechnet, keinen Einfluss auf die Reflexivität, Symmetrie und Transitivität der schwachen Ähnlichkeit. Damit gilt:

6.9 Satz. *Es sei $(\varphi(v), P)$ ein Teilstrukturdefinierer. Ähnlichkeit bezüglich $(\varphi(v), P)$ ist eine Äquivalenzrelationen auf der Menge aller Shapegraphen bezüglich der gewählten Analysespezifikation.*

Mittels dieser Äquivalenzrelation erzeugen wir eine Partition. Methodisch ist es das gleiche Vorgehen wie im vorangegangenen Teilabschnitt, in dem wir Partitionen durch Verwendung von Formelpartitionierer erzeugt haben. Es sei $\mathcal{M}$ eine (endliche und nicht leere) Menge der Shapegraphen, ferner sei $(\varphi(v), P)$ ein Teilstrukturdefinierer. Für einen Shapegraphen $S \in \mathcal{M}$ bezeichnet wie üblich $[S]_{\sim_{(\varphi(v),P)}} = \{T\colon T \in \mathcal{M} \wedge T \sim_{(\varphi(v),P)} S\}$ die durch S repräsentierte Klasse. Wir schreiben wieder einfacher $[S]$, wenn nicht die Gefahr einer Verwechselung besteht. Genauso wie im vorangegangenen Abschnitt sehen wir, dass die Familie $\{[S]\colon S \in \mathcal{M}\}$ der Klassen eine Partition der Menge $\mathcal{M}$ bildet. Wir nennen sie die durch $\sim_{(\varphi(v),P)}$ induzierte Partition und schreiben sie als $\mathcal{M}/_{\sim_{(\varphi(v),P)}}$.

Es gibt zwei extreme Partitionen: alle Shapegraphen werden in einer einzigen Klasse gesammelt, die Partition besteht also aus genau einer Klasse, und jeder Shapegraph gehört zu einer anderen Klasse, jede Klasse besteht also aus genau einem Shapegraphen. Wir untersuchen, ob und wie wir diese Partitionen mittels der teilstrukturbasierten Methode erzeugen können. – Wenn die Partition nur aus einer Klasse bestehen soll, dann müssen alle Shapegraphen einander ähnlich sein. Wir suchen einen Teilstrukturdefinierer, so dass alle Shapegraphen (bezüglich der verwendeten Analysespezifikation) denselben reduzierten Teilgraphen haben. Das wird insbesondere dann erreicht, wenn wir den Teilstrukturdefinierer so wählen, dass für jeden Shapegraphen als Teilstruktur die leere Struktur entsteht. Dazu setzen wir $\varphi(v) = 0$. Für jeden Shapegraphen gilt, dass keine seiner Ecken in die reduzierte Teilstruktur aufgenommen wird. Die zu spezifizierende Prädikatsmenge kann recht beliebig gewählt werden, es dürfen aber keine nullstelligen Prädikate enthalten sein. Wir können $P = \emptyset$ wählen. Dann entsteht für jeden Shapegraphen als Teilstruktur die leere Struktur.

Im anderen Extrem soll jeder Shapegraph in (s)einer eigenen Klasse enthalten sein. Je zwei Shapegraphen dürfen sich also nicht ähnlich sein. Das lässt sich erreichen, wenn für jeden Shapegraphen gilt, dass er mit seiner reduzierten Struktur übereinstimmt. Dazu müssen wir in jedem Shapegraphen alle Ecken auswählen, wir verwenden $\varphi(v) = 1$. Des Weiteren müssen wir alle Prädikate übernehmen, wir setzen P gleich der Menge der Analyseprädikate (einschließlich sm). Wenn zwei (oder mehr) Shapegraphen in einer Klasse enthalten sind, dann müssen sie (paarweise) isomorph sein. Dieser Fall tritt nicht auf, da TVLA als Ausgabe für jeden Programmpunkt eine Liste nicht isomorpher Shapegraphen erzeugt. – Diese Partition entspricht dem Fall, dass wir die teilstrukturbasierte Partitionierung nicht anwenden: Es werden keine

Shapegraphen als ähnlich zusammengefasst, jeder steht für sich. Die teilstrukturbasierte Partitionierungsmethode schließt in harmonischer Weise ihre Nichtnutzung mit ein, und zwar, im Gegensatz zur formelbasierten Methode, auf elegante, leicht zu spezifizierende Weise.

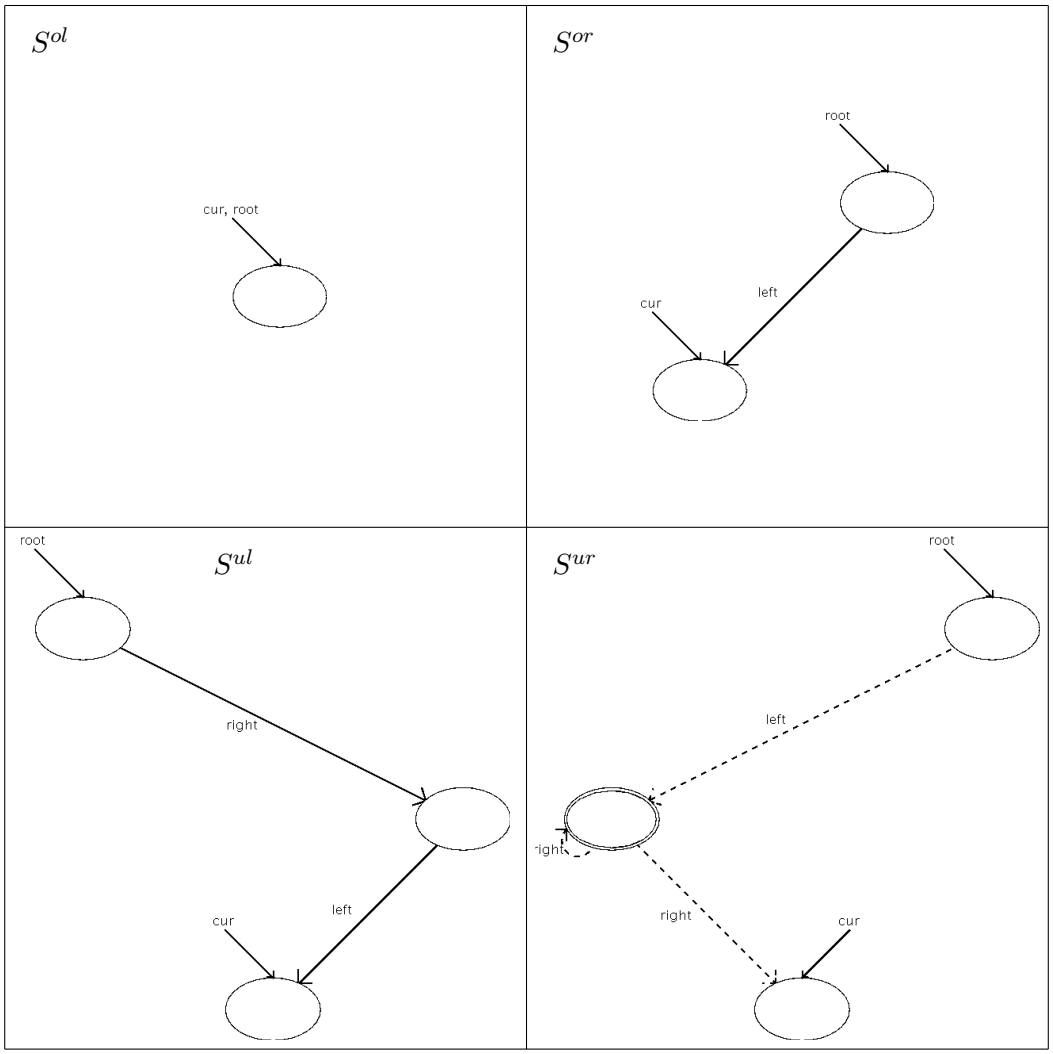

Abbildung 6.6: Verschiedene Strukturen des Suchweges

6.3.3 Anwendungsbeispiele

Nachdem wir die Methode formal entwickelt haben, greifen wir das Suchbeispiel wieder auf. Wir befinden uns am Programmpunkt n2, das ist der Eingangspunkt der Suchschleife. Unser Augenmerk gilt wieder der Struktur des Weges von der Wurzel zur aktuellen Ecke. Wir wollen die Shapegraphenmenge dieses Programmpunktes

nach der Struktur des Suchweges in Klassen einteilen. Die Shapegraphen jeder Klasse sollen in der Gestalt des Weges übereinstimmen. Dazu müssen wir einen Teilstrukturdefinierer finden, der in einem Shapegraphen gerade diesen Suchweg als Teilstruktur auszeichnet. Wir beginnen mit der Spezifikation der Eckenauswahlformel. Wir benötigen (unäre) Prädikate, die zur Charakterisierung des Suchweges geeignet sind. Sicher gehört, sofern vorhanden, das Heapelement, auf das der Zeiger cur zeigt, zu dem Weg, weiterhin alle seine Ahnen. Wir benutzen deshalb neben cur auch das Prädikat anc[cur]. Für den Fall, dass es kein Element gibt, auf das cur zeigt, soll der Weg leer sein; dies ist bei einer eventuellen Verwendung des Prädikats root zu berücksichtigen. Eine adäquate Eckenauswahlformel ist folglich $\varphi(v) := \mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v)$. Als Zweites müssen wir die Prädikate festlegen. Die Teilstruktur soll auf jeden Fall die Zeigerstruktur, sowohl des Heaps als auch der Variablen, widerspiegeln. Die diese Zeigerstruktur widerspiegelnden Prädikate zusammen mit sm genügen bereits. Wir setzen $P = \{\mathsf{root}, \mathsf{cur}, \mathsf{sm}, \mathsf{left}, \mathsf{right}\}$.

In Abbildung 6.6 sehen wir vier Beispiele für Wegstrukturen der von dem Teilstrukturdefinierer induzierten Partition. Sie lassen sich leicht anhand eines Programmlaufes nachvollziehen. Wir können uns am Anfang des Programmablaufes befinden, der Zeiger cur wurde gerade auf die Wurzel gesetzt. Der Shapegraph S^{ol} zeigt diese Situation. Nach einer Iteration ist cur ein Kind der Wurzel, entweder das linke oder das rechte. Der Graph S^{or} zeigt ersteren Fall. Hier entstehen zwei Klassen. Nach einer weiteren Iteration ist cur ein Enkel der Wurzel, der Graph S^{ul} ist ein Beispiel. Bei den beiden Kanten des Weges kann es sich jeweils um left oder right handeln, hier entstehen vier Klassen. Nach einer weiteren und allen folgenden Iterationen erscheint im Inneren des Suchweges eine Summary-Ecke, der Graph S^{ur} ist ein Beispiel für diesen Fall. Die Kanten des Suchweges können wieder jeweils mit left oder right beschriftet sein, die Schleife der inneren Summary-Ecke auch mit beidem, hier entstehen 12 Klassen. Des Weiteren gibt es noch den Fall, dass cur in der letzten Iteration auf ein Blatt des Baumes zeigte und nun der Nullzeiger ist. Insgesamt enthält die Partition 20 Klassen. Im Vergleich zu den 235 am Programmpunkt n2 auftretenden Shapegraphen ist dies eine viel übersichtlichere Anzahl.

Die in diesem Abschnitt betrachtete Partitionierungsmethode erweist sich für diese Beispielanwendung als sehr geeignet. Dies gilt allgemein, wenn die Partitionierung im Hinblick auf der Gestalt von Teilstrukturen, von Teilgraphen, erfolgen soll. Die Partitionierungsmethode unterstützt eine visuelle Sichtweise von logischen Strukturen. Viele Anwender werden sie als intuitiver als die formelbasierte Partitioniermethode empfinden. Die Spezifikation des Teilstrukturdefinierers ist einfach und geht schnell, sowohl relativ im Vergleich zur Spezifikation eines Formelpartitionierers als auch absolut. Eine weitere positive Eigenschaft ist, dass man im Vorfeld keine übermäßig konkreten Vorstellungen besitzen muss, wie die Partitionierung am Ende aussehen soll. Damit ist diese Partitionierungsmethode auch für einen ungeübten Benutzer verwendbar.

Am Ende von Abschnitt 6.2 haben wir diverse Beispiele betrachtet, um den Nut-

zen der formelbasierten Partitioniermethode zu demonstrieren. Wir greifen sie hier wieder auf und zeigen, dass und wie diese Partitionierungen mit der teilstrukturbasierten Methode realisiert werden können. In einem Beispiel waren wir an strukturbedingten Abbruchfällen interessiert. Wir können die Suchschleife in zwei Fällen verlassen: Wir können das gesuchte Element finden, was in jeder Schleifeniteration geschehen kann; dies ist unabhängig vom der Zeigerstruktur des Heaps und der Variablen, also unabhängig vom Shapegraphen. Wir können uns auch in einem Blatt des Baumes befinden, die aktuelle Ecke cur hat keine Kinder. Dann beenden wir die Schleife und haben das gesuchte Element nicht gefunden. Das sind die strukturbedingten Abbruchfälle. Es geht also darum zu unterscheiden, ob die aktuelle Ecke Kinder hat oder nicht. Wir müssen also denjenigen Teilgraphen auszeichnen, der dem Teilbaum mit Wurzel cur entspricht. Dazu gehören genau diejenigen Ecken, die von cur aus erreichbar sind. Bei der Spezifikation der Prädikate kommt es nicht sonderlich drauf an, was wir wählen. Der Teilstrukturdefinierer $(\mathsf{reach}[\mathsf{cur}](v), \emptyset)$ leistet das Geforderte. Wünscht man Teilstrukturen mit mehr Information, kann man auch Prädikate übernehmen. (Solange wir nicht root verwenden, erhalten wir mit der üblichen Analysespezifikation dieselbe Partition.) Es entsteht eine Partition mit drei Klassen: Eine Klasse enthält die Shapegraphen, deren induzierte Teilstruktur die leere Struktur ist; dies ist nur der Startshapegraph. Die zweite Klasse enthält die Shapegraphen, deren induzierte Teilstruktur aus einer Ecke, auf die dann notwendigerweise cur zeigt, besteht. Das sind die Schleifenabbruchfälle. Die dritte Klasse enthält die Shapegraphen, deren induzierte Teilstruktur aus zwei Ecken besteht, cur und denen, die von ihr erreichbar sind. Bei unserer Analysespezifikation handelt es sich dabei zwangsläufig um eine Summary-Ecke. (Wäre root in der Prädikatsmenge enthalten, dann entstünden vier Klassen: Die Klasse, die der zweieckigen Teilstruktur entspricht, teilt sich; die Zeiger root und cur können auf dieselbe Ecke zeigen oder nicht.) – In Abschnitt 6.2 haben wir eine Klasseneinteilung mit zwei anstelle von drei Klassen erhalten. Auch dies lässt sich realisieren. Dazu können wir für den Teilstrukturdefinierer $\varphi(v) := \exists v_1 \colon \mathsf{cur}(v_1) \wedge v \neq v_1 \wedge \mathsf{reach}[\mathsf{cur}](v)$ und eine nahezu beliebige Menge von Prädikaten (keine nullstelligen), beispielsweise wieder $P = \emptyset$, verwenden.

Ein anderes Anwendungsbeispiel bestand darin, Shapegraphen, deren Suchweg einen „allgemeinen“ Charakter hat, in einer Klasse zu sammeln. Die Bedingung hatten wir dahingehend konkretisiert, dass der Suchweg aus (wenigstens) drei Ecken bestehen soll. (Er besteht dann aus genau drei Ecken.) Die Eckenauswahlformel muss genau für die Ecken des Suchweges zu wahr auswerten, zum Beispiel leistet $\varphi(v) := \mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v)$ das Gewünschte. Für die Prädikatsmenge kann $P = \emptyset$ gewählt werden. Die entstehende Partition hat vier Klassen. Die jeweiligen reduzierten Teilstrukturen sind die leere Struktur, ein Graph mit einer Ecke, ein Graph mit zwei isolierten Ecken und ein Graph mit drei isolierten Ecken. Die Klasse, die dem letzten Teilgraphen entspricht, enthält alle Shapegraphen mit dem gewünschten „allgemeinen“ Charakter.

In Abschnitt 6.2 haben wir eine solche Partition mit zwei Klassen realisiert. Man

könnte vermuten, dass dies mit der teilstrukturbasierten Methode nicht möglich sei. Wenn wir den Suchweg als Teilgraphen auszeichnen, also eine Eckenauswahlformel verwenden, die genau für diese drei Ecken zu wahr auswertet, dann wertet sie auch für „jede“ Teilmenge dieser drei Ecken zu wahr aus. Ein Shapegraph, in dem der Suchweg kürzer ist, erzeugt also einen anderen reduzierten Teilgraphen. Es entstehen also mindestens so viele Klassen, wie der Suchweg lang sein kann. Diese Feststellung ist richtig. Allerdings ist a priori nicht klar, das wir den Suchweg als Teilstruktur auszeichnen müssen. Wir werden sogleich sehen, dass es auch anders geht. Dazu werden wir die in Abschnitt 6.2 verwendete Eckenauswahlformel „missbrauchen“. Wir übernehmen sie leicht modifiziert und setzten

$$\begin{aligned}\varphi(v) := \exists v_1, v_2, v_3 \colon\ & v_1 \neq v_2 \wedge v_1 \neq v_3 \wedge v_2 \neq v_3 \wedge \\ & \mathsf{root}(v_1) \wedge \mathsf{anc}[\mathsf{cur}](v_1) \wedge \mathsf{anc}[\mathsf{cur}](v_2) \wedge \mathsf{cur}(v_3) \wedge \\ & v = v_1\end{aligned}$$

sowie $P = \emptyset$. (Die $\varphi(v)$ definierende Formel kann vereinfacht werden, die Variable v_1 ist nicht nötig.) Mit der Formel $\varphi(v)$ schreiben wir die Existenz von drei Ecken auf dem Suchweg vor. Bei Shapegraphen, die keinen solchen Suchweg enthalten, wertet die Formel damit zwangsläufig für jede Ecke zu 0 aus. In diesem Fall entsteht die leere Struktur. Bei Shapegraphen mit einem Suchweg aus drei Ecken, wertet die Formel genau für die Wurzel zu 1 aus. Die induzierte Teilstruktur ist in diesem Fall immer eine einzelne Ecke. Es ist also durchaus möglich, mit der teilstrukturbasierten Partitionierungsmethode derartige Partitionen zu erzwingen. Die benötigten Teilstrukturdefinierer sind zumeist ausgesprochen unintuitiv, da sie der Idee der Methode zuwider stehen.

Wir kommen noch einmal auf die theoretische Mächtigkeit dieser Partitioniermethode zurück. Zu Beginn dieses Abschnittes haben wir die Eckenauswahl für die auszuzeichnende Teilstruktur schon unter diesem Blickwinkel untersucht. Wir haben dort festgestellt, dass für die Auszeichnung der Teilstruktur jede Teilmenge des durch die Analysespezifikation bestimmten Universums selektiert werden kann. Bei der Festlegung der Prädikate der Teilstruktur haben wir uns aus praktischen Erwägungen für eine Methode entschieden, die Einschränkungen beinhaltet. Diese Aussagen beziehen sich aber nur auf die Möglichkeiten, Teilstrukturen auszuzeichnen. Sie sagen über die möglichen Partitionen im Grunde wenig aus. Jede auf diese Weise erzeugbare Partition hat die Eigenschaft, das die Shapegraphen jeder Klasse in einer wie auch immer gearteten Teilstruktur übereinstimmen. Von der Methode her, ist es überhaupt nicht klar, dass sich jede Partition auf diese Weise erzeugen lässt. Es ist wahrscheinlicher, dass mit durch die Methode bedingten Einschränkungen zu rechnen ist. Allerdings ist die Angabe einer nicht erzeugbaren Partition nicht trivial, in den vorangegangenen Beispielen haben wir einige unintuitive Möglichkeiten kennengelernt. Wir wollen auf eine tiefere theoretische Untersuchung des Ansatzes in Bezug auf seine Mächtigkeit verzichten. Sie ist unserem Vorhaben wenig förderlich. Es ging uns bei der teilstrukturbasierte Partitioniermethode um Praktikabilität und nicht um ihre Mächtigkeit im Hinblick darauf, wieviele Partitionen

erzeugt werden können. Dafür steht die formelbasierte Methode bereit. Zudem entwickeln wir im folgenden Abschnitt ein Verfahren, beide Partitionierungsmethoden miteinander zu verschmelzen.

Das Ziel dieses Abschnittes war es, eine Menge von Shapegraphen zu partitionieren, sie in disjunkte Teilmengen zu zerlegen. Shapegraphen innerhalb einer Teilmenge sollen einander ähnlich sein. Dabei haben wir Shapegraphen als beschriftete Graphen interpretiert und diese graphentheoretische Sichtweise bei der Partitionierung betont. Ähnlichkeit haben wir als Ähnlichkeit „im Aussehen“, als Übereinstimmung in der Gestalt bestimmter (beschrifteter) Teilgraphen ausgedrückt. Die vorgestellte Methode spricht den visuellen Sinn und das visuelle Vorstellungsvermögen an. Auf diese Weise harmonisiert sie mit der Intention und Zielsetzung von Visualisierung. Im Vergleich zur formelbasierten Partitioniermethode ist sie ungleich praktischer und anwendungsfreundlicher.

6.4 Kombination von Partitionierungsmethoden

In den beiden vorangegangenen Abschnitten haben wir zwei Verfahren zur Partitionierung von Shapegraphenmengen vorgestellt. Beide Methoden haben ganz verschiedene Charakteristiken, dementsprechend haben beide ihre spezifischen Vorzüge und Unzulänglichkeiten. Eine Methode basiert auf der Auswertung von Formeln, die andere auf der Ähnlichkeit in der Gestalt von Teilstrukturen. Trotz dieses Unterschiedes bestehen strukturelle Ähnlichkeiten. In beiden Fällen haben wir die Ähnlichkeit auf eine Art und Weise definiert, dass sie sich mathematisch als Äquivalenzrelation darstellt. Die Äquivalenzrelationen induzieren jeweils eine Partition der Shapegraphenmenge. In diesem Abschnitt führen wir beide bisher nebeneinander stehenden Methoden zu einem einheitlichen Ganzen zusammen. In einem weiteren Schritt verallgemeinern wir diesen Prozess und beschreiben, wie sich weitere Ähnlichkeitsbegriffe integrieren lassen.

Das Zusammenführen kann auf vielerlei Arten geschehen, kreativen Naturen sind hier kaum Grenzen gesetzt. So viel Freiheit können wir uns nicht erlauben. Wir untersuchen, welche Forderungen wir stellen sollen. Die Ausgangssituation ist diese: Wir haben eine Menge $\mathcal{M}$ und zwei Äquivalenzrelationen $\sim_1$ und $\sim_2$ auf $\mathcal{M}$ gegeben. Wir können bei der Menge gerne an eine Menge von Shapegraphen und bei den Äquivalenzrelationen gerne an die formelbasierte und die teilstrukturbasierte Ähnlichkeit denken. Die folgende Überlegung ist aber unabhängig von der speziellen Art der Elemente und der Äquivalenzrelationen.

Die Ähnlichkeitsbegriffe, die wir zur Partitionierung verwenden, haben wir als Äquivalenzrelationen formuliert. Diese erzeugen uns Partitionen. In diesem strukturellen Rahmen wollen wir verbleiben. Das Resultat der Zusammenführung soll wieder eine

Äquivalenzrelation sein. Dies ist unsere erste Forderung. – Natürlich ist das noch nicht ausreichend. Für das Folgende bezeichnen wir die neue Relation mit $\sim$. Sie soll die ursprünglichen Äquivalenzrelationen respektieren: Wenn zwei Elemente bezüglich $\sim$ äquivalent sind, dann sollen sie es auch bezüglich der beiden ursprünglichen sein; wir fordern $a \sim b \Rightarrow (a \sim_1 b) \wedge (a \sim_2 b)$ für alle $a, b \in \mathcal{M}$. Das ist unsere zweite Forderung.

Betrachten wir für ein $x \in \mathcal{M}$ die durch x repräsentierte Klasse $[x]_\sim$, dazu sei $y \in [x]_\sim$. (Natürlich gilt $x \in [x]_\sim$, also ist $[x]_\sim$ nicht leer.) Die ursprünglichen Relationen sollen respektiert werden, es gilt folglich $y \sim_1 x$ und $y \sim_2 x$, was in Partitionsschreibweise $y \in [x]_{\sim_1}$ und $y \in [x]_{\sim_2}$, also $y \in ([x]_{\sim_1} \cap [x]_{\sim_2})$, bedeutet. Somit folgt $[x]_\sim \subseteq ([x]_{\sim_1} \cap [x]_{\sim_2})$. Prinzipiell sind zwei Fälle möglich: $[x]_\sim = ([x]_{\sim_1} \cap [x]_{\sim_2})$ oder $[x]_{\sim_1} \cap [x]_{\sim_2}$ zerfällt in mehrere Klassen, von denen $[x]_\sim$ eine ist. Für Letzteres gibt es keine Motivation, eine feinere Teilung ist durch $\sim_1$ und $\sim_2$ weder erklärbar noch begründbar. Daher verlangen wir außerdem, dass die Klassen, der durch $\sim$ induzierten Partition, größtmöglich seien. Das ist unsere dritte Forderung. Dann gilt für je zwei Elemente $x, y \in [x]_{\sim_1} \cap [x]_{\sim_2}$ auch $x \sim y$. Die Klassen der durch $\sim$ erzeugten Partition von $\mathcal{M}$ sind damit die nichtleeren Schnitte der Klassen von $\mathcal{M}/_{\sim_1}$ mit den Klassen von $\mathcal{M}/_{\sim_2}$.

Wir schreiten zur Formalisierung. Wir fassen die Spezifikatoren der formel- und der teilstrukturbasierten Ähnlichkeit zusammen. Der kombinierte Ähnlichkeitsbegriff basiert auf:

6.10 Definition. *Es sei F ein Formelpartitionierer und $(\varphi(v), P)$ ein Teilstrukturdefinierer. Wir nennen $\mathcal{D} = (F, \varphi(v), P)$ einen* Partitionsdefinierer.

Zur Erinnerung seien die konkreten Bedingungen an die Elemente eines Formelpartitionierers $(F, \varphi(v), P)$ noch einmal zusammengefasst: Die Menge F ist nach Definition 6.1 eine endliche Menge von geschlossenen Formeln. Nach Definition 6.6 ist $\varphi(v)$ eine Formel mit höchstens einer freien Variable v, des Weiteren ist P eine Teilmenge der Prädikate der Analysespezifikation. – Mittels dieses Spezifikators erklären wir, wann zwei Shapegraphen ähnlich sind:

6.11 Definition. *Es sei $\mathcal{D}$ ein Partitionsdefinierer. Zwei Shapegraphen S und T heißen* ähnlich bezüglich $\mathcal{D}$*, wenn $S \sim_F T$ und $S \sim_{(\varphi(v),P)} T$ gilt, wenn sie also ähnlich bezüglich F und ähnlich bezüglich $(\varphi(v), P)$ sind. Wir schreiben in diesem Fall $S \sim_\mathcal{D} T$.*

Wenn keine Unklarheiten oder Verwechselungen zu befürchten sind, verzichten wir wie gewöhnlich auf die explizite Nennung des Partitionsdefinierers und schreiben anstelle von $S \sim_\mathcal{D} T$ einfacher $S \sim T$. Um auszudrücken, dass zwei Shapegraphen nicht ähnlich sind, schreiben wir $S \not\sim T$.

Wir kommen an dieser Stelle, gewissermaßen als Einschub, noch einmal kurz auf das Thema Isomorphie zu sprechen. Wir hatten vereinbart, dass alle auftretenden

Shapegraphen paarweise nicht isomorph sind. Isomorphe Shapegraphen beschreiben dieselbe Heapsituation, verwenden dazu nur andere „Bezeichnungen“. Im Hinblick auf die Visualisierung wollen wir keine isomorphen Shapegraphen zulassen: Jede Heapsituation soll sinnigerweise nur einmal als Shapegraph beschrieben sein. Im Zuge der formalen Entwicklung der Theorie, fügen wir einen Zusammenhang zwischen Isomorphie und Ähnlichkeit an. Auf den Beweis können wir dank seiner Einfachheit verzichten. Es gilt:

6.12 Satz. *Isomorphe Shapegraphen sind ähnlich bezüglich jedes Partitionsdefinierers.*

Nach diesem kurzen Ausflug widmen wir uns wieder der Ähnlichkeitsrelation. Die Relationen $\sim_F$ und $\sim_{(\varphi(v),P)}$ sind Äquivalenzrelationen. Diese Eigenschaft überträgt sich auf die Relation $\sim_D$:

6.13 Satz. *Für einen Partitionsdefinierer $\mathcal{D}$ ist Ähnlichkeit bezüglich $\mathcal{D}$ eine Äquivalenzrelation.*

Der Beweis ist sehr einfach, die Äquivalenzeigenschaft folgt unmittelbar aus der Definition. Für einen Partitionsdefinierer $\mathcal{D} = (F, \varphi(v), P)$ und zwei Shapegraphen S und T ist $S \sim_{\mathcal{D}} T \Leftrightarrow (S \sim_F T) \wedge (S \sim_{(\varphi(v),P)} T)$. Dabei sind sowohl $\sim_F$ als auch $\sim_{(\varphi(v),P)}$ Äquivalenzrelationen.

Eine Äquivalenzrelation auf einer Menge induziert eine Partition dieser Menge. Für eine (endliche und nicht leere) Menge $\mathcal{M}$ von Shapegraphen bildet die Familie der Klassen $\mathcal{M}/_{\sim_{\mathcal{D}}} = \{[S] \colon S \in \mathcal{M}\}$ eine disjunkte Zerlegung von $\mathcal{M}$. Wenn der Bezug zum Partitionsdefinierer klar und keine Verwechselung zu befürchten ist, schreiben wir für die Partition kürzer $\mathcal{M}/_{\sim}$.

Damit haben wir den einzelnen Ähnlichkeitsbegriffen einen Rahmen gegeben, in den wir sie integrieren. Wir fassen die bisherigen Überlegungen zusammen:

6.14 Satz. *Es sei $\mathcal{M}$ eine endliche und nicht leere Menge von Shapegraphen, ferner sei $\mathcal{D}$ ein Partitionsdefinierer. Ähnlichkeit bezüglich $\mathcal{D}$ induziert als Äquivalenzrelation eine Partition $\mathcal{M}/_{\sim_{\mathcal{D}}}$ der Menge $\mathcal{M}$. Sie heiße die* durch $\mathcal{D}$ induzierte Partition. *Die Shapegraphen einer Klasse sind bezüglich $\mathcal{D}$ ähnlich.*

Bei der formel- und der teilstrukturbasierten Ähnlichkeit haben wir stets untersucht ob und wie die extremen Partitionen erzeugt werden können. Darunter verstehen wir in diesem Zusammenhang solche Partitionen, die nur aus einer einzigen Klasse bestehen, und solche, wo jede Klasse nur einen einzigen Shapegraphen enthält. Wir setzen diese Untersuchung hier fort. Wenn wir eine Partition mit genau einer Klasse erhalten wollen, dann müssen alle Shapegraphen einander ähnlich sein. Bei der formelbasierten Ähnlichkeit leistet $F = \emptyset$ das Gewünschte. Das Gleiche für das teilstrukturbasierte Ähnlichkeitskonzept liefert die Wahl von $\varphi(v) = 0$ und einer

Prädikatsmenge P, die keine nullstelligen Prädikate enthält. Dabei ist die Wahl von $P = \emptyset$ wohl am elegantesten. (In diesem Fall entsteht als Teilstruktur für jeden Shapegraphen die leere Struktur.) Bei der Wahl des Partitionsdefinierers $(\emptyset, 0, P)$, wobei die Teilmenge P der Analyseprädikate keine nullstelligen Prädikate enthalten darf, sind je zwei Shapegraphen einander ähnlich und die resultierende Partition besteht aus nur einer Menge. Das andere Extrem besteht darin, dass jede Klasse genau einen Shapegraphen enthält, dass keine zwei Shapegraphen einander ähnlich sind. Hier genügt es, wenn wir diese Bedingung bei einer der Ähnlichkeitsrelationen erzwingen, die andere kann beliebig sein. Die formelbasierte Methode erwies sich in dieser Hinsicht als nicht sehr praktikabel. Bei der teilstrukturbasierten können wir $\varphi(v) = 1$ setzen und für P die Menge aller Analyseprädikate wählen. (In diesem Fall stimmt jeder Shapegraph mit seiner reduzierten Teilstruktur überein.) Der Formelpartitionierer F kann dann beliebig sein.

Im Abschnitt 6.2 über die Partitionierung mittels Formeln haben wir die Mächtigkeit der Formelpartitioniermethode untersucht. In Satz 6.5 wurde gezeigt, dass mit ihr jede Partition einer Shapegraphenmenge erzeugt werden kann. Diese Aussage gilt auch für den kombinierten Ähnlichkeitsbegriff:

6.15 Satz. *Es sei $\mathcal{M}$ eine endliche und nicht leere Menge von Shapegraphen, weiter sei $\mathcal{P}$ eine Partition von $\mathcal{M}$. Dann gibt es einen Partitionsdefinierer $\mathcal{D}$, so dass die durch $\mathcal{D}$ induzierte Partition von $\mathcal{M}$ gleich $\mathcal{P}$ ist, so dass also $\mathcal{M}/_{\sim_\mathcal{D}} = \mathcal{P}$ gilt.*

Beweis. Wir verwenden die Bezeichnungen des Satzes. Es sei $\mathcal{P}$ eine Partition der Shapegraphenemenge $\mathcal{M}$. Dann existiert nach Satz 6.5 ein Formelpartitionierer F so, dass die durch ihn induzierte Partition gleich $\mathcal{P}$ ist. Wir betrachten den Teilstrukturdefinierer $(0, \emptyset)$. Der reduzierte Teilgraph jedes Shapegraphen (aus $\mathcal{M}$) ist der leere Graph, je zwei Shapegraphen (aus $\mathcal{M}$) sind ähnlich bezüglich der Teilstrukturähnlichkeit. Wir setzen $\mathcal{D} = (F, 0, \emptyset)$. Aus der Definition des Partitionsdefinierers folgt dann unmittelbar: Je zwei Shapegraphen (aus $\mathcal{M}$) sind genau dann bezüglich $\sim_F$ ähnlich, wenn sie bezüglich $\sim_\mathcal{D}$ ähnlich sind. □

Wir haben gesehen, wie sich die beiden in den vorangegangenen Abschnitten dargestellten Begriffe der Ähnlichkeit von Shapegraphen zu einem einheitlichen Ähnlichkeitsbegriff zusammenfassen lassen. Dieses Vorgehen kann problemlos auf weitere Ähnlichkeitsbegriffe erweitert werden. Als einzige Voraussetzung müssen wir fordern, dass sie als Äquivalenzrelationen formuliert werden können. Das ist keine gravierende Einschränkung. Normalerweise sind alle (binären) Relationen, die man umgangssprachlich mit Begriffen wie „gleich“, „äquivalent“ oder „ähnlich“ bezeichnet, Äquivalenzrelation.

Auf der Menge der Shapegraphen bezüglich der zugrunde liegenden Analysespezifikation seien die Äquivalenzrelationen $\sim_1, \sim_2, \ldots, \sim_n$, $n \geq 1$, gegeben. Wir definieren wie oben eine Relation $\sim$. Zwei Shapegraphen sind genau dann ähnlich bezüglich $\sim$ sein, wenn sie für jedes i, $1 \leq i \leq n$, ähnlich bezüglich $\sim_i$ sind. Damit ist $\sim$

eine Äquivalenzrelation. (Das ist sofort einsichtig, wenn wir die an Satz 6.13 angeschlossene Beweisskizze als Induktionsschluss einer vollständige Induktion lesen, der Induktionsanfang, $n = 1$, ist trivial.) Für jede Menge von Shapegraphen (bezüglich der zugrunde liegenden Analysespezifikation) induziert $\sim$ als Äquivalenzrelation eine Partition dieser Menge. Alle Shapegraphen in einer Klasse sind äquivalent, sie sind bezüglich jedes des durch $\sim_i$, $1 \leq i \leq n$, ausgedrückten Ähnlichkeitsbegriffs ähnlich. Jede Partition einer Menge lässt sich auf diese Weise erzeugen, Satz 6.15 gilt ebenfalls.

An dieser Stelle fügen wir eine Plausibilitätsbetrachtung ein. In den vorangegangenen beiden Abschnitten dieses Kapitels haben wir zwei Ähnlichkeitsbegriffe eingeführt: die formelbasierte Ähnlichkeit in Abschnitt 6.2 und die teilstrukturbasierte Ähnlichkeit in Abschnitt 6.3. In diesem Abschnitt haben wir beide Partitioniermethoden kombiniert. Dazu haben wir in Definition 6.10 den Partitionsdefinierer eingeführt. Alle drei Spezifikatoren wurden benutzt, um Äquivalenzrelationen zu erklären (Definitionen 6.2, 6.8 und 6.11). Diese Äquivalenzrelationen haben wir stets auf der Menge aller Shapegraphen bezüglich der vorgegebenen Analysespezifikation definiert. Die Visualisierung stützt sich auf die von der Shapeanalyse zu jedem Programmpunkt berechneten Shapegraphenmengen. Dabei handelt es sich jeweils um eine Teilmenge der Menge aller Shapegraphen (bezüglich der vorgegebenen Analysespezifikation), und es sind diese Teilmengen, die partitioniert werden. Wir untersuchen, wie die Partition einer Menge und die Partition einer ihrer Teilmengen zusammenhängen:

6.16 Satz. *Es sei $\mathcal{M}'$ eine endliche und nicht leere Mengen von Shapegraphen bezüglich der vorgegebenen Analysespezifikation und $\mathcal{D}$ ein Partitionsdefinierer. Die von $\mathcal{D}$ auf $\mathcal{M}'$ induzierte Partition sei mit $\mathcal{P}'$ bezeichnet. Für eine nicht leere Teilmenge $\mathcal{M} \subseteq \mathcal{M}'$ gilt für die von $\mathcal{D}$ induzierte Partition*

$$\mathcal{M}/_{\sim_\mathcal{D}} = \left\{\mathcal{K}' \cap \mathcal{M} \colon \mathcal{K}' \in \mathcal{P}' \wedge \mathcal{K}' \cap \mathcal{M} \neq \emptyset\right\}.$$

Beweis. Wir verwenden die Bezeichnungen des Satzes. Seine Voraussetzungen seien erfüllt. Wir betrachten einen beliebigen Shapegraphen $S \in \mathcal{M}$. In $\mathcal{M}/_{\sim_\mathcal{D}}$ sei $\mathcal{K}$ die Klasse, die S enthält. Sie besteht aus genau den Shapegraphen $T \in \mathcal{M}$ mit $T \sim_\mathcal{D} S$. In $\mathcal{P}'$ sei $\mathcal{K}'$ die Klasse, die S enthält. Es gilt $\mathcal{K} \subseteq \mathcal{K}'$. Für einen Shapegraph $T' \in \mathcal{K}' \setminus \mathcal{K}$ gilt ebenfalls $T' \sim_\mathcal{D} S$. Wegen $T' \notin \mathcal{K}$ folgt $T' \notin \mathcal{M}$. Damit gilt $\mathcal{K} = \mathcal{K}' \cap \mathcal{M}$. Diese Überlegung gilt für jedes $S \in \mathcal{M}$. Um den Beweis zu komplettieren, müssen wir nur noch anmerken, dass $\mathcal{M}/_{\sim_\mathcal{D}} = \{[S]_{\sim_\mathcal{D}} \colon S \in \mathcal{M}\}$ ist. □

In den vorangegangenen Abschnitten haben wir zwei Ähnlichkeitsbegriffe für Shapegraphen vorgestellt. Beide bedienten sich verschiedener Sichtweisen von logischen Strukturen, beiden Begriffe standen unverbunden nebeneinander. In diesem Abschnitt haben wir einen Rahmen geschaffen, in dem wir sie integriert und zusammengeführt haben. Es resultierte der Begriff des Partitionsdefinierers. Das Konzept ist erweiterbar, andere Ähnlichkeitsbegriffe können auf harmonische Weise hinzugefügt werden.

6.5 Verfeinerung von Partitionen

In den vorangegangenen Abschnitten dieses Kapitels haben wir die Partitionierung von Shapegraphenmengen behandelt. Dazu haben wir als zentralen Begriff den Partitionsdefinierer eingeführt. Seine Spezifikation beruht auf den zur Analyse verwendeten Prädikaten. Wir benutzen ihn, um eine gegebene Menge von Shapegraphen zu partitionieren. Unser Augenmerk galt dem Herstellen der Partition, dem Prozess der Partitionierung. Unterschiedliche Partitionen stehen bisher jedoch beziehungslos nebeneinander. Mit diesem Zustand wollen wir uns nicht zufrieden geben. Wir wollen Aspekte herausarbeiten, die beschreiben, wie sich Partitionen (einer Shapegraphenmenge) zueinander verhalten.

Vergegenwärtigen wir uns erneut die Motivation, die uns zum Partitionierungskonzept geführt hat. Als Eingabe für die Visualisierung verwenden wir die Ausgabe einer Shapeanalyse. Die zur Analyse verwendete Spezifikation weist üblicherweise einen höheren Grad an Detailliertheit auf. Wenn wir bestimmte Eigenschaften eines Algorithmus visualisieren wollen, dann müssen diese auch in der Analyse berücksichtigt werden, sie muss hinreichend viele Details berücksichtigen. Die erzeugte Ausgabe ist dann üblicherweise groß. Somit stehen wir vor der Notwendigkeit, diese Masse an Daten für die Visualisierung aufzubereiten. Die Art der Aufbereitung, um die es uns in diesem Kapitel methodisch geht, ist eine Strukturierung. Shapegraphen, die hinsichtlich zu spezifizierender Eigenschaften übereinstimmen, werden in Klassen zusammengefasst. Die Klassen stellen eine vergröberte, eine abstraktere, Sicht der auftretenden Heapsituationen dar.

Üblicherweise wird ein Benutzer innerhalb einer Visualisierungssitzung nicht dauernd denselben Grad an Abstraktheit verwenden wollen. Mal möchte er eine Übersicht und wird dafür eine grobe Sicht, eine Partitionierung mit wenigen Klassen, eine grobe Partitionierung, wünschen. Mal wird er einen bestimmten Aspekt detailliert untersuchen wollen und dafür eine feine(re) Sicht, eine feinere Partitionierung, hinsichtlich dieses Aspektes wünschen. Mal wird er sich einem anderen Aspekt zuwenden wollen, die dafür gewünschte Partitionierung wird ebenfalls eine feine Sicht bieten sollen, aber diesmal hinsichtlich des neuen Aspekts. Somit ist es angeraten, dass wir uns mit dem Thema Veränderung oder Ändern von Partitionen beschäftigen.

6.5.1 Das Konzept der Verfeinerung

In diesem Abschnitt diskutieren wir den Wechsel von einer Partition zu einer anderen. Unser Augenmerk liegt dabei sowohl auf dem, was bei den Partitionen gleich bleibt, als auch auf dem, was sich ändert. Wir untersuchen, wie die ursprüngliche und die veränderte Partition miteinander in Beziehung stehen. Da in unserer Anwendung

alle Partitionen durch Partitionsdefinierer erzeugt werden, weiten wir unsere Untersuchung auf die Partitionsdefinierer aus und klären, auf welche Art sich ein Wechsel der Partition in den Partitionsdefinierern widerspiegelt.

Zunächst einmal gibt es verschiedene Arten von Partitionswechseln. Wenn man von einer feinen Sicht, die einen bestimmten Aspekt detailliert widerspiegelt, zu einer anderen feinen Sicht, die einen anderen Aspekt detailliert zeigt, wechselt, dann werden die beiden Partitionen wahrscheinlich wenig miteinander zu tun haben. Diese Art von Wechsel werden wir hier nicht betrachten. Wir untersuchen den Wechsel von einer groben zu einer feinen Partition und umgekehrt. Wir sehen den hier beschriebenen Ansatz zur Algorithmenvisualisierung in einem Lernumfeld, Ziel der Visualisierung ist ein Zuwachs an Verständnis. Daher interessieren wir uns nur für Partitionswechsel, die diesem Ziel nicht zuwiderstehen. Haben die grobe und die feine Partition nichts oder nur wenig „miteinander zu tun", dann wird dies dem Visualisierungsziel wenig dienlich sein. Besser sind Wechsel, die möglichst viel von der Struktur der ursprünglichen Partition aufrechterhalten. Wir werden diese Art des Wechsels von Partitionen als Verfeinerung beziehungsweise Vergröberung bezeichnen.

Zunächst werden wir die beiden Begriffe für Partitionen definieren. Da Partitionen bei uns mittels Partitionsdefinierer entstehen, widmen wir uns anschließend der Frage, was der Begriff der Verfeinerung auf der Ebene der Partitionsdefinierer bedeutet. Wir beginnen mit:

6.17 Definition. *Es seien $\mathcal{P}$ und $\mathcal{P}'$ zwei Partitionen einer endlichen und nicht leeren Menge von Shapegraphen. Wir nennen $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$, wenn zu jeder Klasse $K' \in \mathcal{P}'$ eine Klasse $K \in \mathcal{P}$ mit $K' \subseteq K$ existiert. Ist $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$, dann nennen wir $\mathcal{P}$ alternativ eine* Vergröberung *von $\mathcal{P}'$.*

In dieser Definition haben wir uns darauf beschränkt, den Begriff der Verfeinerung für Partitionen von Shapegraphenmengen zu definieren. Es ist aber nicht wesentlich, dass die Elemente Shapegraphen sind, die definierende Bedingung nimmt keinen Bezug auf die Art der Elemente. Wir haben uns für diese Einschränkung entschieden, da wir den Begriff der Verfeinerung auf Partitionsdefinierer übertragen, die nur auf Shapegraphen anwendbar sind. – Mit der Festsetzung in obiger Definition ist jede Partition eine Verfeinerung von sich selbst. Manchmal ist es sinnvoll, eine etwas differenziertere Bezeichnung zu verwenden: Ist $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$ und gilt $\mathcal{P} \neq \mathcal{P}'$, dann sprechen wir von einer *echten Verfeinerung*. Unmittelbar aus der Definition erschließt sich das folgende Korollar: Ist $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$, dann gilt $|\mathcal{P}| \leq |\mathcal{P}'|$.

Wir untersuchen genauer, was die Bedingung in Definition 6.17 bedeutet. Wir verwenden die Bezeichnungen der Definition und setzen voraus, dass $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$ sei. Wir betrachten eine Klasse $K \in \mathcal{P}$ der gröberen Partition. Zu jedem Shapegraphen $S \in K$ gibt es eine Klasse $K' \in \mathcal{P}'$ der feineren Partition

mit $S \in K'$. Die Klasse K wird also vollständig durch Klassen von $\mathcal{P}'$ überdeckt. Da $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$ ist, gilt $K' \subseteq K$ für jede dieser Klassen. Das heißt aber nichts anderes, als dass K eine disjunkte Vereinigung von Klassen von $\mathcal{P}'$ ist. Die Umkehrung gilt ebenfalls, was direkt aus der Definition folgt. Damit haben wir gezeigt:

6.18 Satz. *Es seien $\mathcal{P}$ und $\mathcal{P}'$ zwei Partitionen einer endlichen und nicht leeren Menge von Shapegraphen. Die Partition $\mathcal{P}'$ ist genau dann eine Verfeinerung von $\mathcal{P}$, wenn für jede Klasse $K \in \mathcal{P}$ eine Familie $\mathcal{F}_K \subseteq \mathcal{P}'$ von Klassen von $\mathcal{P}'$ existiert, so dass $\mathcal{F}_K$ eine Partition von K ist.*

An dieser Stelle soll kurz eine strukturelle Eigenschaft von Verfeinerungen besprochen werden. Dazu sehen wir sie als binäre Relation auf der Menge aller Partitionen der Menge aller Shapegraphen (bezüglich der zugrunde liegenden Analysespezifikation) an. Im Anschluss an Definition 6.17 haben wir festgestellt, dass jede Partition eine Verfeinerung von sich selbst ist; die Verfeinerungsrelation ist also reflexiv. Zur Definition des Begriffs der Verfeinerung verwenden wir die Mengeninklusion, die eine symmetrische und transitive Relation ist. Beide Eigenschaften übertragen sich. Daher gilt: Die Verfeinerungsrelation ist eine Ordnungsrelation auf der Menge aller Partitionen einer Shapegraphenmenge.

Jetzt wechseln wir unseren Blickwinkel von Partitionen zu (den sie definierenden) Partitionsdefinierern. Wie stellt sich das Konzept der Verfeinerung und Vergröberung von Partitionen für Partitionsdefinierer dar? Wir untersuchen sie unter diesem Blickwinkel. Wir beginnen mit:

6.19 Definition. *Es sei $\mathcal{A}$ die Menge aller Shapegraphen bezüglich der vorgegebenen Analysespezifikation. Des Weiteren seien $\mathcal{D}$ und $\mathcal{D}'$ zwei Partitionsdefinierer. Wir nennen $\mathcal{D}'$ eine* Verfeinerung *von $\mathcal{D}$, wenn die durch $\sim_{\mathcal{D}'}$ induzierte Partition von $\mathcal{A}$ eine Verfeinerung der durch $\sim_{\mathcal{D}}$ induzierten Partition von $\mathcal{A}$ ist, wenn also $\mathcal{A}/_{\sim_{\mathcal{D}'}}$ eine Verfeinerung von $\mathcal{A}/_{\sim_{\mathcal{D}}}$ ist.*

Betrachten wir als ein einfaches Beispiel den Partitionsdefinierer $\mathcal{D} = (\emptyset, 0, \emptyset)$. Der Formelpartitionierer ist die leere Menge, je zwei Shapegraphen sind formelähnlich. Der Teilstrukturdefinierer wählt mit seiner Eckenauswahlformel $\varphi(v) = 0$ niemals eine Ecke aus und es werden wegen $P = \emptyset$ auch keine Prädikate in die Teilstruktur aufgenommen, für jeden Shapegraphen entsteht die leere Struktur als Teilstruktur. Damit sind je zwei Shapegraphen auch teilstrukturähnlich. Zusammengenommen sind also je zwei Shapegraphen bezüglich $\mathcal{D}$ ähnlich. Wenn wir eine (endliche und) nicht leere Menge $\mathcal{M}$ von Shapegraphen betrachten, dann induziert $\mathcal{D}$ eine Partition $\mathcal{P}$ von $\mathcal{M}$, die aus genau einer Klasse besteht. Jede Partition von $\mathcal{M}$ ist eine Verfeinerung der einklassigen Partition $\mathcal{P}$. Folglich ist jeder Partitionsdefinierer eine Verfeinerung von $\mathcal{D}$.

In Satz 6.15 haben wir gezeigt, dass es zu jeder Partition einer Menge von Shapegraphen einen Partitionsdefinierer gibt, der gerade diese Partition erzeugt. Wenn wir diesen Satz umformulieren, dann erhalten wir:

6.20 Satz. *Es sei $\mathcal{M}$ eine endliche und nicht leere Menge von Shapegraphen. Ferner sei $\mathcal{D}$ ein Partitionsdefinierer und $\mathcal{P} = \mathcal{M}/_{\sim_\mathcal{D}}$ die durch $\mathcal{D}$ induzierte Partition von $\mathcal{M}$. Zu jeder Verfeinerung $\mathcal{P}'$ von $\mathcal{P}$ existiert eine Verfeinerung $\mathcal{D}'$ von $\mathcal{D}$ mit $\mathcal{M}/_{\sim_\mathcal{D}} = \mathcal{P}'$.*

Definition 6.19 stützt sich bei der Festsetzung des Begriffs der Verfeinerung für Partitionsdefinierer auf eine Eigenschaft der induzierten Partitionen. Oft ist es hilfreich, sich stattdessen auf die von den Partitionsdefinierern induzierten Ähnlichkeitsrelationen zu beziehen. Es gilt:

6.21 Satz. *Ein Partitionsdefinierer $\mathcal{D}'$ ist genau dann eine Verfeinerung eines Partitionsdefinierers $\mathcal{D}$, wenn für je zwei Shapegraphen S und T (bezüglich der zugrunde liegenden Analysespezifikation) die Implikation $S \sim_{\mathcal{D}'} T \Rightarrow S \sim_\mathcal{D} T$ gilt.*

Beweis. Wir verwenden obige Bezeichnungen. Es gelte die Implikation und es sei S ein beliebiger Shapegraph. Des Weiteren sei $K' = [S]_{\sim_{\mathcal{D}'}}$ die Klasse der von $\mathcal{D}'$ induzierten Partition, die S enthält; entsprechend sei $K = [S]_{\sim_\mathcal{D}}$. Für jeden Shapegraph $T \in K'$ gilt $T \sim_{\mathcal{D}'} S$ und damit nach Voraussetzung $T \sim_\mathcal{D} S$, also $T \in K$. Dies zeigt $K' \subseteq K$. Nach Definition ist $\mathcal{D}'$ damit eine Verfeinerung von $\mathcal{D}$. Die Umkehrung ist offensichtlich: Für je zwei Shapegraphen $S, T \in K'$ gilt per Definition $S \sim_{\mathcal{D}'} T$, und aus der Inklusion $K' \subseteq K$ folgt $S \sim_\mathcal{D} T$. □

Zur Illustration betrachten wir ein Beispiel. Wir examinieren wieder die Suche nach einem Element in einem binären Suchbaum. Wir setzen voraus, dass wir uns an einem Programmpunkt am Anfang der Suchschleife befinden. Als Erstes betrachten wir den Partitionsdefinierer

$$\mathcal{D} = (\emptyset, \mathsf{root}(v), \{\mathsf{root}\}).$$

Der Formelpartitionierer ist die leere Menge, je zwei Shapegraphen sind bezüglich der leeren Menge ähnlich. Der Formelpartitionierer hat damit keinen Einfluss auf die Partitionierung, sie erfolgt ausschließlich auf der Basis gemeinsamer Teilstrukturen. Betrachten wir einen beliebigen Shapegraphen. Seine durch $\mathcal{D}$ definierte Teilstruktur besteht genau aus den Ecken (der Ecke), auf die der Zeiger root zeigt, für die dieses Prädikat wahr ist. Sie enthält auch nur das Prädikat root. Wenn wir alle Shapegraphen (bezüglich der verwendeten Analysespezifikation) durchlaufen und die durch $\mathcal{D}$ induzierten Teilstrukturen berechnen, dann erhalten wir zwei (mögliche) Graphen. Zum einen können wir den leeren Graphen erhalten. Er ist als $S^{(1)}_{red}$ in Abbildung 6.7 gezeigt. Zum anderen können wir den Graphen erhalten, der nur aus der Ecke besteht, auf die root zeigt, samt dem Prädikat root. Dies ist $S^{(2)}_{red}$ in Abbildung 6.7. In unseren Anwendungen tritt die leere Struktur normalerweise

nicht als induzierter Teilgraph auf, da wir als Ausgangsshapegraphen der Analyse normalerweise keine leeren Bäume verwenden. Die Shapegraphen in unseren Anwendungen haben also üblicherweise immer eine Wurzel. Damit besteht die durch $\mathcal{D}$ induzierte Partition üblicherweise aus genau einer Klasse. Im Hinblick auf die Strukturierung der Analyseausgabe ist diese Partition natürlich nicht sonderlich interessant, aber ihre Einfachheit prädestiniert sie als Ausgangspartition für dieses Beispiel.

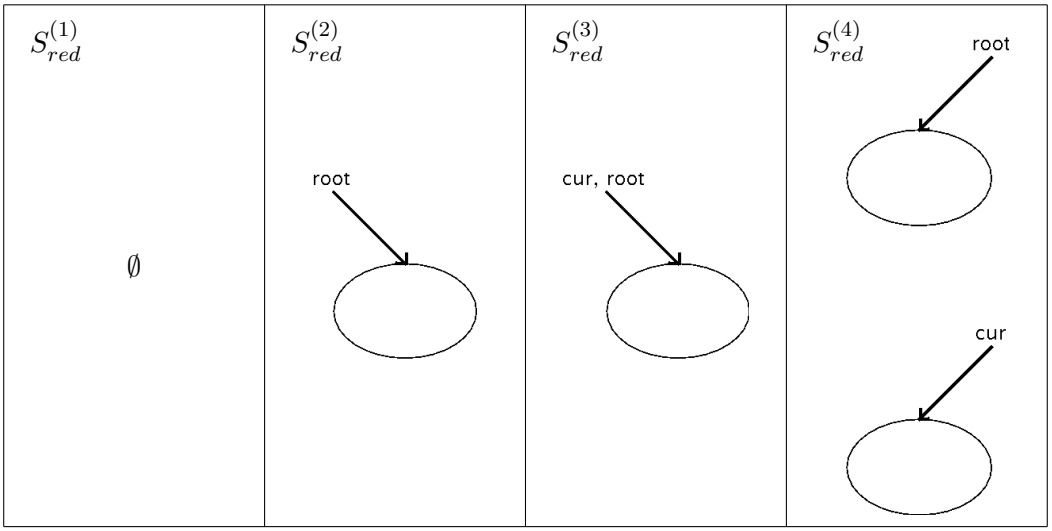

Abbildung 6.7: Reduzierte Teilstrukturen, die durch den Teilstrukturdefinierer $(\mathsf{root}(v) \vee \mathsf{cur}(v), \{\mathsf{root}, \mathsf{cur}\})$ induziert werden ($\emptyset$ bedeutet die leere Teilstruktur)

Als Zweites betrachten wir den Partitionsdefinierer

$$\mathcal{D}_1 = (\emptyset, \mathsf{root}(v) \vee \mathsf{cur}(v), \{\mathsf{root}, \mathsf{cur}\}).$$

Der Formelpartitionierer ist wieder die leere Menge. Die durch $\mathcal{D}_1$ induzierte Teilstruktur eines Shapegraphen besteht aus den Ecken, auf die die Zeiger root und cur zeigen, für die wenigstens eines diese Prädikate wahr ist. Sie enthält auch nur diese beiden Prädikate. Damit bestehen die folgenden vier Möglichkeiten, die in Abbildung 6.7 dargestellt sind:

$S_{red}^{(1)}$ Der Shapegraph enthält weder eine Ecken, auf die der Zeiger root zeigt, noch enthält er eine, auf die cur zeigt. Die induzierte Teilstruktur ist die leere Struktur.

$S_{red}^{(2)}$ Der Shapegraph enthält eine Ecke, auf die der Zeiger root zeigt, aber keine, auf die cur zeigt.

$S_{red}^{(3)}$ Der Shapegraph enthält eine Ecke, so dass die Zeiger root und cur beide auf sie zeigen.

$S_{red}^{(4)}$ Der Shapegraph enthält eine Ecken, auf die der Zeiger root zeigt, und eine davon verschiedene, auf die cur zeigt.

Wie eben bereits geschildert, tritt der Fall der leeren Teilstruktur in unseren Anwendungen üblicherweise nicht auf. Damit besteht die durch $\mathcal{D}_1$ induzierte Partition (üblicherweise) aus drei Klassen. In der durch $\mathcal{D}$ induzierten Partition waren alle Shapegraphen mit eventueller Ausnahme des leeren Graphen in einer Klasse enthalten. Diese Klasse ist in der durch $\mathcal{D}_1$ induzierten Partition in drei Klassen zerfallen. Damit ist gezeigt, dass es sich bei $\mathcal{D}_1$ um eine Verfeinerung von $\mathcal{D}$ handelt.

Als Drittes betrachten wir den Partitionsdefinierer. $\mathcal{D}_2 = (\{f_1, f_2, f_3\}, 0, \emptyset)$, wobei die im Formelpartitionierer auftretenden Formeln durch

$$\begin{aligned} f_1 &:= \exists v_1 \colon \mathsf{root}(v_1) \\ f_2 &:= \exists v_1, v_2 \colon \mathsf{root}(v_1) \wedge \mathsf{cur}(v_2) \\ f_3 &:= \exists v_1, v_2 \colon \mathsf{root}(v_1) \wedge \mathsf{cur}(v_2) \wedge v_1 \neq v_2 \end{aligned}$$

gegeben seien. Den Teilstrukturdefinierer $(0, \emptyset)$ haben wir schon besprochen. Für jeden Shapegraphen ist die durch ihn definierte Teilstruktur die leere Struktur. Je zwei Shapegraphen sind damit bezüglich $(0, \emptyset)$ teilstrukturähnlich. Die Partitionierung einer Shapegraphenmenge mittels $\mathcal{D}_2$ erfolgt ausschließlich auf der Basis des Formelpartitionierers. Wir untersuchen, aus welchen potentiellen Klassen die induzierte Partition besteht. Dabei argumentieren wir über die möglichen Wahrheitswerte der drei Funktionen für einen beliebigen Shapegraphen S. Wir notieren sie als Tupel $\big(W_S(f_1), W_S(f_2), W_S(f_3)\big)$. Zwei Beobachtungen helfen bei der Untersuchung:

1. Bei der von uns verwendeten Analysespezifikation und Shapeanalyse werten die Formeln f_1, f_2 und f_3 für jeden Shapegraphen zu einem definiten Wahrheitswert aus. Der Wahrheitswert $\frac{1}{2}$ tritt also (in den Wahrheitswertetupeln) nicht auf.
2. Die Formeln sind so aufgebaut, dass die Situation der Vorgängerformeln, der Formeln mit kleinerem Index, wiederhergestellt wird. Ist eine Formel wahr, dann müssen es auch die Vorgängerformeln sein. Wir verifizieren diesen Sachverhalt exemplarisch für f_2 und f_1, die anderen Fälle lassen sich analog zeigen. Gilt $W_S(f_2) = 1$ (für einen Shapegraphen S), dann existieren zwei nicht notwendigerweise verschiedene Ecken v_1 und v_2, so dass der Zeiger root auf v_1 und der Zeiger cur auf v_2 zeigt. (Mit dieser Ausdrucksweise meinen wir $W_{S,\beta[\frac{v}{v_1}]}\big(\mathsf{root}(v)\big) = 1$ und $W_{S,\beta[\frac{v}{v_2}]}\big(\mathsf{cur}(v)\big) = 1$.) Insbesondere gibt es dann die Ecke v_1, auf die root zeigt; damit gilt $W_S(f_1) = 1$. Einige Wahrheitswertetripel wie zum Beispiel $(1, 0, 1)$ sind deshalb nicht möglich, die entsprechenden „Klassen“ sind leer.

Wenn wir von allen möglichen Wahrheitswertetripeln diejenigen streichen, die nicht möglich sind, dann verbleibenden vier Tripel: $(0,0,0)$, $(1,0,0)$, $(1,1,0)$ und $(1,1,1)$. Diese vier potentiellen Klassen entsprechen den (potentiellen) Klassen, die durch $\mathcal{D}_2$ induziert werden, die Zuordnung ist gemäß der folgenden Tabelle gegeben.

f_1	f_2	f_3	reduzierter Teilgraph
0	0	0	$S^{(1)}_{red}$
1	0	0	$S^{(2)}_{red}$
1	1	0	$S^{(3)}_{red}$
1	1	1	$S^{(4)}_{red}$

Wir zeigen die Korrespondenz exemplarisch für einen Fall, der Nachweis der anderen Fälle kann analog erfolgen. Dazu sei eine beliebige Menge von Shapegraphen gegeben. Die Klasse in der durch $\mathcal{D}_2$ induzierten Partition, die dem Wahrheitswertetripel $(1,1,1)$ entspricht, sei K_2. Die Klasse in der durch $\mathcal{D}_1$ induzierten Partition, in der jeder Shapegraph als reduzierten Teilgraphen $S^{(4)}_{red}$ hat, sei K_1. Wir zeigen $K_1 = K_2$. Als Erstes sei $S \in K_1$. Wie auch immer S beschaffen ist, er enthält $S^{(4)}_{red}$ als Teilgraphen. Um die Formeln f_1, f_2 und f_3 auswerten zu können, genügt bereits diese Teilstruktur. Die Wahrheitswerte ergeben sich zu $(1,1,1)$, und somit ist $S \in K_2$. Als Zweites setzen wir $S \in K_2$ voraus, es gilt also $W_S(f_1) = W_S(f_2) = W_S(f_3) = 1$. Folglich existieren in S zwei verschiedene Ecken, so dass root auf die eine und cur auf die andere zeigt. In der Analyse sind beide Prädikate als „unique“ deklariert, es kann also jeweils höchstens eine Ecke geben, für die diese Prädikate zu wahr auswerten. Damit gibt es genau eine Ecke, auf die root zeigt, und genau eine, auf die cur zeigt. Wenn wir die durch $\mathcal{D}_1$ definierte Teilstruktur berechnen, dann können wir also den Rest des Graphen außer Acht lassen und es entsteht $S^{(4)}_{red}$, folglich ist $S \in K_2$. Damit haben wir gezeigt: Der Partitionsdefinierer $\mathcal{D}_2$ erzeugt (für jede Menge von Shapegraphen) dieselbe Partition wie $\mathcal{D}_1$. Insbesondere ist damit $\mathcal{D}_2$ eine Verfeinerung von $\mathcal{D}$.

Wir haben oben gezeigt, dass die Relation der Verfeinerung eine Ordnungsrelation ist. Dabei haben wir Verfeinerung als eine (binäre) Relation auf der Menge der Partitionen (der Menge aller Shapegraphen hinsichtlich der gegebenen Analysespezifikation) angesehen. Inzwischen haben wir den Begriff der Verfeinerung auf Partitionsdefinierer übertragen. Jetzt können wir ihn auch als eine (binäre) Relation auf der Menge aller Partitionsdefinierer (hinsichtlich der gewählten Analysespezifikation) ansehen. In diesem Verständnis ist die Verfeinerungsrelation aber keine Ordnungsrelation! Reflexivität und Transitivität gelten nach wie vor, diese Eigenschaften übertragen sich von den Partitionen auf die Partitionsdefinierer. Die Antisymmetrie ist aber verletzt, wie das vorstehende Beispiel zeigt: Die Partitionsdefinierer $\mathcal{D}_1$ und $\mathcal{D}_2$ induzieren dieselbe Partition, jeder Partitionsdefinierer ist eine Verfeinerung des anderen, aber es ist $\mathcal{D}_1 \neq \mathcal{D}_2$.

Während wir in den vorangegangenen Abschnitten die Erzeugung von Partitionen einer Shapegraphenmenge behandelt haben, ging es in diesem Abschnitt um den Wechsel, um den Übergang, von einer Partition zu einer anderen. Während einer Visualisierungssitzung wird ein Anwender den Grad der Abstraktheit ändern wollen: Mal möchte er eine grobe Sicht auf die möglichen Heapsituationen, mal möchte er einzelne Aspekte detailliert betrachten und dazu eine feine Sicht einstellen wollen. Ein solcher Wechsel zwischen grob und fein war Gegenstand der Diskussion. Wir haben dazu den Begriff der Verfeinerung (und Vergröberung) für Partitionen eingeführt und ihn anschließend auf Partitionsdefinierer übertragen.

6.5.2 Verfeinerungen bei der Visualisierung

Im vorangegangenen Teilabschnitt haben wir das Konzept der Verfeinerung diskutiert. Bei seiner Definition haben wir uns auf die induzierten Partitionen gestützt. Während einer Visualisierungssitzung erfolgt eine Veränderung von Partitionen durch das Manipulieren von Partitionsdefinierern. Daher interessieren Arten von Manipulationen, die im Zusammenhang mit Verfeinerungen und Vergröberungen auftreten und interessant sind. Insbesondere wäre es wünschenswert, einen solchen Typ von Manipulation auszeichnen zu können, am besten sogar auf syntaktischer Ebene. Diese Aspekte sind Themen, die in diesem Teilabschnitt behandelt werden.

Eigentlich bezieht sich der Begriff der Verfeinerung auf Paare von Partitionsdefinierern, so wurde er definiert. Aber jeder Partitionsdefinierer kann als Verfeinerung des Partitionsdefinierers $(\emptyset, 0, \emptyset)$ aufgefasst werden. Insofern besagt der Begriff der Verfeinerung auch etwas über Partitionsdefinierer selbst aus. Mathematisch-logische Bedingungen, die in Bezug auf Verfeinerungen auftreten, sagen damit auch etwas über den (strukturellen) Aufbau von Partitionsdefinierern aus. – In Satz 6.21 haben wir gezeigt, dass ein Partitionsdefinierer $\mathcal{D}'$ genau dann eine Verfeinerung eines Partitionsdefinierers $\mathcal{D}$ ist, wenn $S \sim_{\mathcal{D}'} T \Rightarrow S \sim_{\mathcal{D}} T$ gilt. Diese Bedingung beschreibt eine Eigenschaft der von den Partitionsdefinierern abgeleiteten Ähnlichkeitsrelationen, beziehungsweise sie stellt eine Forderungen an sie. Die Bedingung sagt aber wenig Konkretes über die (Struktur der) beteiligten Partitionsdefinierer aus. Die Implikation ist nach Definition äquivalent zu, die beiden Partitionsdefinierer seien durch $\mathcal{D} = (F, \varphi(v), P)$ und $\mathcal{D}' = (F', \varphi'(v), P')$ gegeben,

$$\big(S \sim_{F'} T \wedge S \sim_{(\varphi'(v),P')} T\big) \Rightarrow \big(S \sim_{F} T \wedge S \sim_{(\varphi(v),P)} T\big)\,.$$

Dies wiederum ist äquivalent zu den beiden Gleichungen

$$\big(\forall f' \in F' \colon W_S(f') = W_T(f')\big) \Rightarrow \big(\forall f \in F \colon W_S(f) = W_T(f)\big) \tag{6.1}$$

und

$$\big(S_{(\varphi'(v),P')} \simeq T_{(\varphi'(v),P')}\big) \Rightarrow \big(S_{(\varphi(v),P)} \simeq T_{(\varphi(v),P)}\big)\,. \tag{6.2}$$

Auch aus diesen Gleichungen lässt sich wenig Konkretes über die Struktur der beteiligten Partitionsdefinierer ablesen.

Bei dem Anliegen, Arten von Manipulationen von Partitionsdefinierern auszuzeichnen, die im Zusammenhang mit Verfeinerungen interessant sind, stehen drei Kriterien im Vordergrund, deren Verwirklichung anzustreben ist:

1. Die Art der Manipulation (der Partitionsdefinierer) soll mit der Art und Weise harmonisieren, wie ein Anwender während der Visualisierung vorgeht, um Partitionen zu verfeinern oder zu vergröbern. Wenn es einen typischen Modus gibt, dann sollen sich seine Charakteristiken in der auszuzeichnenden Art widerspiegeln.
2. Wenn wir ein Paar von Partitionsdefinierern $(\mathcal{D}, \mathcal{D}')$ vorliegen haben, wobei $\mathcal{D}'$ eine Verfeinerung von $\mathcal{D}$ ist und $\mathcal{D}'$ auf die auszuzeichnende Art und Weise aus $\mathcal{D}$ hervorgeht, dann möchten wir wenn möglich (syntaktische) Bedingungen formulieren, wie die Komponenten von $\mathcal{D}$ mit denen von $\mathcal{D}'$ zusammenhängen, also Bedingungen, die sich ohne Rückgriff auf die induzierten Ähnlichkeitsrelationen oder der induzierten Partitionen ausdrücken lassen. Auch wäre es vorteilhaft, eine Teilklasse aller Partitionsdefinierer auszeichnen zu können, die bezüglich Manipulationen der auszuzeichnenden Art abgeschlossen ist.
3. Die auszuzeichnende Art von Manipulation soll hinsichtlich ihrer Partitionierungsfähigkeit immer noch sehr mächtig sein. Wenn wir einen Partitionsdefinierer $\mathcal{D}$ mit induzierter Partition $\mathcal{P}$ betrachten, dann soll es zu jeder, oder zumindest zu fast jeder, Verfeinerung $\mathcal{P}'$ von $\mathcal{P}$ einen Partitionsdefinierer $\mathcal{D}'$ geben, der auf die auszuzeichnende Art aus $\mathcal{D}$ hervorgeht und $\mathcal{P}'$ induziert. Diese Forderung ist nicht nur im Kontext der theoretischen Entwicklung des Konzepts zu sehen: Die Verwendung von Manipulationen der auszuzeichnenden Art soll in der Praxis keine allzu großen Restriktionen mit sich bringen.

Die erste Forderung hat ihren Ursprung in der Anwendung des Verfeinerungskonzepts bei der Visualisierung. Daher ist es sie, die es zuerst zu konkretisieren gilt. Betrachten wir dazu den Fall, dass eine bestehende Partition verfeinert werden soll, und versetzen wir uns in die Situation eines Anwenders während einer Visualisierungssitzung. Beim Konzept der Verfeinerung von Partitionen wurde die Idee umgesetzt, dass eine verfeinerte Partition die Struktur der Ausgangspartition bewahren soll, dass die einzelnen Klassen weiter strukturiert sein sollen. In der Praxis besteht ein typisches Vorgehen darin, Spezifikationselemente zum Partitionsdefinierer hinzuzufügen. Dies geht einher mit der Erwartung, dass die Klassen der bestehenden Partition hinsichtlich des neuen Spezifikationselements weiter aufgeteilt werden, dass sie bezüglich des hinzugefügten Elements spezifischer werden. Jedenfalls erwarten wir keinen abrupten Wechsel der Partition. Diese Erwartungshaltung gilt auch für die durch den Teilstrukturdefinierer induzierten Teilstrukturen. Bei der Hinzunahme von Spezifikationselementen zum Teilstrukturdefinierer erwarten wir spezifischere Teilstrukturen, keinesfalls erwarten wir komplett verschiedene. In

graphentheoretischer Terminologie bedeutet diese Erwartungshaltung, dass für jeden Shapegraphen die durch den ursprünglichen Teilstrukturdefinierer induzierte Teilstruktur ein Teilgraph der durch den neuen Teilstrukturdefinierer induzierten Teilstruktur sein soll.

Formalisieren wir diese Vorstellung. Dazu sei $\mathcal{D} = (F, \varphi(v), P)$ ein Partitionsdefinierer. Wir beginnen die Diskussion mit dem Formelpartitionierer. Dazu setzen wir voraus, dass die Teilstrukturdefinierer gleich bleiben. Dann sind für jeden Shapegraphen die induzierten Teilstrukturen gleich und Gleichung (6.2) ist trivialerweise erfüllt. Wie lässt sich die intuitive Vorstellung vom Hinzufügen spezifizierender Elemente bei gleichzeitigem „Erhaltenbleiben“ der Struktur des Formelpartitionierers realisieren? Ein mit dieser Vorstellung konformer Ansatz besteht darin, den neuen Formelpartitionierer F' als Obermenge von F vorauszusetzen. Mit dieser zusätzlichen Forderung ist Gleichung (6.1) trivialerweise erfüllt: Wenn $W_S(f) = W_T(f)$ für alle Formeln $f \in F'$ gilt, hierbei seien S und T zwei beliebige Shapegraphen, dann gilt sie erst recht für jede Formel einer jeden Teilmenge von F', also insbesondere für jede Formel aus F. Dies zeigt, dass $\mathcal{D}' = (F', \varphi(v), P)$ eine Verfeinerung von $\mathcal{D}$ ist.

Aufgrund dieser Einsicht, lässt sich die dritte der aufgestellten Forderungen schon erfüllen. Es sei $\mathcal{D} = (F, \varphi(v), P)$ ein Partitionsdefinierer und $\mathcal{P}$ die von ihm erzeugte Partition einer beliebigen (endlichen und nicht leeren) Menge von Shapegraphen. Des Weiteren sei $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$. Nach Satz 6.5 existiert ein Formelpartitionierer F^*, dessen induzierte Partition gleich $\mathcal{P}'$ ist. Wir bilden den Partitionsdefinierer $\mathcal{D}' = (F \cup F^*, \varphi(v), P)$. Er ist von der oben betrachteten Art, und die von ihm induzierte Partition ist gleich $\mathcal{P}'$. Dazu müssen wir $S \sim_{F^*} T \Leftrightarrow S \sim_{\mathcal{D}'} T$ für je zwei (nicht notwendig verschiedene) Shapegraphen S und T zeigen. Als Erstes gelte $S \sim_{\mathcal{D}'} T$. Dann werten alle Formeln aus $F \cup F^*$ für beide Shapegraphen zum selben Wahrheitswert aus. Insbesondere werten damit alle Formeln aus F^* für S und T zum selben Wahrheitswert aus, was nach Definition 6.2 aber nichts anderes als $S \sim_{F^*} T$ bedeutet. Zum Nachweis der Rückrichtung gelte $S \sim_{F^*} T$. Alle Formeln aus F^* werten für S und T zum selben Wahrheitswert aus. Des Weiteren ist die durch $\sim_{F^*}$ induzierte Partition $\mathcal{P}'$ eine Verfeinerung von $\mathcal{P}$. Nach Satz 6.21 gilt somit ebenfalls $S \sim_{\mathcal{D}} T$. Das heißt, alle Formeln aus F werten für S und T zum selben Wahrheitswert aus und es gilt $S_{(\varphi(v),P)} \simeq T_{(\varphi(v),P)}$. Beides zusammen zeigt $S \sim_{\mathcal{D}'} T$.

Als Nächstes wenden wir uns dem Teilstrukturdefinierer zu. Er besteht aus einer Eckenauswahlformel und einer Teilmenge der Prädikate der Analysespezifikation. Wir diskutieren beide Bestandteile getrennt, wobei wir mit der Prädikatsmenge beginnen. Wie im Falle des Formelpartitionierers korrespondiert die Vorstellung vom Hinzufügen spezifizierender Elemente gut mit dem Vereinigungsoperator. Es sei $\mathcal{D} = (F, \varphi(v), P)$ ein Formelpartitionierer. Des Weiteren sei P' eine Menge von Prädikaten der Analysespezifikation mit $P \subseteq P'$ und $\mathcal{D}' = (F, \varphi(v), P')$. Die Formelpartitionierer sind identisch, damit ist Gleichung (6.1) trivialerweise erfüllt. Für jeden

Shapegraphen S bestehen die induzierten Teilstrukturen $S_{(\varphi(v),P)}$ und $S_{(\varphi(v),P')}$ aus denselben Ecken. Letztere besteht (gegebenenfalls) aus mehr Prädikaten als erstere. In der Sichtweise als beschrifteter Graphen ist $S_{(\varphi(v),P)}$ ein Teilgraph von $S_{(\varphi(v),P')}$. Sind die durch $\mathcal{D}'$ induzierten Teilstrukturen zweier Shapegraphen isomorph, dann sind es erst recht die durch $\mathcal{D}$ induzierten. Dies zeigt, dass Gleichung (6.2) erfüllt ist. Damit ist $\mathcal{D}'$ eine Verfeinerung von $\mathcal{D}$.

Als Letztes verbleibt die Eckenauswahlformel zu diskutieren. Die Vorstellung, dass die Hinzunahme spezifizierender Elemente zu einem spezifischeren Teilgraphen führen soll, korrespondiert im Falle der Eckenauswahlformel damit, dass in jedem Shapegraphen (gegebenenfalls) mehr Ecken ausgewählt werden. In den zuvor besprochenen Fällen, wo wir es mit Mengen zu tun hatten, bot der Vereinigungsoperator eine angemessene Formalisierung. Hier haben wir es mit einer logischen Formel zu tun und daher bietet sich der Disjunktionsoperator an. Es sei $\mathcal{D} = (F, \varphi(v), P)$ ein Partitionsdefinierer. Des Weiteres sei $\varphi'(v)$ eine Formel mit höchstens einer freien Variablen und $\mathcal{D}' = (F, \varphi(v) \vee \varphi'(v), P)$. Für jeden Shapegraphen S, sein Universum sei mit U_S bezeichnet, gilt

$$\left\{u \in U_S \colon \beta_{\left[\frac{v}{u}\right]}\big(\varphi(v)\big) = 1\right\} \subseteq \left\{u \in U_S \colon \beta_{\left[\frac{v}{u}\right]}\big(\varphi(v) \vee \varphi'(v)\big) = 1\right\}.$$

In graphentheoretischer Terminologie ist die induzierte Teilstruktur $S_{(\varphi(v),P)}$ ein Teilgraph von $S_{(\varphi(v) \vee \varphi'(v),P)}$. Dies hatten wir angestrebt. Bedauerlicherweise ist aber die durch $\mathcal{D}'$ induzierte Partition einer Shapegraphenmenge nicht notwendigerweise eine Verfeinerung der durch $\mathcal{D}$ induzierten.

Wir demonstrieren diesen Sachverhalt anhand eines Beispiels, wobei wir wieder den Suchalgorithmus für Suchbäume zugrunde legen und die Shapegraphenmenge $\mathcal{M}$ des Programmpunktes am Schleifeneingang betrachten. Als Formelpartitionierer wählen wir stets die leere Menge. Dann sind je zwei Shapegraphen formelähnlich und der Formelpartitionierer hat keinen Einfluss auf die Partitionierung. Außerdem wählen wir auch als Prädikatsmenge stets die leere Menge. Die Partitionierung erfolgt damit ausschließlich auf der Basis der durch die Eckenauswahlformel ausgezeichneten Ecken. Uns geht es in diesem Beispiel um die Unterscheidung zwischen strukturbedingten Abbruchfällen und Nichtabbruchfällen. Wir haben dieses Ansinnen sowohl bei der formelbasierten, vergleiche Seite 106, als auch bei der teilstrukturbasierten Partitionierung, vergleiche Seite 121, untersucht.

Wir betrachten zunächst die geschlossene Formel

$$f := \exists v_1, v_2 \colon v_1 \neq v_2 \wedge \mathsf{cur}(v_1) \wedge \mathsf{reach}[\mathsf{cur}](v_2).$$

Sie wertet für diejenigen Shapegraphen zu wahr aus, die eine Ecke besitzen, auf die der Zeiger cur zeigt, und bei denen diese Ecke kein Blatt des Baumes ist, wenn der Shapegraph also keinen strukturbedingten Abbruchfall darstellt. Beispielsweise wertet die Formel für den Shapegraphen S_{na} in Abbildung 6.8 zu 1 aus, für den Shapegraphen S_a dagegen zu 0.

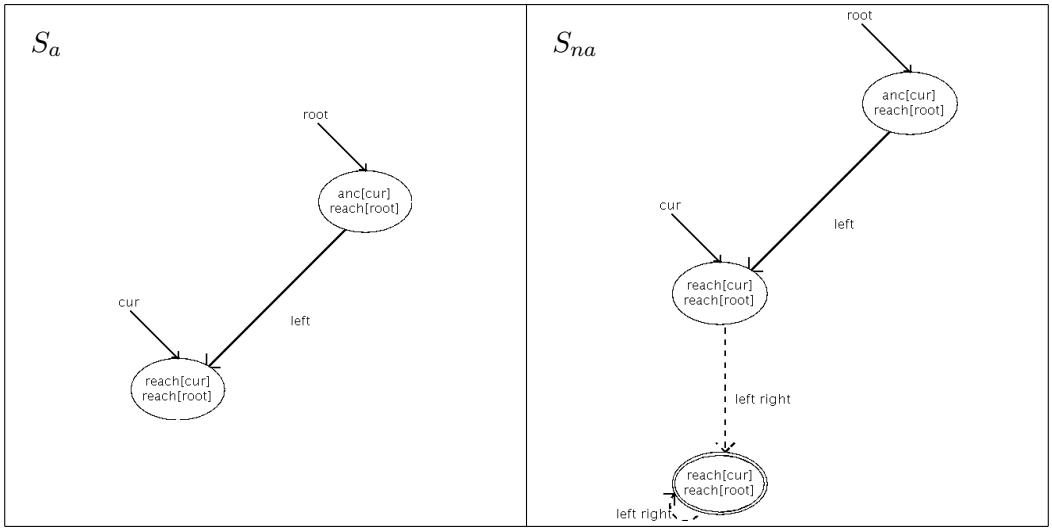

Abbildung 6.8: Strukturbedingte Abbruch- und Nichtabbruchfälle

Den Partitionsdefinierer, den wir als Erstes untersuchen, ist $\mathcal{D}_1 = (\emptyset, f \wedge \mathsf{root}(v), \emptyset)$. Für alle Shapegraphen, die einen strukturbedingten Abbruchfall repräsentieren, wertet die Eckenauswahlformel zu 0 aus und wegen $P = \emptyset$ entsteht in diesem Fall der leere Graph als Teilstruktur. Für Shapegraphen, die keinen strukturbedingten Abbruchfall darstellen, wertet sie genau für die Wurzel zu 1 aus, es entsteht der eineckige Graph als Teilstruktur. Insgesamt entstehen zwei Teilstrukturen und somit besteht die durch $\mathcal{D}_1$ induzierte Partition $\mathcal{M}/_{\sim_{\mathcal{D}_1}}$ aus zwei Klassen. – Als Zweites betrachten wir den Partitionsdefinierer $\mathcal{D}_2 = (\emptyset, \neg f \wedge \mathsf{root}(v), \emptyset)$. Er ist sehr ähnlich zu $\mathcal{D}_1$ und bewirkt auch sehr Ähnliches. Die Partition $\mathcal{M}/_{\sim_{\mathcal{D}_2}}$ besteht ebenfalls aus zwei Klassen: In einer sind die Shapegraphen gesammelt, die strukturbedingte Abbruchfälle repräsentieren und in der anderen Klasse die strukturbedingten Nichtabbruchfälle. Allerdings sind die durch $\mathcal{D}_2$ definierten Teilstrukturen nicht dieselben wie im Falle $\mathcal{D}_1$, sie sind vertauscht.

Beide Partitionsdefinierer induzieren dieselbe Partition, es gilt $\mathcal{M}/_{\sim_{\mathcal{D}_1}} = \mathcal{M}/_{\sim_{\mathcal{D}_2}}$. Insbesondere ist damit $\mathcal{D}_2$ eine Verfeinerung von $\mathcal{D}_1$. Wir betrachten zwei Shapegraphen S und T aus $\mathcal{M}$, die der Nichtabbruchklasse angehören. Für sie ist Gleichung (6.2) erfüllt: In beiden Partitionen sind S und T jeweils in einer gemeinsamen Klasse enthalten. Bezüglich $\mathcal{D}_1$ haben S und T den eineckigen Graphen als Teilstruktur und hinsichtlich $\mathcal{D}_2$ den leeren Graphen. Der eineckige Graph ist kein Teilgraph des leeren Graphen. Dies verdeutlicht, dass Gleichung (6.2) nicht automatisch einen strukturellen Zusammenhang zwischen den durch die Partitionsdefinierer induzierten Teilstrukturen impliziert.

Der aufmerksame Leser hat sicherlich bemerkt, dass wir in diesem Beispiel davon ausgegangen sind, dass die Formel f für die betrachteten Shapegraphen zu einem definiten Wahrheitswert auswertet. Diese Voraussetzung ist aber nicht notwendi-

gerweise erfüllt, bei (sehr) groben Analysen wird sie nicht gelten. Im Grunde genommen müssen wir verlangen, dass die Analyseausgabe diese Eigenschaft erfüllt. Dazu genügt es, wenn wir voraussetzen, dass cur und $\mathsf{reach}[\mathsf{cur}]$ Abstraktionsprädikate sind. Das ist bei der hier verwendeten Analysespezifikation der Fall, und es wird sogar üblicherweise für die im Hinblick auf Visualisierungen verwendeten Analysespezifikationen gelten. Deswegen haben wir diesen Punkt anfangs nicht explizit hervorgehoben.

Setzen wir das Beispiel fort. Die Eckenauswahlformel in $\mathcal{D}_1$ war $\varphi_1(v) := f \wedge \mathsf{root}(v)$ und in $\mathcal{D}_2$ war sie $\varphi_2(v) := \neg f \wedge \mathsf{root}(v)$. Als Drittes betrachten wir den Partitionsdefinierer $\mathcal{D}' = (\emptyset, \varphi'(v), \emptyset)$ mit der Eckenauswahlformel $\varphi'(v) := \varphi_1(v) \vee \varphi_2(v)$. Sie ist eine Disjunktion und entspricht damit dem gewählten Ansatz. Nun gilt – man beachte, dass f stets zu einem definiten Wahrheitswert auswertet –:

$$\begin{aligned}\varphi'(v) &= \varphi_1(v) \vee \varphi_2(v)\\ &= \big(f \wedge \mathsf{root}(v)\big) \vee \big(\neg f \wedge \mathsf{root}(v)\big)\\ &= (f \vee \neg f) \wedge \mathsf{root}(v)\\ &= \mathsf{root}(v).\end{aligned}$$

Wenn wir in diesem Beispiel wie üblich voraussetzen, dass der leere Shapegraph nicht auftritt, dann besitzt jeder Shapegraph wenigstens eine Wurzel. Für jeden Shapegraphen ist die durch $\mathcal{D}'$ definierte Teilstruktur der eineckige Graph ohne Prädikate. Je zwei Shapegraphen sind bezüglich $\sim_{\mathcal{D}'}$ ähnlich, die durch $\mathcal{D}'$ induzierte Partition besteht aus genau einer Klasse. Diese Partition ist definitiv keine Verfeinerung von den von $\mathcal{D}_1$ und $\mathcal{D}_2$ induzierten Partitionen, sie ist im Gegenteil eine Vergröberung. Die Tatsache, dass eine um eine Teilformel disjunktiv erweiterte Eckenauswahlformel für jeden Shapegraphen zu einer umfassenderen, spezifischeren Teilstruktur führt, impliziert nicht, dass es sich bei der induzierten Partition um eine Verfeinerung handelt.

Ein Grund dieses Phänomens liegt darin, dass Formeln große Freiheit bieten. Mit der Eckenauswahlformel können nahezu beliebige Eigenschaften spezifiziert werden. Auf diese Weise können zum Beispiel Gruppen von Shapegraphen bestimmte Teilgraphen „zugewiesen“ bekommen, so sind wir im Beispiel vorgegangen. Syntaktisch spiegelt sich diese Vielfalt an Möglichkeiten unter anderem im Erlaubtsein der All- und Existenzquantoren wider. Aber auch eine Beschränkung auf quantorenfreie Eckenauswahlformeln führte zu keinem Erfolg: Es kann ja in der Analysespezifikation ein nullstelliges Prädikat geben, dass genau die Eigenschaft beschreibt, die in der Eckenauswahlformel spezifiziert werden soll. Eine Beschränkung der Eckenauswahlformel auf eine Disjunktion von Konjunktionen unärer Prädikate bringt auch keine Abhilfe: Die Bedingung kann in der Analysespezifikation formal als einstelliges Prädikat formuliert worden sein und wir haben im vorangegangenen Beispiel gesehen, dass schon eine einzige Disjunktion das Problem aufwerfen kann. (Auch eine einzige Konjunktion kann zu dem Phänomen führen.)

In diesem Abschnitt behandeln wir Verfeinerungen, die bei der Visualisierung auftreten. Eine typische Art ist das Hinzufügen spezifizierender Elemente zum Partitionsdefinierer. Die Überlegungen dieses Abschnitts haben gezeigt, dass das Hinzufügen von (geschlossenen) Formeln zum Formelpartitionierer zu einer Verfeinerung führt. Ebenso führt die Hinzunahme von Prädikaten der Analysespezifikation zur Prädikatsmenge des Teilstrukturdefinierers zu einer Verfeinerung. Die Eckenauswahlformel ist jedoch problematisch. Wir lassen sie deshalb bei der syntaktischen Auszeichnung einer Art von Verfeinerung unverändert, das heißt wir behalten die Universen der jeweiligen induzieren Teilstrukturen bei. Wir setzen:

6.22 Definition. *Es sei $\mathcal{D} = (F, \varphi(v), P)$ ein Partitionsdefinierer, des Weiteren sei F' eine Menge von geschlossenen Formeln und P' eine Teilmenge der Prädikate der Analysespezifikation. Der Partitionsdefinierer $\mathcal{D}' = (F \cup F', \varphi(v), P \cup P')$ heiße eine* aggregierte Verfeinerung *von $\mathcal{D}$.*

In diesem Kapitel wurde ein parametrisches Ähnlichkeitskonzept für Shapegraphen entwickelt. Ein formelbasierter Ähnlichkeitsbegriff betont die Sichtweise eines Shapegraphen als logische Struktur. Ein teilstrukturbasierter Ähnlichkeitsbegriff betont die Sichtweise als Graph. Anschließend wurden beide Begriffe in einem einheitlichen Rahmen zusammengefasst. Das Ähnlichkeitskonzept erlaubt, die Analyseausgabe bezüglich des gewählten Ähnlichkeitsbegriffs zu strukturieren, zu abstrahieren. Während einer Visualisierungssitzung wird man üblicherweise nicht dauernd den gleichen Grad an Abstraktion verwenden wollen. Daher haben wir uns in diesem Abschnitt mit dem Verändern von Partitionen beschäftigt. Verfeinerungen und Vergröberungen von Partitionen wurden als zentrale Begriffe eingeführt. Im ersten Teilabschnitt wurde das Konzept entwickelt. Im zweiten Teilabschnitt wurden Verfeinerungen untersucht, die im Umfeld der Visualisierung auftreten. Während einer Visualisierungssitzung werden Verfeinerungen und Vergröberungen durch die Manipulation des Partitionsdefinierers erzeugt. Es wurde eine typische Art von Modifikation untersucht und syntaktisch beschrieben.

7 Ergänzende Betrachtungen zum Ähnlichkeitsbegriff

> Wie jedes gegen sich selbst einen Bezug hat, so muss es auch gegen andere ein Verhältnis haben.
>
> *(Johann Wolfgang Goethe)*

Die hier behandelten Themen schließen an die des letzten Kapitels an. Im ersten Abschnitt setzen wir die Betrachtungen zur Aufbereitung von Klassen zueinander ähnlicher Shapegraphen im Hinblick auf die Visualisierung fort. Die dort behandelten Methoden setzen auf der Ähnlichkeitsvorstellung des letzten Kapitels auf, sie folgen aber einer anderen Idee. Im zweiten Abschnitt gehen wir von Shapegraphen zu Sequenzen von Shapegraphen (abstrakten Zuständen) über. Wir betrachten Kantenzüge im Transitionsgraphen und übertragen den Ähnlichkeitsbegriff auf sie. Auch führen wir einen reduzierten Transitionsgraphen ein. Bei der Navigation im Transitionsgraphen, beim Voranschreiten im Kontrollflussgraphen, muss man vom aktuellen Shapegraphen (abstrakten Zustand) zu einem geeigneten Nachfolgegraphen übergehen. Die Inhalte des dritten Abschnittes bietet hier Hilfestellung. Wir entwickeln zwei Charakterisierungen von Shapegraphen, welche die Nachfolgerauswahl unterstützen. Die erste stützt sich auf den Ähnlichkeitsbegriff, während sich die zweite davon methodisch unterscheidet.

7.1 Visualisierung von Klassen ähnlicher Shapegraphen

7.1.1 Zur Extension des Begriffs Visualisierung

Dieser Abschnitt behandelt eine weitere Aufbereitung der Analyseausgabe. Im Grunde genommen handelt es sich nicht nur um eine weitere Aufbereitung, sondern um eine weitere Form der Aufbereitung. Sie wird nicht direkt auf die Analyseausgabe angewandt, vielmehr setzt sie auf dem in Kapitel 6 beschriebenen Ähnlichkeitskonzept auf. Es handelt sich also um eine Aufbereitung der Klassen einer (von einem Partitionsdefinierer) induzierten Partition im Hinblick auf die Visualisierung. In der linearen Sequenz, die mit der Shapeanalyse beginnt und mit der eigentlichen Darstellung von (Shape-)Graphen (beispielsweise am Bildschirm) endet, ist das hier zu

Besprechende zwischen der Strukturierung und der eigentlichen Visualisierung, der eigentlichen Darstellung lokalisiert.

Um auf das Thema, worum es in diesem Abschnitt geht, hinzuführen, soll an dieser Stelle diskutiert werden, was alles im Kontext dieser Arbeit unter dem Begriff Visualisierung verstanden werden kann. Dabei klären wir auch, was wir alles mit dem Begriff umfassen, welche Extension wir ihm zuerkennen. Auf die Frage, was allgemein unter Visualisierung zu verstehen sei, herrscht im Grundsätzlichen Einigkeit. Jedoch unterscheiden sich die Antworten zum Teil deutlich darin, wie weitgefasst der Begriff verstanden wird. So wird in [Duden 2007] eine sehr enge Sicht angegeben: Visualisieren wird als „optisch darstellen, veranschaulichen“ verstanden und der Werbesprache zugeordnet. Demgegenüber beginnt das Lemma zu Visualisierung in [Brockhaus 2006] sehr allgemein mit der Charakterisierung als „Bezeichnung für bildliche Formulierung und Kommunikation, das heißt für Aufbereitung von Information mit vor allem bildlichen Mitteln wie auch für visuelle Wahrnehmung“. Mit einem solch weitgefassten Verständnis kann man zum Beispiel auch das Drehbuch eines Films als Visualisierung ansehen. Wir beschäftigen uns weder mit Werbung noch mit Filmen, sondern mit Algorithmenvisualisierung auf der Basis von Shapeanalyse. Was soll Visualisierung für uns, in unserem Rahmen bedeuten, wo soll sie einsetzen, was soll sie umfassen?

Wenn wir den Begriff Visualisieren (sehr) eng fassen, dann würden wir darunter lediglich die Darstellung der Graphen (Shapegraphen, Transitionsgraph) am Bildschirm verstehen. Visualisierung bestünde dann aus den mit dieser Darstellung zusammenhängenden Aspekten. Wir können grob drei Gruppen von Aspekten unterscheiden. Zum Ersten sind es Aspekte, die zur konkreten visuellen Repräsentation, zur Darstellung, zum „Zeichnen“ der Graphen gehören. Hiermit ist das Layout gemeint. Es umfasst die Gestaltung der Seite mit dem Graphen, sein „Ausbreiten“ und das Schema für die Anordnung seiner Teile. Konkret beinhaltet es die Festlegung der Eckenpositionen, das Kantenlayout und der Beschriftungen. Auch abstrahiertere Formen des Layouts gehören hierher. Man kann beispielsweise die „Zeichendimensionen“ (Koordinatenachsen, Farbe, Form von Ecken und dergleichen) mit Prädikaten verknüpfen, so dass ein Voranschreiten in der Dimension eine semantische Bedeutung bezüglich des Prädikats hat. Verknüpft man beispielsweise eine Koordinatenachse mit dem Prädikat dle (data less or equal), dann drückt die Achse eine Sortierungsinformation aus. Zum Zweiten handelt es sich um Aspekte, welche die Darstellung einer ganzen Sequenz von Graphen betreffen, die das Nacheinander der Shapegraphen, das Voranschreiten im Programmfluss (in Animation) umsetzen. Hierzu gehört die graphische Transformation eines (gezeichneten) Graphen in einen Nachfolgegraphen wie sie beispielsweise in [Bieber 2001] mittels „Inbetweening graphs“ realisiert wurde. Zum Dritten handelt es sich um Aspekte, welche die Bedienung des Visualisierungsprogramms, die Interaktion (Interaktionsmöglichkeiten) des Programms mit dem Anwender betreffen. Ein Benutzer sollte auf viele Faktoren der Visualisierung sinnvoll Einfluss nehmen können. Zu diesen Aspekten gehören zum Beispiel die Navigation durch die Shapeanalyseausgabe oder die Einstellung unter-

schiedlich abstrakter Sichten auf die Analyseausgabe. Wenn wir Visualisierung im Folgenden in diesem engen Sinn meinen, dann sprechen wir auch von *eigentlicher Visualisierung*.

Das andere Extrem ist eine weitgefasste Vorstellung des Begriffs Visualisierung. Wir würden dann alle in dieser Arbeit behandelten Themen als Teile der Visualisierung ansehen. Wenn man den Begriff noch weiter fassen möchte, dann könnte man sogar die Shapeanalyse selbst als Teil der Visualisierung ansehen. Schließlich berechnet sie (für einen Algorithmus) zu jeden Programmpunkt eine als Graph auffassbare Beschreibung der an diesem Programmpunkt möglichen konkreten Heapsituationen. Das entspricht der Vorstellung, dass ein Drehbuch, es handelt sich um eine Vorstufe eines Films, schon Visualisierung ist. In dieser Analogie könnte man die in dieser Arbeit behandelten Methoden zur Aufbereitung der Shapeanalyseausgabe als dramaturgische Umgestaltung und Straffung der Drehbuchvorlage verstehen, die wichtigen Wesenszüge der Beteiligten und die bedeutungstragenden Aktionen sollen dem Betrachter deutlicher vor Augen geführt werden.

Wir wollen die Trennung, was wir im Rahmen dieser Betrachtungen normalerweise mit dem Begriff Visualisierung bezeichnen und was nicht, in der Mitte dieser beiden Extreme vornehmen. Die Shapeanalyse ist für unseren Ansatz zur Algorithmenvisualisierung nur insofern relevant, als uns ihre Ausgabe als Eingabe dient. Sie ist der Startpunkt unser Bemühungen, aber kein integraler Bestandteil. Auch werden wir das in Kapitel 6 entwickelte Ähnlichkeitskonzept mit seiner Klasseneinteilung noch nicht als Visualisierung interpretieren. Demgegenüber werden wir die Art der Aufbereitung, die das Thema dieses Abschnitts bildet, dazuzählen.

Doch wo ist der Unterschied zwischen der Klasseneinteilung, der Form der Aufbereitung, die wir auch als Strukturierung bezeichnen, und der Form der Aufbereitung, um die es hier gehen wird? Die Shapeanalyse liefert für jeden Progammpunkt eine Menge von Shapegraphen; die Klasseneinteilung gibt dieser eine zusätzliche Struktur, die es erlaubt, Ähnliches von nicht Ähnlichem zu unterscheiden. Das, was Shapeanalyse ausmacht, bleibt erhalten: die Lesart der Shapegraphen, ihr Abstraktionsniveau, ihr Informationsgehalt, der Informationsgehalt des Ganzen. Diese Art der Umgestaltung der Shapeanalyseausgabe hat durchaus das Ziel, der (eigentlichen) Visualisierung (der konkreten Darstellung) einen höheren Wert in Bezug auf Lernvorgänge und deren Effektivität zu geben. Da sie aber „Shapeanalyse invariant lässt“ (beziehungsweise lassen soll), ist ihr Blick primär immer auf die Shapeanalyse zurückgewandt.

Die Aufbereitung, um die es in diesem Abschnitt gehen soll, wendet den rückwärtsgewandten Blick von der Shapeanalyse auf die eigentliche Visualisierung. Unser Fokus ist also nach vorne gerichtet. Wir begnügen uns nicht mehr nur mit einer Strukturierung der Analyseausgabe wie beispielsweise der Einprägung einer zusätzlichen Struktur. Wir fassen Aufbereitung weiter, durchaus als eine Form von (leichter) Umgestaltung. Dabei lösen wir uns (ein wenig) von dem Fundament der Shapeanalyse. Wir verlangen nicht mehr, dass das Resultat der Umgestaltung Shapegraphen sein

müssen. Unser Fokus ist darauf ausgerichtet, die in der Analyseausgabe codierte Information durch die Visualisierung effektiver kommunizieren zu können. Der Unterschied zwischen der Strukturierung, zu der die Klasseneinteilung gehört, und der in diesem Abschnitt betrachteten Form der Aufbereitung ist also in der Blickrichtung begründet. Wir sprechen von einer Methode als einer Visualisierungsmethode, wenn sie primär auf die eigentliche Visualisierung und nicht auf die Shapeanalyseausgabe ausgerichtet ist.

Nachdem wir nun die Trennlinie zwischen dem, was wir Visualisierung nennen wollen und was nicht, allgemein gezogen haben, benennen wir nun die Gegenstände, um die es hier gehen soll, konkret. Um welche Art von Aufbereitung geht es hier? Die im Folgenden vorgestellten Methoden sind zwischen Strukturierung (Klasseneinteilung) und der eigentlichen Visualisierung angeordnet. Ihre Eingaben sind damit Klassen einer induzierten Partition. (Es wird sich herausstellen, dass einige der im nächsten Teilabschnitt vorgestellten Methoden auch auf beliebige Shapegraphenmengen angewandt werden können, doch gehen wir grundsätzlich von Klassen aus.) Unsere bisher einzige Methode zur Aufbereitung der Shapeanalyseausgabe war das Klassenkonzept; letztendlich wurde also jede Klasse direkt visualisiert. Die Anzahl der Graphen in den Klassen und deren Größe bestimmen die Komplexität, sowohl die der Visualisierung selbst, als auch die der visuellen Repräsentation (was sich der Anwender betrachtet). Wir bezeichnen dies als *Visualisierungskomplexität*. Häufig wird man einen Graphen pro Bildschirmseite präsentieren. Die Anzahl der Graphen stellt also einen eigenständigen Aspekt der Visualisierungskomplexität dar, wir nennen ihn die *primäre Visualisierungskomplexität*. Die Größe eines Graphen wirkt sich auf die (subjektive) Erfassbarkeit der im Graphen codierten Information aus, wir nennen sie die *sekundäre Visualisierungskomplexität*.

Gegenstand dieses Abschnittes sind Methoden, die Visualisierungskomplexität einer Klasse zu senken. Trivialerweise kann man natürlich jede Klasse zur leeren Menge reduzieren, was ein Höchstmaß an Reduktion der Visualisierungskomplexität liefert. Allerdings verbleibt nach dieser Reduktion überhaupt keine Information mehr über die Klasse, beispielsweise darüber, was sie charakterisiert. Uns interessieren nur Methoden, die eine brauchbare Repräsentation der Klasse erlauben, ihre charakterisierenden und charakteristischen Merkmale müssen erhalten bleiben. Des Weiteren soll sich diese Information leichter als vorher herauslesen lassen, dieser Punkt besitzt eine deutlich subjektive Färbung.

Unser Visualisierungskonzept ist modular aufgebaut: Ein (erstes) Modul ist für die Strukturierung der Shapeanalyseausgabe zuständig. Die Ausgabe wird an das Klassenvisualisierungsmodul übergeben, bei der eine (oder mehrere) Methoden angewandt werden. Deren Ausgabe wird an das Modul übergeben, was die konkrete Präsentation realisiert. Jedes Modul kann dabei um Methoden erweitert werden. Für das Zusammensetzen der Methoden und Module ist wichtig, dass alle Ein- und Ausgaben derselben Schnittstellenspezifikation genügen. Die Methoden zur Klassenvisualisierung sind als offene Liste zu verstehen. Bei Bedarf können sie problemlos

um weitere Methoden erweitert werden.

Konkret behandeln wir in diesem Abschnitt fünf Methoden zur Klassenvisualisierung. Drei von ihnen (simultane Visualisierung sowie Einbettungs- und V-Ecken-Visualisierung) wurden in [Johannes u. a. 2005] vorgestellt. Im nächsten Teilabschnitt präsentieren wir die verschiedenen Methoden. Durch die Beschreibung wird auch die Berechnungsvorschrift definiert. Anschließend gehen wir im folgenden Teilabschnitt von der deskriptiven Darstellung zu einer vergleichenden und evaluierenden Betrachtung über. Wir stellen Vor- und Nachteile der einzelnen Methoden dar. Des Weiteren leiten wir daraus bevorzugte Anwendungssituationen und Anwendungsmöglichkeiten der einzelnen Methoden ab.

7.1.2 Methoden der Klassenvisualisierung

Als Eingabe der Klassenvisualisierungsmethoden haben wir Klassen einer induzierten Partition im Sinn. Es wird sich herausstellen, dass einige Methoden auch auf beliebige Shapegraphenmengen angewandt werden können. Aus diesem Grund bezeichnen wir die Eingabe formal häufig einfach nur als Shapegraphenmenge. Die Shapegraphen einer Klasse einer durch einen Partitionsdefinierer induzierten Partition zeichnen sich in zweierlei Hinsicht aus. Wenn wir die Auswertung des Formelpartitionierers für einen Shapegraphen als Wahrheitswertetupel ansehen, dann werten alle Shapegraphen der Klasse zum selben Tupel aus, während alle Shapegraphen des Programmpunktes außerhalb der Klasse zu anderen Tupeln auswerten. Zweitens haben alle Shapegraphen der Klasse die gleiche induzierte Teilstruktur und die reduzierten Teilgraphen aller Shapegraphen des Programmpunktes außerhalb der Klasse haben eine davon verschiedene. Wir beschränken unsere Aufmerksamkeit auf die Shapegraphen und lassen den Formelpartitionierer in der folgenden Diskussion unberücksichtigt.

Jede der im Folgenden beschriebenen Methoden lässt sich formal als eine Funktion auffassen. Jede Methode erhält als Eingabe eine endliche und nichtleere Menge von Shapegraphen. Die Forderung nach Endlichkeit ist zwingend, da die Funktion effektiv, also auch in endlicher Zeit, berechenbar sein muss. Die Forderung nach der Verschiedenheit von der leeren Menge dient in erster Linie der Bequemlichkeit, da alle in dieser Arbeit vorgestellten Strukturierungsmethoden der TVLA-Ausgabe nichtleere Mengen liefern. Als Definitionsbereich der Funktion können wir die Potenzmenge der Menge aller möglichen Shapegraphen bezüglich der gewählten Analysespezifikation oder gar die Potenzmenge der Menge aller überhaupt möglichen Shapegraphen annehmen. Das Bild der Funktion ist eine Menge von gerichteten und mit Prädikaten und Prädikatswerten der Analysespezifikation beschrifteten Graphen. Es soll an dieser Stelle nicht darüber diskutiert werden, ob nicht nahezu jeder solche Graph als Shapegraph interpretiert werden kann. Der entscheidende Punkt ist, dass die Graphen in der Ausgabemenge nicht notwendigerweise Shapegraphen von der Art sein brauchen, wie sie üblicherweise von TVLA erzeugt werden. Bei einer der Methode

werden wir es mit Ausgabegraphen zu tun, die bei TVLA nicht auftreten können. Um dies zu betonen sprechen wir deshalb anstelle von Shapegraphen allgemeiner von Graphen oder von *Visualisierungsgraphen*.

Es bezeichne $\mathcal{M}$ eine endliche und nichtleere Menge von Shapegraphen, die als Eingabe, als Argument der berechnenden Funktion, diene. Die von den einzelnen Methoden berechneten Ausgabemengen $\mathcal{N}$ können in ihrer Art recht verschieden sein. Einige der Methoden berechnen eine Teilmenge $\mathcal{N} \subseteq \mathcal{M}$ als Ausgabe. Die Shapegraphen in $\mathcal{N}$ sind dabei auf die eine oder andere Weise charakteristisch für $\mathcal{M}$. Worin diese Charakteristik besteht, hängt konkret von der jeweiligen Methode ab. Die Diskussion im vorangegangenen Absatz hat schon anklingen lassen, dass wir nicht immer so vorgehen. Einige Methoden berechnen eine Menge von Shapegraphen beziehungsweise Graphen, die keine Teilmenge der Eingabe ist, aber eine praktikable und für die Visualisierung geeignete Charakterisierung von $\mathcal{M}$ gestattet.

Simultane Visualisierung

Es sei $\mathcal{M}$ eine (endliche und nichtleere) Menge von Shapegraphen. Die diese Methode beschreibende Funktion f ist die Identität, es gilt also $f(\mathcal{M}) = \mathcal{M}$. Sie ist eine der denkbar einfachsten und damit auch zeiteffizientesten Methoden. Dass konstante Funktionen im Hinblick auf die Reduzierung der Visualisierungskomplexität keinen Nutzen bieten, ist einsichtig. Die Benennung „simultan“ soll nicht so sehr den temporalen Aspekt der „Gleichzeitigkeit“ betonen als vielmehr, dass alle Shapegraphen in $\mathcal{M}$ „zusammen“ und „gemeinsam“ die graphische Präsentation von $\mathcal{M}$ darstellen. Ob diese zum selben Zeitpunkt oder beispielsweise nacheinander präsentiert werden, soll dabei nicht von Bedeutung sein.

Einbettungsvisualisierung

Als Ausgangssituation zur Beschreibung dieser Methode sei eine Menge $\mathcal{M}$ von Shapegraphen gegeben, bei der wir der Einfachheit wieder davon ausgehen, dass sie endlich und nichtleer sei. Die hier zu entwickelnde Methode lässt sich als eine Funktion auf der Potenzmenge aller möglichen Shapegraphen auffassen, die eine (endliche und nichtleere) Shapegraphenmenge $\mathcal{M}$ auf eine ihrer Teilmengen $\mathcal{N} \subseteq \mathcal{M}$ abbildet. Sinnigerweise soll mit $\mathcal{M} \neq \emptyset$ auch $\mathcal{N} \neq \emptyset$ gelten. Die Methode der Einbettungsvisualisierung folgt damit der Idee, einige Shapegraphen aus $\mathcal{M}$ auszuwählen und die Übrigen unberücksichtigt zu lassen. Die ausgewählte Teilmenge $\mathcal{N}$ enthält Shapegraphen, die in einer näher zu kennzeichnenden Weise als repräsentativ für $\mathcal{M}$ angesehen werden können.

Ein Shapegraph ist eine formale und üblicherweise auch abstrahierte Beschreibung von konkreten Heapsituationen. Sie erfolgt hinsichtlich der Analysespezifikation,

also insbesondere hinsichtlich der dort vereinbarten logischen Prädikate. Jede konkrete Heapsituation können wir mit einem „konkreten Shapegraphen“ identifizieren: Seine Individuen sind die einzelnen Heapzellen, und er enthält alle Prädikate der Analysespezifikation, die für diese konkrete Heapinstanz ausgewertet werden. Die Shapeanalyse benutzt abstrahierte Shapegraphen. Die Abstrahierung ist in erster Linie ein Mittel, um die Terminierung der Analyse zu garantieren. Üblicherweise gehen dabei Informationen über die Datenwerte und über die Anzahl von Heapzellen verloren. Daher beschreibt ein Shapegraph in der Regel unendlich viele konkrete Heapsituationen.

Es ein S ein Shapegraph bezüglich der zugrunde liegenden Analysespezifikation. Wir bezeichnen mit $k(S)$ die Menge aller konkreten Shapegraphen, also letztendlich aller konkreten Heapsituationen, die durch S beschrieben werden. Wir gehen dabei davon aus, dass alle diese konkreten Shapegraphen ein kanonisches Universum besitzen, so dass wir sie einfacher miteinander in Beziehung setzen können. Für zwei Shapegraphen S und T kann natürlich $k(S) \cap k(T) = \emptyset$ gelten. Wenn der Schnitt nichtleer ist, dann sind die folgenden drei Fälle möglich:

- $k(S) \subseteq k(T)$
- $k(T) \subseteq k(S)$
- $k(S) \cup k(T)$ liegt weder vollständig in $k(S)$ noch in $k(T)$

Uns interessiert der Fall der Inklusion, das sind die ersten beiden Möglichkeiten. Wir beschränken uns in der Betrachtung auf den ersten Fall, der andere ist hierzu symmetrisch. Es gelte also $k(S) \subseteq k(T)$. Das bedeutet nichts anderes, als dass alle von S beschriebenen (konkreten) Heapsituationen auch von T beschrieben werden. Gegebenenfalls, wenn die Inklusion echt ist, beschreibt T aber noch zusätzliche. Wegen $|k(S)| \leq |k(T)|$ kann S als „restriktiver“ gedeutet werden. Im Vergleich zu T können wir den Shapegraphen S als „spezifischer“ oder „konkreter“ ansehen. Demgegenüber können wir T als „umfassender“ oder „allgemeiner“ als S bezeichnen.

Betrachten wir zur Verdeutlichung als Beispiel die beiden Shapegraphen S_{or} und T der ersten Zeile in Abbildung 7.1. Da sie bei der Analyse von Algorithmen für binäre Bäume auftreten, ist der durch die Zeigervariablen und -komponenten induzierte Graph ein binärer Baum, so dass wir anstelle von Heapsituation einfacher und deutlicher von Baum sprechen. Der Shapegraph T beschreibt Bäume, bei denen es eine Wurzel root und eine davon verschiedene Ecke cur gibt, die ein Blatt des Baumes ist; der Weg von root zu cur besteht aus wenigstens drei Ecken und alle (den Zeigerkomponenten entsprechenden) Kanten auf diesem Weg sind mit right beschriftet. An den inneren Ecken des Weges können noch Teilbäume hängen, und von diesen ist wenigstens einer vorhanden. Der Shapegraph S_{or} ist ähnlich: Der einzige Unterschied besteht darin, dass der Weg von der Wurzel root zum aktuellen Element cur aus genau drei Ecken besteht. Wird ein Baum, eine konkrete Heapsituation von

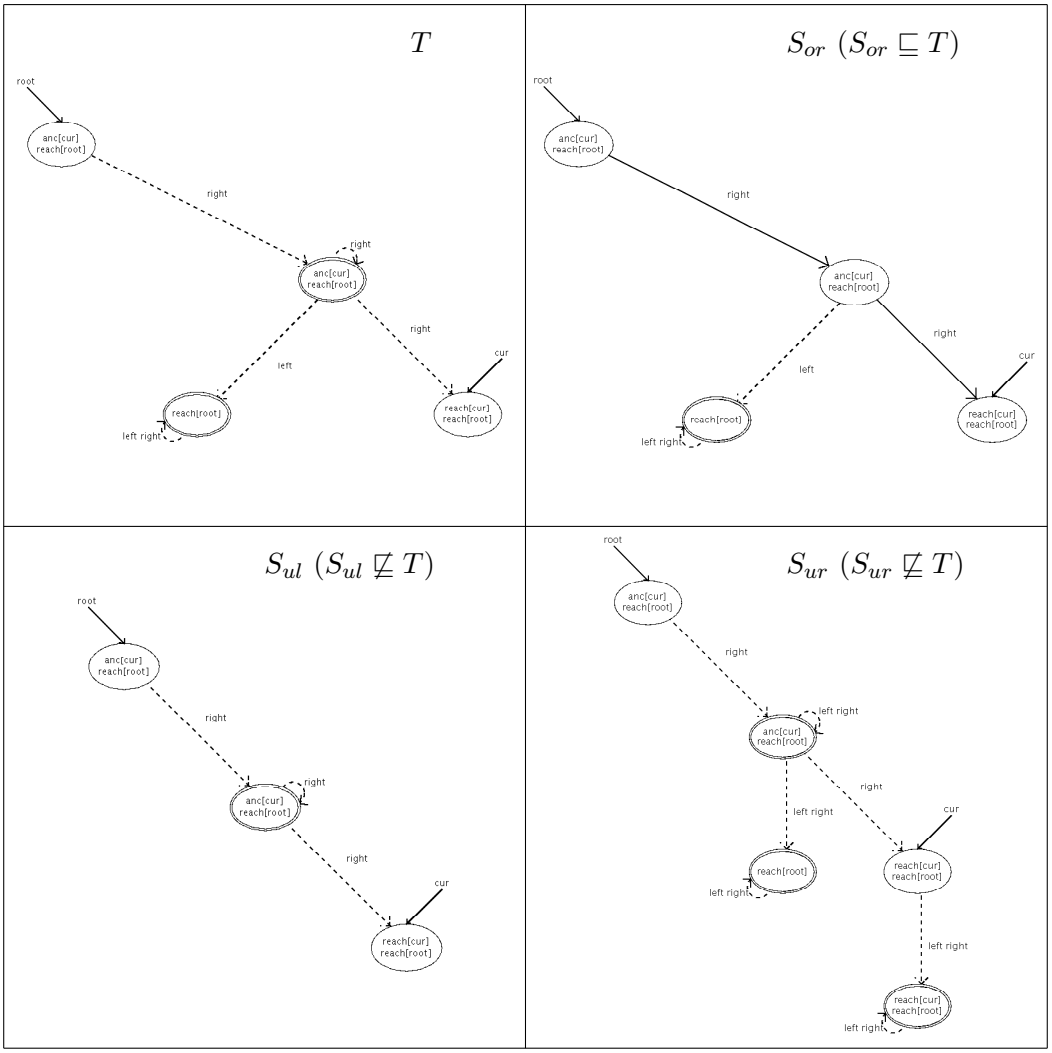

Abbildung 7.1: Einbettung von Shapegraphen

S_{or} beschrieben, dann wird sie damit auch zwangsweise von T beschrieben. Folglich gilt $k(S_{or}) \subseteq k(T)$.

Diese Tatsache wollen wir für unsere Visualisierung nutzen. Wenn wir im Falle $k(S) \subseteq k(T)$) anstelle von S und T nur den Shapegraphen T visualisieren, wenn wir also etwa S aus $\mathcal{M}$ entfernen, dann erhalten wir immer noch eine korrekte, konservative Beschreibung der durch $\mathcal{M}$ repräsentierten Heapsituationen. Die neue Beschreibung ist höchstens gröber geworden. Wir können nun, wann immer für zwei Shapegraphen $S \in \mathcal{M}$ und $T \in \mathcal{M}$ die Bedingung $k(S) \subseteq k(T)$ gilt, den Shapegraphen S aus $\mathcal{M}$ entfernen. Auf diese Weise bleiben am Ende die „allgemeinsten“ Shapegraphen von $\mathcal{M}$ übrig. Die Methode der Einbettungsvisualisierung bestimmt die Ausgabemenge $\mathcal{N} \subseteq \mathcal{M}$, indem sie aus der Shapegraphenmenge $\mathcal{M}$ diese „allge-

meinsten“ Shapegraphen auswählt.

Nach diesen einleitenden Betrachtungen schreiten wir zur formalen Explikation. Die eben skizzierte Berechnungsvorschrift können wir so nicht umsetzen: Für einen Shapegraphen S die Menge $k(S)$ aller konkreten Heapsituationen zu berechnen, ist, wenn nicht unmöglich, so doch wenigstens extrem ineffizient. Für zwei Shapegraphen S und T müssen wir die Gültigkeit von $k(S) \subseteq k(T)$ direkt aus S und T bestimmen können. Hierzu bedienen wir uns des Konzepts der *Einbettung*, wie es in Abschnitt 3.6, Definition 3.2 (Seite 47), vergleiche gegebenenfalls auch [Sagiv u. a. 2002], beschrieben ist. Formal stellt sich die Einbettung als eine binäre Relation auf der Menge aller Shapegraphen (bezüglich der zugrunde liegenden Analysespezifikation) dar. Zur Charakterisierung von Eigenschaften wird auf der Menge der (dreiwertigen) Wahrheitswerte die Informationsordnung $\sqsubseteq$ herangezogen.

In der Definition der Einbettung wird die Surjektivität der die Einbettung vermittelnden Funktion verlangt. Hierauf können (und wollen) wir nicht verzichten. Diesen Sachverhalt demonstrieren wir gleich anhand eines Beispiels, auch werden wir noch Beispiele zur Verdeutlichung betrachten. Doch zuvor diskutieren wir einige Aspekte, die uns weitere Einsicht vermitteln. – Für jeden Shapegraphen S gilt $S \sqsubseteq S$, die identische Abbildung auf dem Universum von S leistet das Geforderte. Die Einbettungsrelation ist also reflexiv. Gilt $S_1 \sqsubseteq S_2$ und $S_2 \sqsubseteq S_3$, wobei die die Einbettung vermittelnden Funktionen f_{12} und f_{23} seien, dann zeigt die Konkatenation $f_{23} \circ f_{12}$, dass auch $S_1 \sqsubseteq S_3$ gilt. Die Einbettungsrelation ist transitiv.

Der Shapegraph S sei in den Shapegraphen T einbettbar. Dann gibt es also eine die Einbettung vermittelnde Surjektion f vom Universum von S auf das Universum von T, die die in der Definition angegebenen Bedingungen erfüllt. Die in dieser Arbeit verwendeten und skizzierten Analysespezifikationen sind so beschaffen, dass in allen von TVLA berechneten Shapegraphen alle Abstraktionsprädikate p_a stets definite Wahrheitswerte besitzen. (In [Manevich 2003], Kapitel 4.2, werden solche Shapegraphen definit genannt; dort findet sich auch ein alternativer Beweis für die folgende Überlegung.) In diesem Fall reduziert sich die Bedingung $p_a^S(u) \sqsubseteq p_a^T\big(f(u)\big)$ aus der Definition zu $p_a^S(u) = p_a^T\big(f(u)\big)$. Mit anderen Worten bedeutet dies, dass f jedes Abstraktionsprädikat respektiert, sie bildet ein Individuum auf ein Individuum mit gleichem kanonischen Namen ab. Nun sind die von TVLA berechneten Shapegraphen (im Normalfall) so beschaffen, dass es in jedem Shapegraphen zu jedem kanonischen Namen höchstens eine Ecke mit diesem kanonischen Namen gibt. Die Surjektion f ist dann folglich auch injektiv, insgesamt also eine Bijektion.

Nach dieser Betrachtung kehren wir zu der Forderung nach Surjektivität in der Definition der Einbettung zurück. Wir untersuchen, was geschehen würde, wenn wir auf sie verzichten würden. Hierzu betrachten wir in Abbildung 7.1 die beiden Shapegraphen T und S_{ul} der linken Spalte. Wir wollen versuchen, S_{ul} in T einzubetten. Die Funktion, die die Einbettung vermitteln soll, sei mit f bezeichnet. Die in der Definition geforderten Bedingungen an die Prädikate gilt auch für die Abstraktionsprädikate. In beiden Shapegraphen sind alle Wahrheitswerte der Abstraktionsprädi-

kate definit. Mithin muss die die Einbettung vermittelnde Funktion sie respektieren, wodurch sie eindeutig bestimmt ist. Korrespondierende Ecken auf dem Suchweg von der Wurzel root zur aktuellen Ecken cur werden aufeinander abgebildet. Die nicht auf dem Suchweg liegende Summary-Ecke (in T) tritt dann nicht als Bild auf. Die von T beschriebenen Heapsituationen haben wir oben bereits erklärt. Für dieses Beispiel ist es wichtig zu bemerken, dass von den inneren Ecken des Suchweges von root nach cur wenigstens ein Teilbaum (zur linken Seite) herabhängt. Das bedeutet insbesondere, dass es in allen durch T beschriebenen Heapsituationen wenigstens ein Heapelement gibt, das nur von root (aber nicht von cur) erreichbar ist und das kein Ahne des aktuellen Elementes (oder der Wurzel) ist. Bei allen Heapsituationen, die durch S_{ul} beschrieben werden, gibt es keine solche Heapzelle. In diesem Sinne ist jede von S_{ul} beschriebene Heapsituation von jeder von T beschriebenen verschieden. Es gilt $k(T) \cap k(S_{ul}) = \emptyset$. Das widerspräche unserer eingangs beschriebenen Intention. Die Forderung nach Surjektivität ist also notwendig.

Die angekündigte Vorstellung von Beispielen zur Einbettung steht noch aus. Dazu betrachten wir die Shapegraphen in Abbildung 7.1. Wir hatten uns schon überlegt, dass die Inklusion $k(S_{or}) \subseteq k(T)$ besteht. Jetzt überlegen wir uns, dass auch $k(S_{or}) \sqsubseteq k(T)$ gilt. Die die Einbettung vermittelnde Funktion muss eine Bijektion auf den Universen sein, die kanonische Namen invariant lässt. Die Zuordnung ist intuitiv und eindeutig. Es lässt sich leicht verifizieren, dass die in Definition 3.2 geforderten Bedingungen an die Prädikatswerte dann erfüllt sind, entweder stimmen korrespondierende Werte überein oder der Wert in T ist $\frac{1}{2}$. Für S_{ul} hatten wir schon gesehen, dass er sich nicht in T einbetten lässt. Auch S_{ur} kann nicht in T eingebettet werden. Zum einen stimmt die Eckenzahl in S_{ur} und T nicht überein, weswegen es keine Bijektion zwischen ihren Universen geben kann. Zum anderen haben die Prädikate left und right in S_{ur} für einige Argumente[1] den Wahrheitswert $\frac{1}{2}$, während sie in T für die korrespondierenden Argumente den Wert 0 haben.

Wir betrachten erneut die Situation $S \sqsubseteq T$, wobei die Funktion f die Einbettung vermittele. Wie bisher habe jedes Abstraktionsprädikat in S und T nur definite Wahrheitswerte. (Und natürlich gibt es in jedem Shapegraphen zu jedem kanonischen Namen höchstens ein Individuum mit diesem Namen.) Des Weiteren gelte $T \sqsubseteq S$, die zugehörige Funktion sei g. Sowohl f als auch g sind Bijektionen, die jeweils kanonische Namen invariant lassen. Daher sind $f \circ g$ und $g \circ f$ Identitäten. Diese Tatsache hat Auswirkungen auf die Wahrheitswerte der Prädikate. Wir demonstrieren dies anhand eines einstelligen Prädikates. Hierzu bezeichne u ein beliebiges Individuum des Universums von S. Wenn beide Einbettungen hintereinander ausgeführt werden, so ergibt sich $p^S(u) \sqsubseteq p^T\big(f(u)\big) \sqsubseteq p^S\big(g(f(u))\big) = p^S(u)$, woraus $p^S(u) = p^T\big(f(u)\big)$ folgt. Die Bijektion f respektiert also jedes Prädikat der Analysespezifikation. Gleiches gilt auch für g. Satz 8.4 zeigt, dass f und g deshalb Isomorphismen sind, die Shapegraphen S und T sind isomorph. Die von TVLA

1. Man betrachte die Summary-Ecke, die das Innere des Suchweges repräsentiert, oder diese und die Summary-Ecke, welche die am Suchweg herabhängenden Teilbäume repräsentiert.

berechnete Ausgabe ist derart, dass zu jedem Programmpunkt des analysierten Algorithmus eine Menge nichtisomorpher Shapegraphen berechnet wird. In diesem Fall sind S und T gleich. Folglich ist die Einbettungsrelation dann antisymmetrisch. Wir fassen zusammen:

7.1 Satz. *Die Einbettungsrelation ist reflexiv und transitiv (auf jeder Menge von Shapegraphen). Ist $\mathcal{M}$ eine Menge von paarweise nichtisomorphen Shapegraphen, so dass für jeden Shapegraphen alle Abstraktionsprädikate definite Wahrheitswerte haben, so ist die Einbettungsrelation antisymmetrisch auf $\mathcal{M}$; sie definiert eine (partielle) Ordnung auf $\mathcal{M}$.*

In der einleitenden Motivation hatten wir die Einbettungsvisualisierung dadurch charakterisiert, dass sie aus einer Menge $\mathcal{M}$ von Shapegraphen die „allgemeinsten“ auswählt. Wann immer für zwei Shapegraphen S und T aus $\mathcal{M}$ die Menge der von S beschriebenen (konkreten) Heapsituationen eine Teilmenge der von T beschriebenen ist, wenn also $k(S) \subseteq k(T)$ gilt, so entfernen wir S aus $\mathcal{M}$. Die verbleibenden Shapegraphen bilden die Ausgabe. Diese Enthaltenseinrelation auf der Ebene der konkreten Heapsituationen lässt sich auf der Ebene der (dreiwertigen) Shapegraphen adäquat durch den Begriff der Einbettung ausdrücken. Wann immer $S \sqsubseteq T$ für zwei Shapegraphen S und T aus $\mathcal{M}$ gilt, entfernen wir S aus $\mathcal{M}$. Es verbleiben genau die bezüglich $\sqsubseteq$ maximalen Elemente[2].

7.2 Definition. *Es sei $\mathcal{M}$ eine Teilmenge der Menge der Shapegraphen eines Programmpunktes. Die Ausgabe der* Einbettungsvisualisierung *sei die Menge der bezüglich der Einbettungsrelation $\sqsubseteq$ maximalen Elemente von $\mathcal{M}$.*

Nach der formalen Spezifikation der Methode demonstrieren wir sie zur Verdeutlichung an einem Beispiel. Wir betrachten wieder die Suche nach einem Element in einem binären Suchbaum, wobei wir unsere Standardspezifikation verwenden. Wir befinden uns am Programmpunkt n2, das ist der Schleifeneingangspunkt. Da es in dieser Arbeit um die Aufbereitung der Shapeanalyseausgabe für die Visualisierung geht, strukturieren wir die Shapegraphenmenge zum Programmpunkt n2. Wir partitionieren sie mithilfe eines Partitionsdefinierers, und $\mathcal{M}$ sei eine der Klassen. Der Einfachheit halber setzen wir den Formelpartitionierer $F = \emptyset$, so dass er keine Auswirkung bei der Partitionierung hat. Den Teilstrukturdefinierer wählen wir so, dass die induzierten Teilstrukturen jeweils die Suchwege von der Wurzel zur aktuellen Ecke sind, vergleiche Abschnitt 6.3. Die Klassen repräsentieren also gerade die verschiedenen Strukturen des Suchweges. Für dieses Beispiel betrachten wir diejenige Klasse, bei der die reduzierte Teilstruktur aller Shapegraphen aus $\mathcal{M}$ der Graph in Abbildung 7.2 ist.

2. Es sei $\mathcal{M}$ eine Menge und $\sqsubseteq$ eine Ordnungsrelation auf $\mathcal{M}$. Ein Element $S_m \in \mathcal{M}$ heiße *maximales Element* von $\mathcal{M}$, wenn $\forall S \in \mathcal{M}\colon S_m \sqsubseteq S \Rightarrow S = S_m$ gilt.

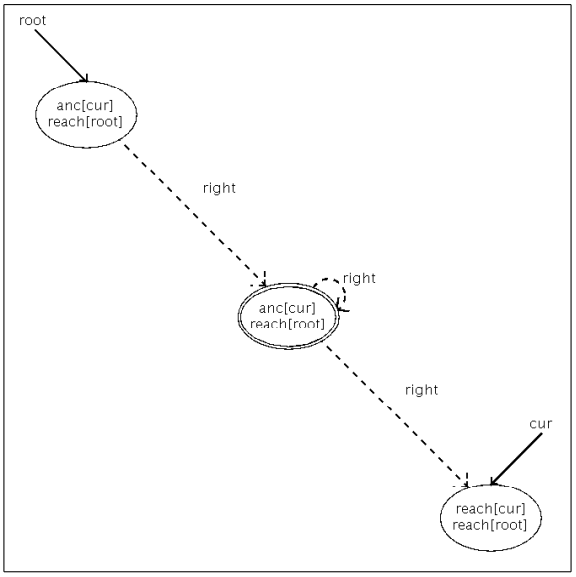

Abbildung 7.2: Eine induzierte Teilstruktur (Suchbaum)

Zu dieser Klasse gehören zum Beispiel auch die vier Shapegraphen aus Abbildung 7.1. Der Graph in Abbildung 7.2 ist nicht nur der reduzierte Teilgraph jedes Shapegraphen dieser Klasse, er tritt auch selbst als Shapegraph auf. Insgesamt sind in dieser Klasse 10 Shapegraphen enthalten. Wenn wir ihre maximalen Elemente berechnen, dann erhalten wir derer vier. Sie sind in Abbildung 7.3 gezeigt.

S_{m1} : Der Shapegraph besteht nur aus dem Suchweg, der identisch mit dem induzierten Teilgraphen ist.

S_{m2} : Der Shapegraph ist ähnlich zu S_{m1}, jedoch ist die aktuelle Ecke cur kein Blatt des Baumes. Es existiert ein Teil des Baumes, der noch nicht (oder an dem noch nicht entlang) traversiert wurde. Er ist in einer Summary-Ecke „unterhalb“ von cur zusammengefasst.

S_{m3} : Der Shapegraph besteht aus dem Suchweg, wobei die aktuelle Ecke cur ein Blatt des Baumes ist. Entlang des Suchweges hängen Teilbäume – mindestens einer – herab, die in einer Summary-Ecke zusammengefasst sind.

S_{m4} : Der Shapegraph ist ähnlich zu S_{m3}, jedoch ist cur kein Blatt und es existieren noch nicht besuchte Teile des Graphen, die wie bei S_{m2} in einer weiteren Summary-Ecke unterhalb von cur zusammengefasst sind.

Unser Verfahren zur Algorithmenvisualisierung baut auf einer von TVLA berechneten Shapeanalyse auf. Die verwendeten Analysespezifikationen sind so beschaffen und die Shapeanalyse wird so gestaltet, dass die Einbettungsrelation antisymmetrisch auf der Analyseausgabe ist. Wir untersuchen, welche Konsequenzen entstehen,

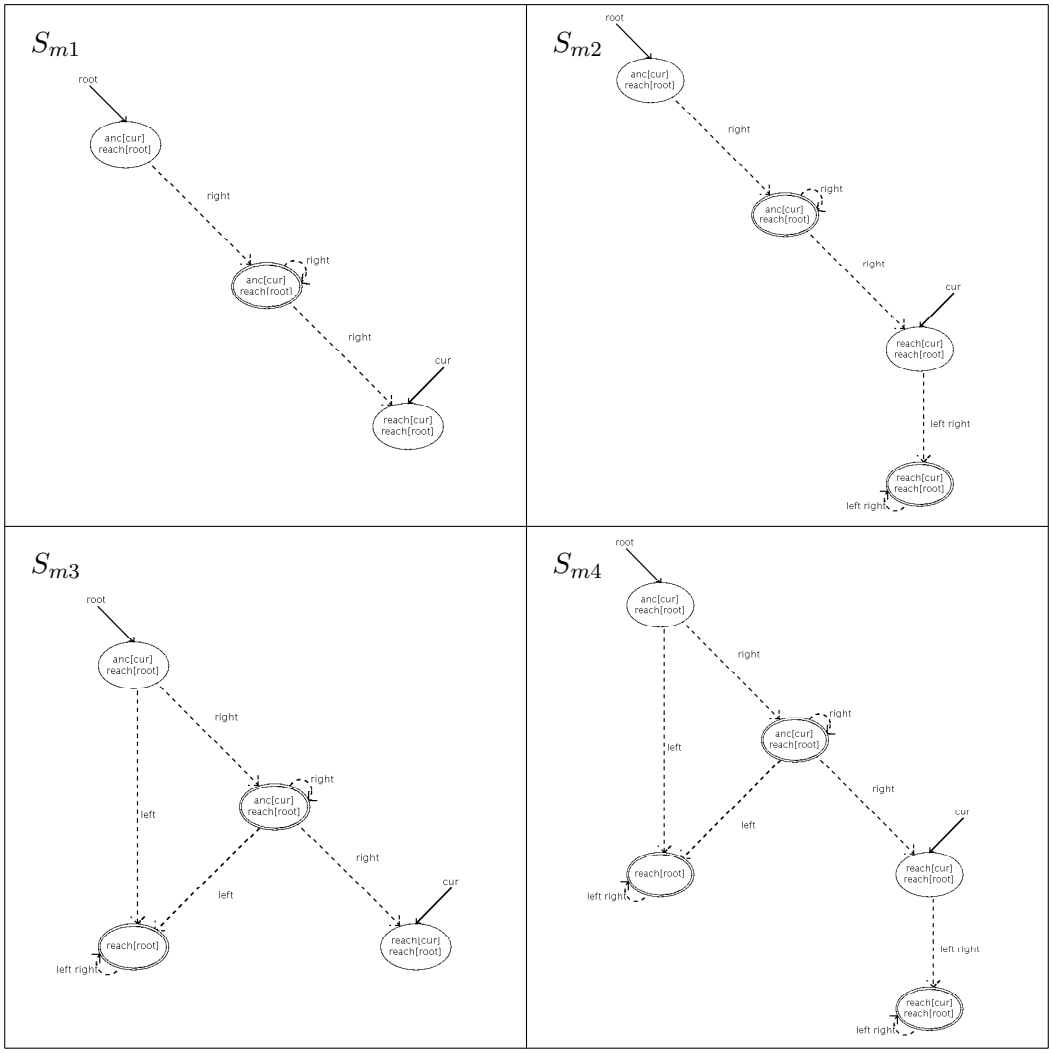

Abbildung 7.3: Bezüglich Einbettung maximale Shapegraphen der Klasse mit induzierter Teilstruktur wie in Abbildung 7.2

wenn diese Bedingung nicht erfüllt ist. Der Sichtweise der Einbettung als Relation wollen wir eine zweite Sichtweise, eine graphentheoretische, zur Seite stellen. Einer Shapegraphenmenge, auf der die Einbettungsrelation erklärt ist, ordnen wir einen gerichteten Graphen zu. Seine Ecken sind die Shapegraphen und er erhält eine Kante von S nach T genau dann, wenn $S \neq T$ und S in T einbettbar ist. (Die Reflexivität der Einbettung würde sonst zu Schlingen führen, die nicht erwünscht sind.) Die Transitivität bedeutet, dass, wenn immer es einen (gerichteten) Kantenzug von S nach T gibt, es auch eine Kante von S nach T gibt. Betrachten wir als Erstes den Fall, dass $\sqsubseteq$ nicht antisymmetrisch auf $\mathcal{M}$ ist. Dann existieren zwei Shapegraphen S und T mit $S \sqsubseteq T$ und $T \sqsubseteq S$. Im Graphen existiert also eine Kante von S

nach T und eine von T nach S. Der Graph enthält einen Kreis. Als Zweites setzten wir die Existenz eines Kreises im Relationsgraphen voraus. Zwei seiner Ecken, die zudem auch benachbart sein sollen, seien S und T. Damit existiert ein Kantenzug von T nach S und wegen der Transitivität der Einbettungsrelation auch eine Kante von T nach S. Es gilt also $S \sqsubseteq T$ und $T \sqsubseteq S$, aber es ist $S \neq T$ (auch $S \not\simeq T$). Damit ist $\sqsubseteq$ nicht antisymmetrisch. Unter der Voraussetzung der Transitivität ist die Antisymmetrie äquivalent zur Kreisfreiheit des schlingenbefreiten Relationsgraphen.

Für die Definition der Einbettungsvisualisierung haben wir die Antisymmetrie der Einbettungsrelation auf der betrachteten Shapegraphenmenge $\mathcal{M}$ als Voraussetzung verwendet. Die Einbettung ist damit eine (partielle) Ordnung auf $\mathcal{M}$. Was kann geschehen, wenn Antisymmetrie nicht besteht? Eine Möglichkeit besteht darin, dass sich die Methode als nicht anwendbar erklärt. Das stellt eine potentielle Einschränkung dar. Wir können die Methode aber auf Shapegraphenmengen verallgemeinern, auf denen die Einbettungsrelation nicht antisymmetrisch ist. Wenn der (schlingenbefreite) Relationsgraph einen Kreis enthält, dann kann jeder Shapegraph des Kreises wegen der Transitivität der Einbettungsrelation in jeden anderen eingebettet werden. Wir können daher den Kreis zu einer einzelnen Ecke kontrahieren. Dies iterieren wir, bis der Graph kreisfrei ist. Das bedeutet aber nichts anderes, als dass wir die starken Zusammenhangskomponenten berechnen und sie jeweils zu einer Ecke kontrahieren. In dem so entstandenen Hilfsgraphen bestimmen wir die maximalen Elemente und fügen sie zur Ausgabemenge zusammen.

Die Methode der Einbettungsvisualisierung wählt aus einer Menge $\mathcal{M}$ von Shapegraphen einige repräsentative Vertreter aus. Diese werden so bestimmt, dass die durch sie beschriebenen konkreten Heapsituationen alle von den Shapegraphen aus $\mathcal{M}$ beschriebenen umfassen. Formal haben wir dies auf den Begriff der Einbettung zurückgeführt. Die Einbettungsvisualisierung berechnet die bezüglich der Einbettungsrelation maximalen Elemente von $\mathcal{M}$. Trivialerweise ist die Ausgabe damit eine Teilmenge der Eingabe. Die Einbettungsvisualisierung führt in der Regel also zu einer Verringerung der Visualisierungskomplexität.

Reduzierter-Teilgraph-Visualisierung

Es sei $\mathcal{M}$ eine Klasse der Shapegraphenmenge eines Programmpunkt eines Algorithmus, die durch einen gegebenen Partitionsdefinierer induziert sei. Damit ist $\mathcal{M}$ insbesondere von der leeren Menge verschieden und endlich. Jede Formel des Formelpartitionierers wertet für alle Shapegraphen aus $\mathcal{M}$ zum gleichen Wahrheitswert aus. Des Weiteren haben alle Shapegraphen aus $\mathcal{M}$ die gleiche reduzierte Teilstruktur, vergleiche Definition 6.7 auf Seite 112. Den Formelpartitionierer wollten wir bei der Betrachtung der Methoden außer Acht lassen, unser Augenmerk liegt auf den Shapegraphen selbst. Nicht nur, dass alle Shapegraphen in $\mathcal{M}$ (bis auf Isomorphie)

den gleichen reduzierten Teilgraphen R haben, kein Shapegraph zu diesem Programmpunkt außerhalb von $\mathcal{M}$ besitzt R als Teilgraph. Damit ist der gemeinsame Teilgraph R eine sinnvolle Charakterisierung der Klasse $\mathcal{M}$.

Die Methode der Reduzierter-Teilgraph-Visualisierung berechnet für eine Klasse $\mathcal{M}$ deren gemeinsame reduzierte Teilstruktur R, die als einziger Graph ihre Ausgabe bildet. Wenn man die Tatsache explizit nutzt, dass es sich bei $\mathcal{M}$ um eine Klasse einer induzierten Partition handelt, dann gestaltet sich die Berechnung sehr effizient: Man braucht nur für einen Shapegraphen aus $\mathcal{M}$ seine induzierte Teilstruktur zu berechnen. – Der Fall $R \in \mathcal{M}$ ist möglich, vergleiche das Beispiel aus den Abbildungen 7.1, 7.2 und 7.3. Häufig wird aber $R \notin \mathcal{M}$ gelten. Wir interpretieren die Methode deshalb besser als eine Funktion, die aus einer Shapegraphenmenge einen Visualisierungsgraphen ableitet. Da die Ausgabemenge nur aus einem einzigen Shapegraph besteht, der zudem ein Teilgraph jedes Shapegraphen aus $\mathcal{M}$ ist, ist die gesamte Visualisierungskomplexität, sowohl die Anzahl der Graphen in der Ausgabe als auch deren Größe, in der Regel sehr gering.

Wir haben bei der Beschreibung dieser Methode explizit davon Gebrauch gemacht, dass es sich bei der Eingabemenge $\mathcal{M}$ um eine Klasse einer durch einen Partitionsdefinierer induzierten Partition handelt. Wenn dies nicht der Fall ist, dann brauchen nicht alle Shapegraphen in $\mathcal{M}$ dieselbe reduzierte Teilstruktur zu besitzen, sofern überhaupt ein Teilstrukturdefinierer gegeben ist und wir von einer Teilstruktur sprechen können. Und selbst wenn alle Shapegraphen aus $\mathcal{M}$ dieselbe induzierte Teilstruktur besitzen, können auch Shapegraphen außerhalb von $\mathcal{M}$ diese haben. In diesem Fall ist völlig unklar, was genau diese Teilstruktur in der Menge $\mathcal{M}$ im Verhältnis zu anderen Shapegraphenmengen charakterisiert. Formal spricht nichts dagegen, die Methode auf beliebige Shapegraphenmengen zu verallgemeinern: Als Ausgabe berechnen wir die reduzierten Teilgraphen der Shapegraphen der Eingabemenge. Allerdings ist mit einer deutlich verminderten Aussagekraft der zurückgegebenen Menge mit den reduzierten Teilstrukturen zu rechnen. – Es ist auch denkbar, die Methode dahingehend zu verallgemeinern, dass man den Durchschnitt aller Graphen aus $\mathcal{M}$ berechnet, also die größte in allen Shapegraphen enthaltene Teilstruktur. Dazu würden wir noch voraussetzen müssen, dass es sich bei $\mathcal{M}$ um eine Menge von Shapegraphen mit kanonischem Universum handelt. Für eine Klasse einer induzierten Partition ist der gemeinsame Teilgraph eine Teilstruktur des Durchschnitts. Abgesehen von der Tatsache, dass uns diese Verallgemeinerung ein gutes Stück von der ursprünglichen Idee fortträgt, ist auch dann wieder unklar, wie sich dieser Durchschnitt zu den Durchschnitten der Shapegraphen anderer Shapegraphenteilmengen (des Programmpunktes) verhält.

Single-Structure-Visualisierung

Wieder bezeichne $\mathcal{M}$ eine endliche und nichtleere Menge von Shapegraphen. Die Single-Structure-Visualisierung beruht auf der Idee, aus der Eingabemenge $\mathcal{M}$ einen

einzigen Graphen zu berechnen. Dieser Graph wird, außer wenn $\mathcal{M}$ nur aus einem einzigen Shapegraphen besteht, kein bei den in dieser Arbeit betrachteten Analysen vorkommender Shapegraph sein. Wir charakterisieren den berechneten Ausgabegraphen daher als Visualisierungsgraphen. In der Anfangszeit der Entwicklung von TVLA, als die Berechnung von Shapeanalysen noch nicht so effizient wie heutzutage möglich war, wollte man trotzdem über die Möglichkeit verfügen, größere Analysen durchführen zu können. Dafür war man gewillt, (stärkere) Informationsvergröberungen in Kauf zu nehmen. Das hauptsächliche Ziel war eine Terminierung der Shapeanalyse in akzeptabler Zeit. Zu diesem Zweck wurde die Single-Structure-Analyse in TVLA eingeführt. Alle an einem Programmpunkt auftretenden Heapsituationen werden durch einen einzigen (Shape-)Graphen repräsentiert. Genaugenommen handelt es sich dabei um eine Join-Methode, die als Vereinigungsoperator realisiert ist.

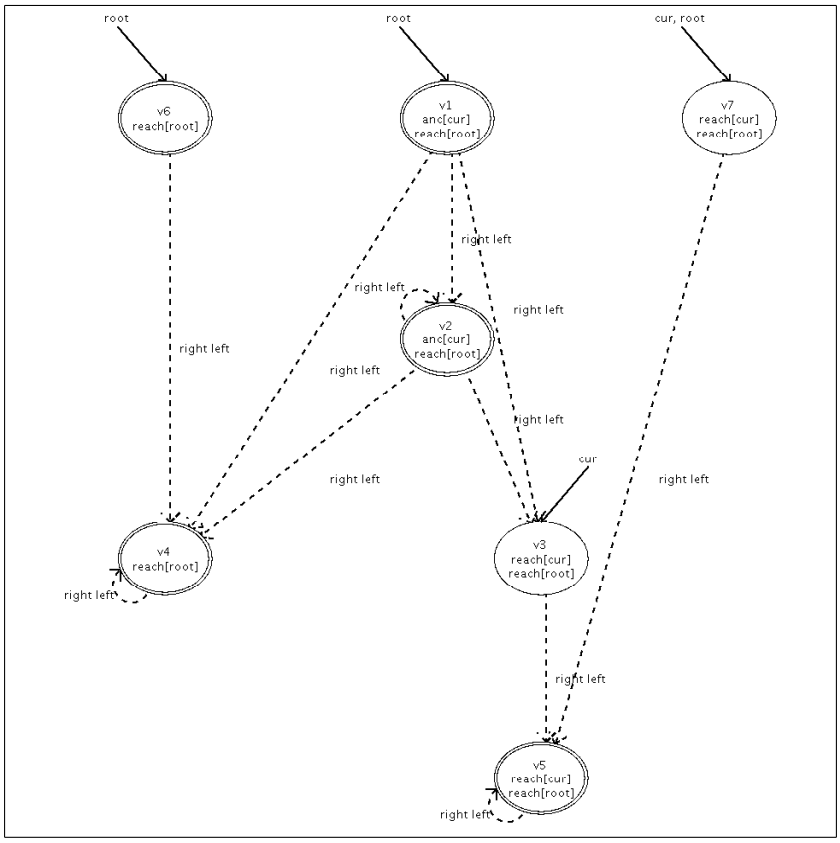

Abbildung 7.4: Ausgabe der Single-Structure-Analyse für den Schleifeneingangsprogrammpunkt des Baumsuchalgorithmus

Wenn wir für den Suchalgorithmus auf Suchbäumen, er ist in Abbildung 4.3 auf Seite 66 dargestellt, eine Single-Structure-Analyse durchführen, dann erhalten wir für den Schleifeneingangsprogrammpunkt n2 den Graphen in Abbildung 7.4. Er re-

präsentiert alle an diesem Programmpunkt auftretenden Heapsituationen. Es sticht unmittelbar ins Auge, dass es mehrere potentielle Wurzeln, nämlich v1, v6 und v7, gibt. Bei den mit Standardanalysen erzeugten Shapegraphen ist das undenkbar: Zeigervariablen sind in der Analysespezifikation mit dem Attribut „unique“ versehen, welches sicherstellt, dass es in einem Shapegraphen nur eine Ecke gibt, für die dieses Prädikat wahr ist. Jede Ecke im Single-Structure-Graphen steht wie bisher für konkrete Heapzellen, für welche die unären Prädikatswerte zu den zugehörigen Wahrheitswerten auswerten. Wenn eine Ecke im Single-Structure-Graphen vorhanden ist, dann bedeutet dies, dass es wenigstens eine konkrete Heapsituation bei diesem Programmpunkt gibt, in der Zellen mit den entsprechenden Eigenschaften existieren. In dieser konkreten Heapsituation brauchen aber nicht zu jeder Ecke des Single-Structure-Graphen Heapzellen existieren. Dieses Faktum erschwert das Lesen eines Shapegraphen deutlich.

Untersuchen wir den Single-Structure-Graphen in Abbildung 7.4 genauer. Als Erstes betrachten wir die Situationen, die beim erstmaligen Erreichen des Schleifeneingangspunktes vorliegen. Der Zeiger cur zeigt dann auf die Wurzel. Die zugehörige Ecke im Graphen ist v7. Besteht der Baum aus weiteren Heapzellen, dann werden diese durch v5 repräsentiert. Als Zweites betrachten wir Situationen, in denen der Suchweg aus wenigstens drei Heapzellen besteht. Im Graphen ist dieser Weg durch die Ecken v1, v2 und v3 repräsentiert. Die am Suchweg eventuell herabhängenden Teilbäume sind durch v4, der noch nicht besuchte Teil des Baumes durch v5 repräsentiert. Die Situationen, in denen der Suchweg nur aus der Wurzel und der (davon verschiedenen) aktuellen Ecke besteht, sind dadurch eingeschlossen, dass es left- und right-Kanten von v1 nach v3 gibt. Als Drittes betrachten wir den Fall, dass die Suche erfolglos war, der Zeiger cur ist dann NULL. Dann besteht der Baum entweder nur aus der Wurzel, der Ecke v6, oder es gibt weitere Heapzellen, die durch v4 repräsentiert sind. Durch die Vereinigung ist auch eine Vergröberung entstanden: die Ecken v1 und v6 sind Summary-Ecken.

Wir konnten den Graphen in Abbildung 7.4 derart leicht interpretieren, weil wir die konkret auftretenden Heapsituationen kennen. Bei einer Visualisierung im Hinblick auf Lernziele ist aber die andere Richtung maßgebend. Wir sehen den Graphen und wollen Aussagen über die konkreten Heapsituationen ableiten. Wenn wir nichts über binäre Suche wüssten, würden wir dann mit diesem Graphen etwas anfangen können? Würden wir überhaupt erkennen, dass es um Bäume geht? Daher verlangen wir im Rahmen unseres Visualisierungsansatzes in der auf Seite 61 formulierten Strukturerhaltungsforderung, dass die graphentheoretische Struktur eines Shapegraphen die Zeigerstruktur der durch ihn repräsentierten konkreten Heapsituationen widerspiegelt soll. Diese Forderung wird von Single-Structure-Graphen (bis auf Ausnahmen) verletzt.

Die Single-Structure-Visualisierung berechnet für eine gegebene Shapegraphenmenge $\mathcal{M}$ mittels der Single-Structure-Join-Methode eine einzelne Struktur für $\mathcal{M}$, die

den einzigen Graphen der Ausgabemenge bildet: Wir initialisieren eine leere Struktur A. Anschließend iterieren wir über alle Shapegaphen $S \in \mathcal{M}$, in jedem Iterationsschritt vereinigen wir A mit S mittels der Single-Structure-Join-Methode. Zurückgegeben wird als Ausgabe die Menge $\{A\}$. Es handelt sich zwar nur um einen einzigen Graphen, die primäre Visualisierungskomplexität ist also gering, dieser ist aber im Verhältnis zu den Eingabegraphen normalerweise relativ groß.

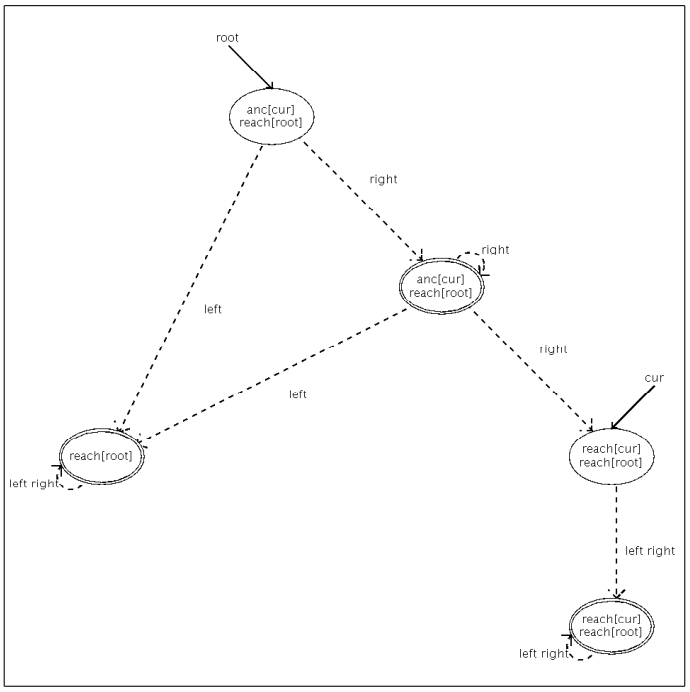

Abbildung 7.5: Single-Structure-Graph für die Klasse mit induzierter Teilstruktur wie in Abbildung 7.2

Betrachten wir als Beispiel wieder die Klasse mit induzierter Teilstruktur wie in Abbildung 7.2. Wenn wir die 10 Shapegraphen der Klasse als Eingabe der Single-Structure-Methode verwenden, dann erhalten wir als Ausgabe den Graphen in Abbildung 7.5. Wie wir in der vorangegangenen Diskussion dargestellt haben, braucht es in einer konkreten Heapsituation nicht zu jeder Ecke des Graphen konkrete Heapzellen geben, die zu ihr korrespondieren. Diese Tatsache erschwert die Interpretation des Graphen normalerweise stark. In diesem speziellen Fall ist die Situation jedoch nicht dramatisch. Alle Shapegraphen der Klasse haben als induzierte Teilstruktur den Graphen aus Abbildung 7.2. Lediglich die beiden Summary-Ecken, die für die am Suchweg herabhängenden Teilbäume und für den noch nicht besuchten Teil des Baumes stehen, können fehlen.

V-Ecken-Visualisierung

Als Ausgangspunkt betrachten wir wieder eine nichtleere und endliche Menge von Shapegraphen, die die Eingabe der Methode sei. Wir knüpfen an das Beispiel an, das wir bei der Entwicklung der Einbettungsvisualisierung untersucht hatten. Dort haben wir eine Klasse einer induzierten Partition für den Schleifeneingangspunkt des Suchalgorithmus betrachtet. Die Shapegraphen dieses Programmpunktes hatten wir hinsichtlich der Struktur des Weges von der Wurzel zur aktuellen Ecken partitioniert. Eine der Klassen – diejenige, bei der die induzierte Teilstruktur jedes Graphen der Shapegraphen aus Abbildung 7.2 ist – haben wir genauer betrachtet. Alle Shapegraphen dieser Klasse haben natürlich dieselbe Struktur des Suchweges. Entlang des Suchweges können Teilbäume herunterhängen. Ist die aktuelle Ecke cur kein Blatt des Baumes, dann können sich „unterhalb“ cur bisher noch nicht betrachtete Teile des Baumes befinden. Diese beiden Arten von Baumteilen sind in den jeweiligen Shapegraphen durch Summary-Ecken repräsentiert. Die Shapegraphen der Klasse unterscheiden sich dadurch, dass diese Summary-Ecken vorhanden sind oder nicht; auch können sich (zusätzlich) die Werte binärer Prädikate dort unterscheiden, wo diese Summary-Ecken als Argument auftreten.

Für uns Menschen hat die Vorstellung von „kann vorhanden sein oder nicht“ durchaus natürlichen Charakter. Insbesondere bei dieser Anwendung, wo das Hauptaugenmerk auf der Struktur des Suchweges liegt, würde sich eine solche Sicht anbieten; zudem würde damit die Klasse sehr kompakt beschrieben werden können. In der Shapeanalyse und damit auch in TVLA werden nur zwei Arten von Individuen (Ecken) unterschieden: solche, die für genau eine konkrete Heapzelle stehen (Individuen-Ecke), und solche, die für mindestens eine konkrete Heapzelle stehen (Summary-Ecke). Es gibt aber keine Individuenart, die beispielsweise null oder mehr konkrete Heapzellen repräsentiert.[3] Da unsere Visualisierung auf einer Shapeanalyse basiert, müssen wir diese Gegebenheit respektieren. Dessen ungeachtet verbietet es sich nicht, die Analyseausgabe nachträglich um eine weitere Eckenart anzureichern. Wir zeichnen Ecken aus, von denen wir erlauben, dass sie keine konkrete Entsprechung zu besitzen brauchen, diese Ecken stehen also für null oder mehr konkrete Heapzellen. Wir werden sie *V-Ecken* (auch *V-Individuen*) nennen. Den Namen können wir als Visualisierungsecke lesen, schließlich werden sie im Hinblick auf die Visualisierung eingeführt, oder wir können ihn als virtuelle Ecke lesen, was unterstreicht, dass die Ecke gegebenenfalls gar nicht vorhanden ist, dass es gegebenenfalls keine konkrete Heapzelle gibt, die ihr entspricht.

Methodisch wird sich die V-Ecken-Visualisierung als eine Verallgemeinerung oder Erweiterung der Einbettungsvisualisierung darstellen. Anstelle der dort verwendeten Einbettungsrelation $\sqsubseteq$ verwenden wir eine modifizierte Art der Einbettung, die das

3. Die Single-Structure-Analyse stellt eine gewisse Ausnahme dar, sie ist aber keine „Standardanalyse“. Überdies ist ihre berechnete Ausgabe unvereinbar mit der Strukturerhaltungsforderung, die wir auf Seite 61 im Hinblick auf die Visualisierung fordern.

Konzept der V-Ecke mit einbezieht. – Wir beginnen damit, dass wir erklären, was V-Ecken sind:

7.3 Definition. *Es sei S ein Shapegraph. Eine Teilmenge V_S seines Universums sei ausgezeichnet. Die Elemente von V_S bezeichnen wir als* V-Ecken*, den Shapegraphen S nennen wir einen* V-Ecken-Graph*.*

Diese Eckenmenge V_S für jeden Shapegraphen S kann auf zweierlei Arten natürlich verwaltet werden. Zum Einen kann sie als Teilmenge der Ecken explizit zu jedem Shapegraphen gespeichert werden. Wir können aber auch nachträglich ein neues einstelliges Prädikat einführen. Es erhält für eine Ecken in einem Shapegraphen den Wahrheitswert 1, wenn die betreffende Ecken eine V-Ecke ist, und den Wert 0 andernfalls. Das Konzept der V-Ecke benutzen wir in der Visualisierung. Es ist nicht nötig, die Analysespezifikation um dieses Prädikat zu ergänzen und die Analyse damit durchzuführen. Das Hinzufügen braucht erst danach, in der Visualisierungsaufbereitung zu geschehen.

Durch Zulassen von V-Ecken schwächen wir den Begriff der Einbettung ab. Wir setzen:

7.4 Definition. *Es seien S und T zwei Shapegraphen. In T sei eine Teilmenge V_T seines Universums als V-Ecken ausgezeichnet. Der Shapegraph S heiße* V-einbettbar *in T, wenn es eine Abbildung f des Universums von S in das Universum von T gibt, die folgende Bedingungen erfüllt:*

1. *Bis auf die Surjektivität erfüllt f die Bedingungen einer Einbettung aus Definition 3.2, das heißt:*
   a) *Für jedes Prädikat (Prädikatssymbol) p, es habe Stelligkeit n, und für alle $u_1, \ldots, u_n \in S$ gelte*

   $$p^S(u_1, \ldots, u_n) \sqsubseteq p^T\big(f(u_1), \ldots, f(u_n)\big),$$

   wobei $\sqsubseteq$ die Informationsordnung auf der Menge der Wahrheitswerte bezeichne.
   b) *Für jedes $u_T \in T$ gelte*

   $$(|\{u \in S\colon f(u) = u_T\}| > 1) \sqsubseteq sm^T(u_T).$$

2. *Für jedes $u_T \in T$ gelte: Ist u_T nicht im Bild von f, dann ist u_T eine V-Ecke, also:*

   $$\forall u_T \in T\colon\ (\forall u \in S\colon f(u) \neq u_T) \Rightarrow u_T \in V_T.$$

Ist S in T V-einbettbar, dann schreiben wir dafür $S \sqsubseteq_V T$.

Unmittelbar aus der Definition ergibt sich $S \sqsubseteq T \Rightarrow S \sqsubseteq_V T$. Jede Ecke von T tritt hier bei einer Einbettung als Bild auf, daher ist die Menge der V-Ecken in diesem Fall nicht von Bedeutung. Aus diesem Argument folgt auch, dass die V-Einbettung reflexiv ist. – Die V-Einbettung gestattet noch eine zweite Sicht. In der Definition verlangen wir, dass einige Ecken in T bei der Einbettung „übrig bleiben“ dürfen. Diese können wir aber auch vorher aus T entfernen und dann eine (normale) Einbettung verwenden. In dieser Lesart gilt:

7.5 Satz. *Es seien S und T zwei Shapegraphen. Ihre kanonischen Universen seien U_S und U_T, und es gelte $U_S \subseteq U_T$. In T seien (wenigstens) alle Ecken aus $U_T \setminus U_S$ als V-Ecken ausgezeichnet. Der Shapegraph T' entstehe aus T indem alle Ecken aus $U_T \setminus U_S$ aus seinem Universum gelöscht und alle Prädikate auf das neue Universum eingeschränkt werden. Dann gilt $S \sqsubseteq_V T \Leftrightarrow S \sqsubseteq T'$.*

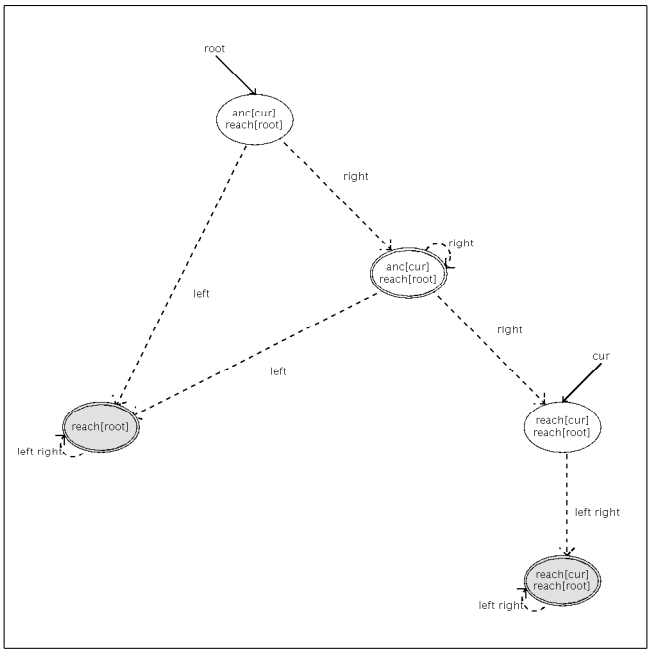

Abbildung 7.6: Visualisierungsgraph mit V-Ecken

Betrachten wir zur Veranschaulichung ein Beispiel. Es sei T der Graph aus Abbildung 7.6. Als V-Ecken zeichnen wir die beiden Summary-Ecken aus, die nicht auf dem Suchweg liegen; in der Abbildung sind sie eingefärbt dargestellt. Des Weiteren sei S der Shapegraph aus Abbildung 7.2, er besteht nur aus dem Suchweg. Jede Funktion, die eine (V-)Einbettung vermittelt, muss kanonische Namen invariant lassen. Daher müssen korrespondierende Ecken des Weges von der Wurzel zur aktuellen Ecke aufeinander abgebildet werden. Das bestimmt die Funktion eindeutig. Die nur aus dem Suchweg bestehende Teilstruktur von T und S stimmen überein. Die Bedingungen in Definition 7.4 sind daher erfüllt, es gilt $S \sqsubseteq_V T$.

Die V-Einbettung ist eine binäre Relation. Von ihrer Reflexivität haben wir uns schon überzeugt. Wie verhält es sich mit ihrer Transitivität? Dazu betrachten wir die drei Shapegraphen S_{m1}, S_{m2} und S_{m4} in Abbildung 7.3. In S_{m2} erklären wir die Summary-Ecke unterhalb von cur als V-Ecke. Dann gilt $S_{m1} \sqsubseteq_V S_{m2}$. In S_{m4} erklären wir die Summary-Ecke, die für die am Suchweg herabhängenden Teilbäume steht, als V-Ecke. Es ist $S_{m2} \sqsubseteq_V S_{m4}$. Aber es gilt $S_{m1} \not\sqsubseteq_V S_{m4}$. Würden wir in S_{m4} die beiden nicht auf dem Suchweg liegenden Summary-Ecken als V-Ecken auszeichnen, dann würde auch $S_{m1} \sqsubseteq_V S_{m4}$ gelten. Ob Transitivität besteht oder nicht, hängt also mit der Wahl der V-Ecken zusammen. Im ersten Fall hatten wir in S_{m2} die Summary-Ecke unter cur als V-Ecke ausgezeichnet, während wir dieselbe Ecke, also die mit gleichem kanonischen Namen, in S_{m4} nicht als V-Ecke ausgezeichnet haben. Diese Inkonsistenz ist der Grund für die Nichttransitivität. Für eine Menge von Shapegraphen wollen und werden wir daher die V-Ecken auf konsistente Weise auszeichnen.

7.6 Definition. *Es sei V eine Teilmenge der Menge aller kanonischen Namen bezüglich der Analysespezifikation. Des Weiteren sei $\mathcal{M}$ eine endliche und nichtleere Menge von Shapegraphen, die ohne Beschränkung der Allgemeinheit kanonische Universen besitzen. Für jeden Shapegraphen $S \in \mathcal{M}$ sei eine Teilmenge V_S seines Universums U_S als V-Ecken ausgezeichnet. Wir nennen die Auszeichnung* konform für $\mathcal{M}$, *wenn $V_S = U_S \cap V$ gilt.*

Wenn wir den Gedankengang zur Transitivität im obigen Absatz formal ausformulieren, dann erhalten wir:

7.7 Satz. *Es sei $\mathcal{M}$ eine endliche und nichtleere Menge von Shapegraphen mit einer konformen V-Ecken-Auszeichnung. Dann ist die V-Einbettung transitiv auf $\mathcal{M}$.*

Die gewünschte Konsistenz werden wir dadurch sicherstellen, dass wir eine konforme Auszeichnung der V-Ecken durchführen. Dieses Vorgehen bietet sich aus didaktischen Gesichtspunkten an. Wir sehen unseren Ansatz zur Algorithmenvisualisierung in einem weiten Sinn als ein lernunterstützendes Mittel. Es dürfte daher, speziell für wenig fortgeschrittene Benutzer, wenig sinnvoll sein, von Programmpunkt zu Programmpunkt oder gar von Klasse zu Klasse verschiedene Sätze von V-Ecken zu verwenden. Methodisch werden wir die Spezifikation der V-Ecken durch eine Eckenauswahlformel φ_V vornehmen. Auf diese Weise sind wir methodisch konsistent mit unserem Vorgehen in Abschnitt 6.3, wo wir die für einen Shapegraphen induzierte Teilstruktur auch unter Verwendung einer Eckenauswahlformel definiert haben, vergleiche Definition 6.7.

Da wir die Formel φ_V für die Ecken von Shapegraphen auswerten, verlangen wir wieder, dass sie höchstens eine freie Variable habe. Dies machen wir durch die Schreibweise $\varphi_V(v)$ deutlich. In einem Shapegraphen S wird eine Ecke u genau dann als V-Ecke ausgezeichnet, wenn $W_{S,\beta[\frac{v}{u}]}\big(\varphi_V(v)\big) = 1$ gilt. Wir nennen die Formel $\varphi_V(v)$ konform für eine Shapegraphenmenge $\mathcal{M}$, wenn die durch sie bestimmte

Auszeichnung konform für $\mathcal{M}$ ist. Wir nennen sie *konform*, wenn sie konform für jede Shapegraphenmenge ist.

Die Bedingung konform für jede Shapegraphenmenge ist äquivalent dazu, dass sie für die Menge aller (möglichen) Shapgegraphen bezüglich der Analysespezifikation konform ist. Dies wird erfüllt sein, wenn die Formel $\varphi_V(v)$ so beschaffen ist, dass die Eckenauswahl unabhängig von etwaigen Eigenschaften einzelner Shapegraphen ist. Daher sind Quantoren, sowie null- und zweistellige Prädikate problematisch. Die Formel $\varphi_V(v)$ soll letztendlich kanonische Namen auswählen, sie soll eine Teilmenge der kanonischen Namen bezüglich der verwendeten Analysespezifikation auszeichnen. Für eine syntaktische Auszeichnung von $\varphi_V(v)$ stellen sich dieselben Probleme wie in Abschnitt 6.5.2 bei der Eckenauswahlformel.

Jetzt haben wir alles beisammen, um die Methode formal definieren zu können. Wir gehen dabei wie bei der Einbettungsvisualisierung vor, anstelle der Einbettungsrelation verwenden wir die V-Einbettung. Auch in diesem Fall müssen wir berücksichtigen, dass die Relation nicht antisymmetrisch zu sein braucht. Der abgeleitete Relationsgraph enthält dann Kreise. Wie zuvor berechnen wir die starken Zusammenhangskomponenten, was in zur Größe des Graphen linearen Zeit möglich ist, vergleiche [Cormen u. a. 2001, Abschnitt 22.5], und kontrahieren jede Zusammenhangskomponente zu einer einzelnen Ecke. Wir setzen:

7.8 Definition. *Es sei $\mathcal{M}$ eine Menge von Shapegraphen und $\varphi_V(v)$ eine konforme Eckenauszeichnung. Die Methode der* V-Ecken-Visualisierung *sei durch folgenden Algorithmus gegeben:*

1. *Berechne zu $\sqsubseteq_V$ den schlingenbefreiten Relationsgraphen.*
2. *Berechne die starken Zusammenhangskomponenten und kontrahiere sie jeweils zu einem Shapegraphen.*
3. *Bestimme unter den verbliebenen Shapegraphen die bezüglich $\sqsubseteq_V$ maximalen Elemente, dies sind die Ecken ohne ausgehende Kanten.*
4. *Ausgabe: Menge der maximalen Elemente.*

Wir demonstrieren und untersuchen, wie sich die V-Einbettung bei dem Suchbeispiel verhält. Dazu betrachten wir das Beispiel, das wir auch bei der Einbettungsvisualisierung untersucht haben. Wir befinden uns am Programmpunkt n2, dem Schleifeneingangspunkt, und haben die Shapegraphenmenge dieses Programmpunktes gemäß der Struktur des Suchweges partitioniert. Wir betrachten die Klasse, in der alle Shapegraphen als reduzierte Teilstruktur den Shapegraphen aus Abbildung 7.2 haben. Bei der Diskussion der Einbettungsvisualisierung haben wir gezeigt, dass diese Klasse vier (bezüglich der Einbettungsrelation $\sqsubseteq$) maximale Elemente hat, sie sind in Abbildung 7.3 gezeigt. Darauf können wir aufbauen. Für je zwei Shapegraphen S und T gilt $S \sqsubseteq T \Rightarrow S \sqsubseteq_V T$. Daher genügt es, wenn wir für die bezüglich $\sqsubseteq$ maximalen Shapegraphen V-Ecken-Graphen bestimmen.

Als V-Ecken zeichnen wir in jedem Shapegraphen alle Ecken aus, die nicht auf dem Suchweg liegen. Wir verwenden als Eckenauswahlformel

$$\varphi_V(v) := \neg\, \mathsf{cur}(v) \wedge \neg\, \mathsf{anc}[\mathsf{cur}](v).$$

Diese Formel ist die negierte Eckenauswahlformel, die wir zur Definition der induzierten Teilstruktur jedes Shapegraphen verwenden. Damit ist klar, dass eine Ecke genau dann eine V-Ecke ist, wenn sie nicht auf dem Suchweg liegt, wenn sie nicht in der gemeinsamen induzierten Teilstruktur der Klasse liegt. Betrachten wir nun den V-Ecken-Graph T aus Abbildung 7.6. Wir haben uns oben schon davon überzeugt, dass $S_{m1} \sqsubseteq_V T$ gilt. Ebenso kann leicht verifiziert werden, dass jeder der vier maximalen Shapegraphen V-einbettbar in T ist. Die 10 Shapegraphen der Klasse lassen sich folglich durch einen einzigen V-Ecken-Graph repräsentieren. Das stellt eine erhebliche Reduzierung der Visualisierungskomplexität dar.

7.1.3 Vergleich der Methoden

Diese Arbeit handelt von der Aufbereitung der Shapeanalyseausgabe im Hinblick auf ihre Visualisierung. Dazu haben wir mit dem Ähnlichkeitskonzept eine Methode eingeführt, um die Analyseausgabe zu strukturieren. Dieser Abschnitt beschäftigt sich mit einer weiteren, nachgeschalteten Form der Aufbereitung. Die Anwendung der hier besprochenen Methoden erfolgt nach der Strukturierung, aber vor der eigentlichen Visualisierung. Im vorangegangenen Teilabschnitt haben wir verschiedene Methoden zur Visualisierung einer Klasse vorgestellt. Dort haben wir uns in erster Linie auf die Beschreibung der Methode und die mit ihr zusammenhängenden und benötigten theoretischen Konzepte konzentriert. Dieser Teilabschnitt behandelt die Methoden unter einem pragmatischen Gesichtspunkt. Wir vergleichen sie miteinander, und wir charakterisieren sie unter dem Gesichtspunkt, wie sie sich in Bezug auf die Visualisierung verhalten.

Die Methoden, die wir im vorangegangenen Teilabschnitt besprochen haben, sind: simultane Visualisierung, Einbettungsvisualisierung, Reduzierter-Teilgraph-Visualisierung, Single-Structure-Visualisierung und V-Ecken-Visualisierung. In Bezug auf die V-Ecken-Visualisierung ist anzumerken, dass sie (neben der Shapegraphenmenge) eine weitere Eingabe benötigt: eine konforme V-Ecken-Auszeichnung. Deren Festlegung ist Aufgabe des Visualisierungsinszenators, vergleiche Kapitel 9. Häufig stellt dies aber keinen zusätzlichen Aufwand dar: In vielen Fällen kann die Negation der Eckenauswahlformel für die V-Ecken-Auszeichnung verwendet werden. – Unter einem strukturellen Gesichtspunkt basieren sie auf unterschiedlichen Prinzipien. Eine Art wählt, beziehungsweise berechnet, aus der Shapegraphenmenge eine Teilmenge aus, die sie als repräsentativ oder charakteristisch für die Menge erklärt. Worin das Charakteristische besteht, hängt dabei von der Methode ab. Hierher gehören die simultane Visualisierung und die Einbettungsvisualisierung. Eine andere Art von Methode berechnet aus der Shapegraphenmenge als Ausgabe eine Menge

von Graphen, die nicht notwendigerweise Shapegraphen (des Programmpunktes) zu sein brauchen, aber trotzdem eine adäquate Repräsentation der Eingabemenge erlauben. Wir sprechen dann allgemeiner von Visualisierungsgraphen. Hierher gehören die Reduzierter-Teilgraph-, die Single-Structure- und die V-Ecken-Visualisierung.

Der ersten Gesichtspunkt, hinsichtlich dessen wir die Methoden miteinander vergleichen, sind die Anforderungen an die Eingabe. Grundsätzlich handelt es sich dabei stets um eine endliche und nichtleere Menge von Shapegraphen. Da wir die Methoden als einen Zwischenschritt zwischen Klasseneinteilung und der eigentlichen Visualisierung verstehen, haben wir natürlich in erster Linie Klassen als Eingabemengen im Sinn. Jedoch ist dies bei einigen Methoden nicht zwingend. Die simultane Visualisierung kann, ohne jedwede Einschränkung in Kauf nehmen zu müssen, beliebige Shapegraphenmengen als Eingabe erhalten. Das trifft im Prinzip auch auf die Einbettungs- und V-Ecken-Visualisierung sowie auf die Single-Structure-Visualisierung zu. Jedoch ist zu erwarten, dass die Qualität der berechneten (Visualisierungs-) Graphen im Hinblick auf die Visualisierungsziele bei Klassen höher ist. Die Reduzierter-Teilgraph-Visualisierung ist (eigentlich) nur auf Klassen anwendbar, da nur in diesem Fall die induzierte Teilstruktur eine sinnvolle Charakterisierung der Klasse erlaubt. – Auch wenn die Anwendung einzelner Methoden nicht auf Klassen einer Partition beschränkt ist, sondern sie auch auf beliebige Shapegraphenmengen angewandt werden können, so wollen wir uns in der restlichen Diskussion doch wieder, wie auch grundsätzlich in diesem Abschnitt, auf den Fall der Klasse beschränken.

Ein weiterer Gesichtspunkt, hinsichtlich dessen wir die Methoden miteinander vergleichen wollen, ist die Ausgabe. In den einleitenden Sätzen dieses Teilabschnitts haben wir schon angemerkt, dass wir es in den Ausgabemengen mit verschiedenen Arten von Graphen zu tun haben. Es kann sich um „normale“ Shapegraphen handeln, die dann jeweils sogar eine Teilmenge der Eingabemenge bilden, um „weniger normale“ Shapegraphen oder um abgeleitete Graphen, die wir dann neutral Visualisierungsgraphen nennen. Uns soll an dieser Stelle die Größe der Ausgabe interessieren, die wir zu der Größe der Eingabe in Beziehung setzen. Da die Ausgabe letztendlich das ist, was der eigentlichen Visualisierung übergeben wird, stellt die Ausgabegröße ein brauchbares Maß für die Komplexität des zu Visualisierenden dar. Wir unterscheiden zwei Größen, die Anzahl der Graphen und deren (maximale) Größe, die wir auch als *primäre* und *sekundäre Visualisierungskomplexität* bezeichnen.

Die simultane Visualisierung verhält sich neutral, was wenig überrascht: Sowohl primäre als auch sekundäre Visualisierungskomplexität sind im Verhältnis zur Eingabe gleich. Die Einbettungsvisualisierung bestimmt eine Teilmenge der Eingabe. Damit ist die primäre Visualisierungskomplexität kleiner oder gleich im Vergleich zur Eingabe. Das bei der Einführung der Methode betrachtete Beispiel hat gezeigt, dass bei geeigneten Anwendungsbedingungen mit einer (deutlichen) Reduzierung

zu rechnen ist. Des Weiteren bleibt die sekundäre Visualisierungskomplexität im Vergleich zur Eingabe unverändert. Die V-Ecken-Visualisierung ist eine Verallgemeinerung der Einbettungsvisualisierung. Die berechneten Shapegraphen sind nur um eine Eckenauszeichnung bereichert, die sekundäre Visualisierungskomplexität ist damit im Prinzip gleich geblieben. Die Anzahl der Shapegraphen in der Ausgabe ist höchstens so groß wie bei der Einbettungsvisualisierung. Das dort betrachtete Beispiel hat demonstriert, das bei geeigneten Anwendungsbedingungen eine (beeindruckend) deutliche Reduzierung der primären Visualisierungskomplexität möglich ist. Die Reduzierter-Teilgraph-Visualisierung gibt eine einelementige Menge zurück, der Graph ist zudem ein Teilgraph jedes Shapegraphen der Eingabe. Die primäre Visualisierungskomplexität hat sich gewaltig verringert. Bei der sekundären Visualisierungskomplexität hängt es stark vom Partitionsdefinierer ab; in der Regel, bei nichttrivialen Partitionsdefinierern, können wir eine Reduzierung erwarten. Die Single-Structure-Visualisierung berechnet ebenfalls nur einen einzelnen Graphen, die primäre Visualisierungkomplexität ist damit wieder äußerst gering. Allerdings hat dieser Graph den Charakter einer Vereinigung (der Shapegraphen der Eingabemenge), die sekundäre Visualisierungskomplexität ist deutlich angestiegen.

Eine kurze Erörterung des Berechnungsaufwandes soll an dieser Stelle angefügt werden. Wir vereinfachen die Untersuchung und betrachten nur die Abhängigkeit von der Größe $|\mathcal{M}|$ der Eingabemenge. Die simultane Visualisierung ist die identische Funktion. Ihre Ausgabe ist die Eingabe, die Laufzeit ist konstant. Die Reduzierter-Teilgraph-Visualisierung muss die gemeinsame reduzierte Teilstruktur aller Shapegraphen aus $\mathcal{M}$ bestimmen. Da es sich bei $\mathcal{M}$ um eine Klasse handelt, genügt es, die induzierte Teilstruktur eines Shapegraphen zu bestimmen. Die Berechnung der reduzierten Teilstrukturen muss bei der Klasseneinteilung sowieso geschehen, daher liegt die Information im Grunde genommen schon vor. Der Zeitaufwand der Methode kann also als konstant angesehen werden. Die Single-Structure-Visualisierung iteriert über die Eingabemenge, daher geht $|\mathcal{M}|$ linear in die Laufzeit ein. Die Einbettungs- und die V-Ecken-Visualisierung benötigen den Relationsgraphen. Dazu muss für jedes Paar von Shapegraphen aus $\mathcal{M}$ geprüft werden, ob sich der eine in den anderen einbetten lässt. Die Größe der Eingabemenge geht damit als $|\mathcal{M}|^2$ in die Laufzeit ein.

Die vorgestellten Methoden zur Klassenvisualisierung transformieren eine Klasse in eine in der Regel kleinere Menge von Graphen. Die in den Shapegraphen der Eingabemenge codierte Information über die konkreten Heapsituationen wird durch eine kleinere Menge ausgedrückt. Dass dies zu einer Vergröberung des Informationsgehaltes führen kann, beziehungsweise auch führt, ist daher nicht verwunderlich. Die simultane Visualisierung führt zu keinem Informationsverlust, da die Ausgabe mit der Eingabe übereinstimmt. Die Einbettungsvisualisierung führt (in der Regel) zu einer geringen Vergröberung: Einzig die Information über das definitive Auftreten von speziell(er)en Fällen geht verloren. Bei der V-Ecken-Visualisierung verhält es sich genauso, jedoch ist der Informationsverlust (etwas) höher. Auch bei der Single-

Methode	Vis.-Komplexität primär	sekundär	Informationsgehalt	subjektive Aussagekraft
simultane Vis.	∘	∘	∘	∘
Einbettungsvis.	+	∘	−	+
Reduzierter-Teilgraph-Vis.	+++	+	−−−	−/+
Single-Structure-Vis.	+++	−−	−(−)	−−/+
V-Ecken-Vis.	++	∘	−(−)	+

Abbildung 7.7: Vergleich der Methoden zur Klassenvisualisierung; + bedeutet Verbesserung, − bedeutet Verschlechterung im Vergleich zur Eingabe beziehungsweise zur simultanen Visualisierung (Einträge sind pro Spalte relativ zueinander)

Structure-Visualisierung tritt Informationsverlust ein. Es ist jedoch schwierig, diesen mit dem bei den beiden vorangegangenen Methoden entstehenden in Beziehung zu setzen. Die Reduzierter-Teilgraph-Visualisierung reduziert die Eingabeklasse auf die gemeinsame induzierte Teilstruktur. Jede Information über die Situation außerhalb dieser Teilstruktur geht verloren. Der Informationsverlust ist daher im Allgemeinen (sehr) hoch.

Zum Abschluss wollen wir uns der Aussagekraft der Graphen der Ausgabemenge zuwenden. Damit meinen wir die Eigenschaft, wie gut sich die Graphen „lesen“ lassen. Das beinhaltet hauptsächlich den Aspekt, wie effizient sich die in ihnen codierte Information über die konkret möglichen Heapsituationen aus ihnen extrahieren lässt. Die Beurteilung basiert nicht auf formalen Kriterien, sondern auf den Erfahrungen im Umgang mit den Methoden, was der Einschätzung einen subjektiven Charakter gibt. Auch bezieht sie sich auf die Visualisierung im Allgemeinen, spezielle Visualisierungsziele können durchaus zu anderen Präferenzen führen. Die simultane Visualisierung nimmt die neutrale Vergleichsposition ein. Die Einbettungs- und V-Ecken-Visualisierung verhalten sich ähnlich. Wir stufen die Aussagekraft in beiden Fällen als gut ein. Die Single-Structure-Visualisierung ist eine Form von Vereinigung. Der entstehende Graph ist häufig (extrem) unübersichtlich, was der Beispielgraph in Abbildung 7.4 gezeigt (oder zumindest angedeutet) hat. In speziellen Anwendungssituationen entsteht aber ein Graph mit deutlicher Aussage, wie der Graph in Abbildung 7.5 für die dort betrachtete Klasse gezeigt hat. Die Spannweite der Aussagekraft der resultierenden Graphen ist groß. Die Reduzierter-Teilgraph-Visualisierung liefert eine im Vergleich zur Eingabe sehr kompakte, übersichtliche und gut charakterisierende Ausgabe, was als positiv zu bewerten ist. Jedoch sind keine detaillierten Informationen in der Ausgabe präsent, sie gestattet deshalb kaum mehr als einen ersten Überblick über die Klasse. Dieser Aspekt ist, je nach dem worauf gerade Wert gelegt wird, als positiv oder negativ zu bewerten.

Um einen schnellen Überblick zu ermöglichen, haben wir die Ergebnisse der Vergleiche in Tabelle 7.7 zusammengefasst. Allerdings haben wir dort nicht alle Eigen-

schaften, hinsichtlich derer wir die Methoden miteinander vergleichen, aufgenommen. Ein Kreis bedeutet, dass sich die Ausgabe in der betrachteten Eigenschaft wie die ursprüngliche Eingabe verhält, ein Minus steht für eine Verringerung der Qualität der Eigenschaft und ein Plus steht für eine Erhöhung. Die Einträge sind pro Spalte relativ zueinander zu lesen. Zum Beispiel bedeuten zwei Pluszeichen eine stärkere Verbesserung als ein Pluszeichen; unterscheiden sich zwei Methoden bezüglich zweier Eigenschaften um jeweils ein Pluszeichen, dann bedeutet das aber nicht zwangsläufig, dass die Verbesserung bei beiden Eigenschaften gleich stark ausgeprägt ist.

Nachdem wir die charakteristischen Eigenschaften jeder Methode im Vergleich diskutiert haben, sollen einige Ratschläge und Empfehlungen zu ihrer Anwendung angeführt werden. Unsere Sicht ist dabei pragmatisch. Für einen ersten Überblick über eine Klasse oder für einen schnellen Vergleich mehrerer Klassen ist die Reduzierter-Teilgraph-Visualisierung die Methode der Wahl. Sie repräsentiert eine Klasse auf sehr kompakte Weise, allerdings enthält die Repräsentation wenig Information. Das andere Extrem bildet die simultane Visualisierung. Sie bietet die gesamte verfügbare Information und eignet sich daher für eine genaue Inspektion der Gegebenheiten innerhalb einer Klasse. Die Einbettungs- und die V-Ecken-Visualisierung sind dazwischen positioniert. V-Ecken-Graphen haben eine leicht andere Lesart als normale Shapegraphen, was eine Eingewöhnungszeit verlangt; dafür ist die zu primäre Visualisierungskomplexität in der Regel kleiner. Wir empfinden beide Methoden als „Allrounder“. Die Single-Structure-Visualisierung ist problematisch. Bei der Betrachtung ihrer Ausgabe drängte sich uns des Öfteren das Wort „konfus“ auf, andere Male lieferte sie recht brauchbare Graphen. Bei dieser Methode muss im Einzelfall beurteilt werden, inwieweit das Ergebnis im Hinblick auf die Visualisierungsziele befriedigt.

In diesem Abschnitt haben wir die Visualisierung von Klassen behandelt. Diese Arbeit beschäftigt sich mit Methoden, eine Shapeanalyseausgabe für die Visualisierung aufzubereiten. Unseren Ansatz zur Algorithmenvisualisierung verstehen wir als ein lernunterstützendes Mittel. Die Aufbereitung ist auf dieses Ziel ausgerichtet. Ein wesentliches Prinzip besteht in der Strukturierung der Shapeanalyseausgabe. Thema dieses Abschnitts ist eine weitere (Stufe der) Aufbereitung, die nach der eben angesprochenen Strukturierung, aber vor der eigentlichen Visualisierung, erfolgt. Konkrete Aspekte der Darstellung von Shapegraphen wie etwa das Zeichnen waren nicht Gegenstand der Erörterung. Im vorangegangenen Teilabschnitt wurden verschiedene Methoden behandelt. In diesem Teilabschnitt wurden die Methoden miteinander verglichen: Charakteristische Eigenschaften wurden benannt und bevorzugte Anwendungsmöglichkeiten hervorgehoben.

7.2 Ähnliche Ausführungspfade

In Kapitel 6 haben wir ein Ähnlichkeitskonzept für Shapegraphen entwickelt. Es gestattet eine Menge von Shapegraphen im Hinblick auf die Visualisierung zu strukturieren. In Verbindung mit den Klassenvisualisierungsmethoden des letzten Abschnitts dient dies zu Erhöhung der Aussagekraft der Visualisierung. Nun wird man sich bei der Visualisierung aber nicht damit zufrieden geben wollen, nur die Shapegraphen eines Programmpunktes visualisieren und betrachten zu wollen. Für ein tieferes Verständnis über den Algorithmus ist es auch erforderlich, die (abstrakte) Programmausführung zu studieren. Wir wollen also untersuchen, wie ein Shapegraph eine Folge von (TVLA-)Aktionen „durchläuft“ und wie er sich dabei verändert. So kommen wir unweigerlich auf Folgen von Shapegraphen, wobei die Abfolge der Programmpunkte durch den Kontrollflussgraphen bestimmt ist. Auch der Menge dieser Shapegraphsequenzen wollen wir eine zusätzliche Struktur geben, auch hier wollen wir einen Ähnlichkeitsbegriff einführen. Zu diesem Zweck übertragen wir in diesem Abschnitt das Ähnlichkeitskonzept für Shapegraphen auf Sequenzen von Shapegraphen.

Ähnlichkeit für Ausführungspfade

Für das Konzept der ähnlichen Ausführungspfade legen wir einen Ähnlichkeitsbegriff für Shapegraphen zugrunde, den wir auf Pfade im Transitionsgraphen übertragen werden. Ein alternativer Ansatz bestünde darin, sich nicht auf diese Basis zu stützen, und sogleich eine Vorstellung von Ähnlichkeit für Pfade zu entwickeln. Wir halten es für eine plausible Hypothese, dass die Ähnlichkeit zweier zusammengesetzter Dinge auch die Ähnlichkeit korrespondierender Komponenten implizieren sollte. Da wir unseren Ansatz zur Algorithmenvisualisierung in einem weiten Sinn als lernunterstützendes Mittel ansehen, ist eine solche Forderung sicher gerechtfertigt. Wenn wir nun zwei Pfade betrachten, die wir als ähnlich ansehen wollen, dann sollten also korrespondierende Shapegraphen einander ähnlich sein. Unter diesem Gesichtspunkt ist es daher keine Einschränkung, wenn wir mit einem Ähnlichkeitsbegriff für Shapegraphen beginnen.

Bisher ging es um Strukturierungsmethoden für Shapegraphenmengen. Daher war es bisher ausreichend, wenn wir die Shapeanalyseausgabe als Mengen von Shapegraphen vorausgesetzt haben. Für die Betrachtungen in diesem Abschnitt ist diese Art der Ausgabe nicht hinreichend, wir benötigen zusätzlich Kontrollflussinformationen in der Ausgabe. Daher setzen wir in diesem Abschnitt voraus, dass die Shapeanalyseausgabe als Transitionsgraph vorliegt. Seine Ecken sind abstrakte Zustände (n, S), bestehend aus einem Programmpunkt n und einem Shapegraphen S; seine Kanten sind mit TVLA-Aktionen beschriftet und korrespondieren mit denen des Kontrollflussgraphen; vergleiche Definition 3.7. In der einleitenden Motivation haben wir von durch den Kontrollflussgraphen bestimmten Folgen von Shapegraphen gesprochen. Das sind Pfade im Kontrollflussgraphen. Die Bezeichnung Pfad ist als

Synonym zu Kantenzug gemeint, die in ihnen vorkommenden Ecken dürfen sich wiederholen.

Wir sehen unseren Ansatz zur Algorithmenvisualisierung als ein lernunterstützendes Mittel. Deshalb erlauben wir keine beliebigen Programme. In Abschnitt 4.3 haben wir eine Normalform für TVLA-Kontrollflussgraphen definiert. So verlangen wir, dass TVLA-Programme deterministisch sind. Für je zwei Programmpunkte m und n gibt es daher höchstens eine TVLA-Aktion von m nach n. Im Transitionsgraphen sind daher alle Kanten von abstrakten Zuständen mit Programmpunkt m zu abstrakten Zuständen mit Programmpunkt n mit der gleichen TVLA-Aktion beschriftet. Des Weiteren folgt daraus auch, dass alle Kanten zwischen Shapegraphen zweier Klassen mit der gleichen TVLA-Aktion beschriftet sind. Aus diesem Grund ist es nicht erforderlich, dass wir jedes Mal explizit auf die Kantenbeschriftung eingehen. Wir wollen sie im Folgenden unberücksichtigt lassen, bei Bedarf kann sie aus dem Kontrollflussgraphen rekonstruiert werden. Einen Kantenzug notieren wir durch die Angabe seiner Ecken (abstrakten Zustände).

TVLA-Programme in Normalform setzen sich durch Konkatenation aus drei Arten von Konstrukten, die wir Blöcke nennen, zusammen. Zum Ersten handelt es sich um Aktionenblöcke. Ein Kantenzug im Transitionsgraphen, bei dem die Programmpunkte seiner abstrakten Zustände zu einem Aktionenblock gehören – wir sprechen abkürzend von einem Kantenzug im Aktionenblock –, ist zwangsläufig ein Weg. Zum Zweiten handelt es sich um Verzweigungsblöcke. Jeder Kantenzug in der (beziehungsweise durch die) Verzweigung ist ebenfalls ein Weg. Wenn wir allerdings zwei Wege durch dieselbe Verzweigung betrachten, dann können die besuchten Programmpunkte verschieden sein. Als Drittes handelt es sich um Schleifenblöcke. Ein Kantenzug durch eine Schleife enthält normalerweise Kreise. Beinhaltet der Schleifenrumpf eine Verzweigung als Teilblock, dann gehören bei zwei Kantenzügen durch die Schleife korrespondierende Ecken ebenfalls nicht notwendigerweise zum gleichen Programmpunkt.

Wir werden im Folgenden einen Ähnlichkeitsbegriff für Kantenzüge formalisieren, bei dem wir zugrunde legen, dass die Kantenzüge gleiche Programmpunkte durchlaufen. Wir setzen:

7.9 Definition. *Es sei $\mathcal{TG}$ der Transitionsgraph einer Shapeanalyseausgabe, ferner sei $\mathcal{D}$ ein Partitionsdefinierer und es gelte $k \geq 1$. Zwei Kantenzüge $\big((m_1, S_1), (m_2, S_2), \ldots, (m_k, S_k)\big)$ und $\big((n_1, T_1), (n_2, T_2), \ldots, (n_k, T_k)\big)$ in $\mathcal{TG}$ heißen* ähnlich bezüglich $\mathcal{D}$*, wenn für jedes i, $1 \leq i \leq k$ gilt:*

1. *$m_i = n_i$*
2. *$S_i \sim_{\mathcal{D}} T_i$, das heißt S_i und T_i sind ähnlich bezüglich $\mathcal{D}$.*

Ein triviales Beispiel ist das folgende: Sind S und T zwei bezüglich $\mathcal{D}$ ähnliche Shapegraphen desselben Programmpunktes n, dann sind (n, S) und (n, T) zwei bezüglich $\mathcal{D}$ ähnliche Kantenzüge der Länge 0. – Wir betrachten zur Verdeutlichung

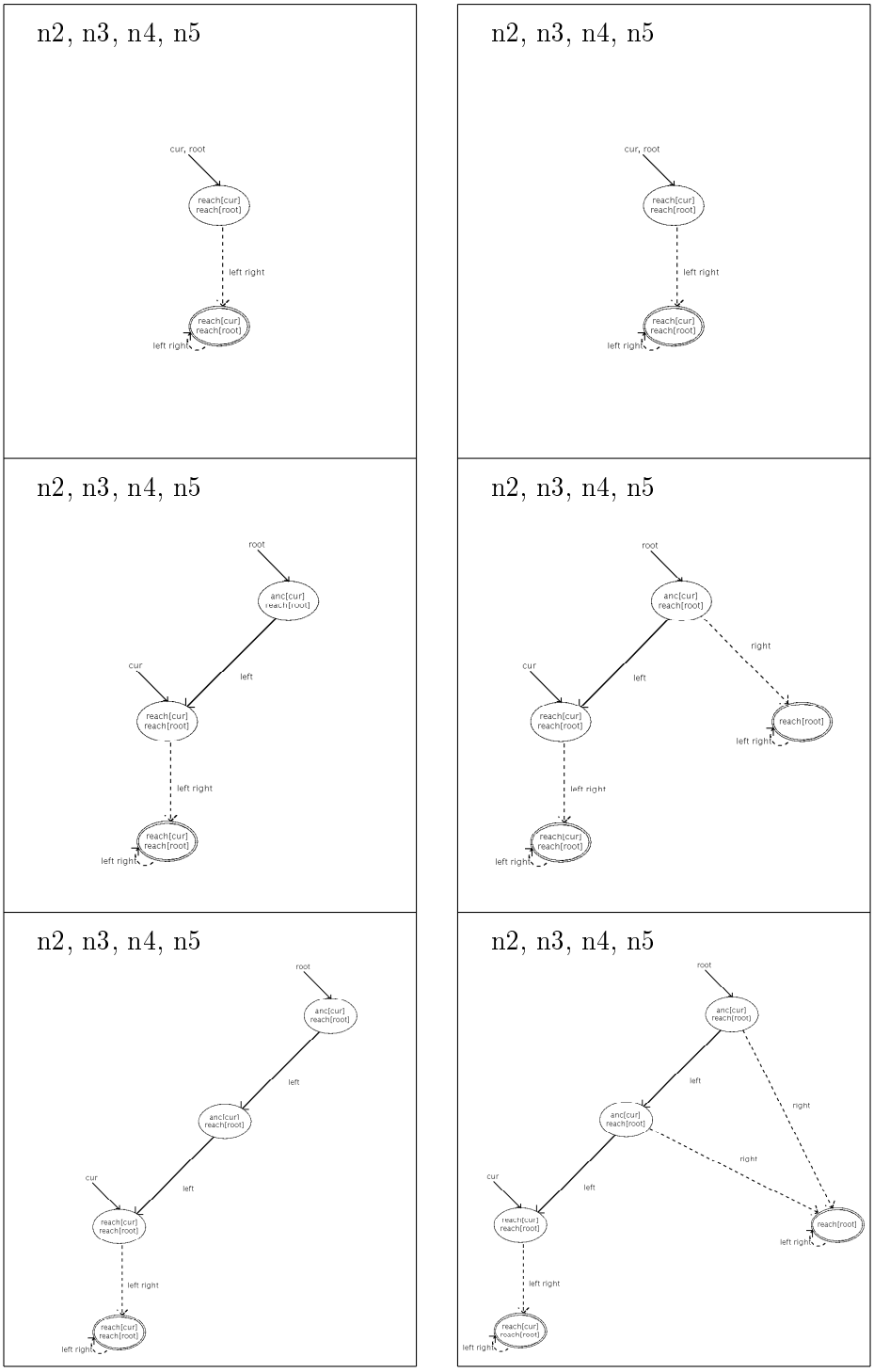

Abbildung 7.8: Zwei bzgl. der Suchwegstruktur ähnliche Ausführungspfade, Teil 1

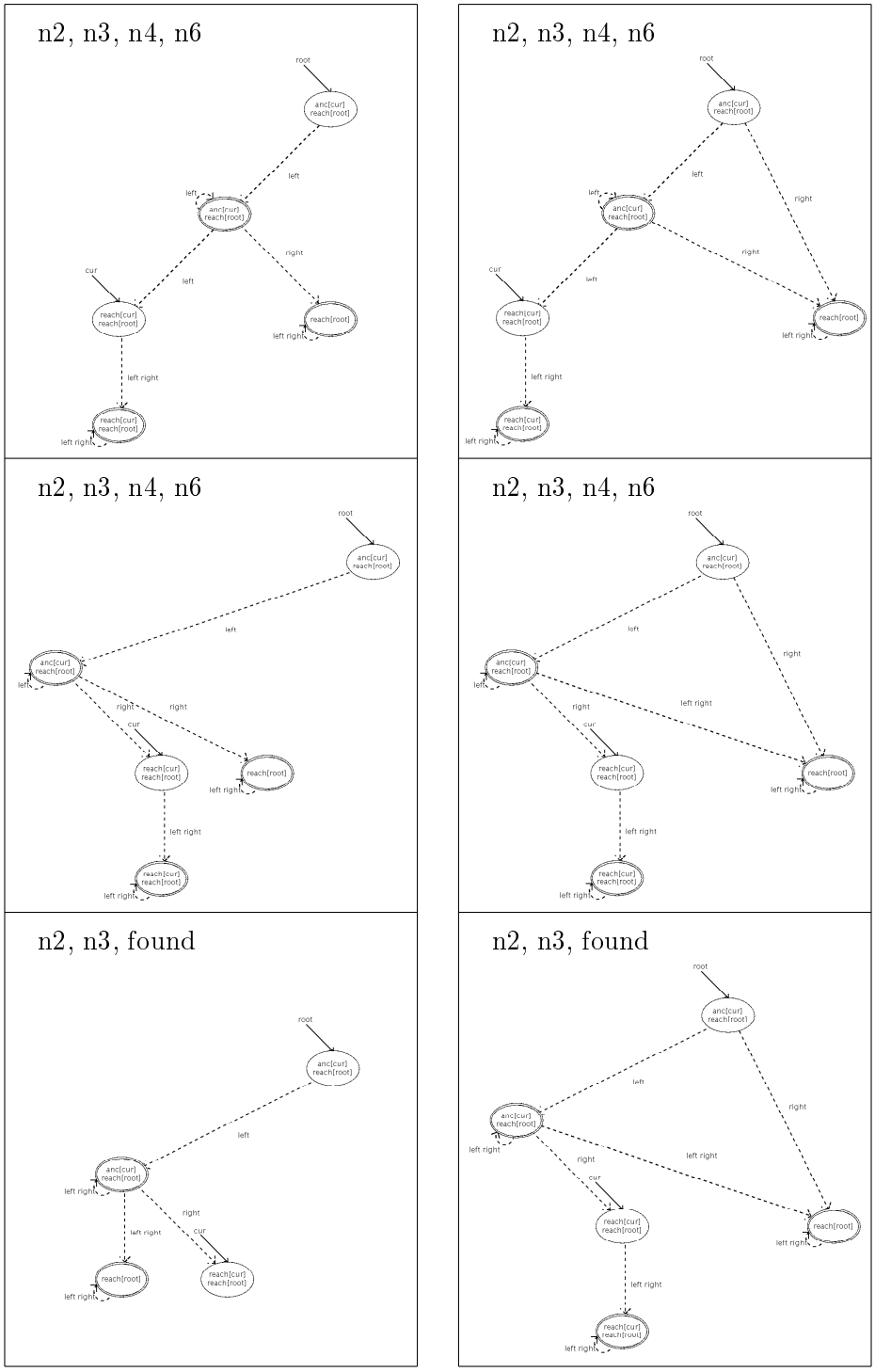

Abbildung 7.9: Zwei bzgl. der Suchwegstruktur ähnliche Ausführungspfade, Teil 2

ein umfassenderes Beispiel. Hierfür ziehen wir die Suche in einem Suchbaum heran, der Algorithmus ist in Abbildung 4.3 auf Seite 66 dargestellt. In den Abbildungen 7.8 und 7.9 sind zwei Kantenzüge durch den Transitionsgraphen dargestellt, jede Spalte zeigt einen Pfad. Sie beginnen beide im Programmpunkt n2, die erste TVLA-Aktion des Programms (eine Zeigerzuweisung) haben wir ausgelassen, und beide enden im terminalen Programmpunkt „found“. Jeder Kantenzug besteht aus 23 abstrakten Zuständen. Sind in einem Kantenzug die Shapegraphen zweier aufeinanderfolgender abstrakten Zustände gleich, so fassen wir sie in der Abbildung zusammen: Wir stellen nur einen Graphen dar und notieren dazu die aufeinanderfolgenden Programmpunkte. Wenn wir die Ecken beider Kantenzüge parallel durchlaufen, dann stimmen die Programmpunkte korrespondierender abstrakter Zustände überein. Die beiden Pfade entstehen, wenn wir bei der abstrakten Programmausführung dreimal hintereinander vom aktuellen Element zu seinem linken Kind verzweigen und danach zweimal zum rechten. Für unsere Ähnlichkeitsbetrachtung wählen wir den Partitionsdefinierer $\mathcal{D}$ so, dass Shapegraphen mit der gleichen Struktur des Suchweges einander ähnlich sind. Wie in Kapitel 6 verwenden wir dazu $\mathcal{D} = (\emptyset, \mathsf{cur}(v) \vee \mathsf{anc}[\mathsf{cur}](v), \{\mathsf{root}, \mathsf{cur}, \mathsf{reach}[\mathsf{root}], \mathsf{reach}[\mathsf{cur}], \mathsf{sm}, \mathsf{anc}[\mathsf{cur}], \mathsf{left}, \mathsf{right}\})$. Der Formelpartitionierer ist die leere Menge, er hat keinen Einfluss auf die Partitionierung. Die Eckenauswahlformel wählt, falls vorhanden, das aktuelle Element cur und seine Ahnen aus, das sind die Ecken des Suchweges. Ein eckenweiser (shapegraphenweiser) Vergleich der beiden Kantenzüge in den Abbildungen 7.8 und 7.9 zeigt, dass korrespondierende Ecken die gleiche Suchwegstruktur haben und damit bezüglich $\mathcal{D}$ ähnlich sind. Damit sind beide Kantenzüge bezüglich $\mathcal{D}$ ähnlich.

Wir setzen bei ähnlichen Pfaden voraus, dass sie paarweise die gleichen Programmpunkte durchlaufen. Oben sind wir auf die Struktur von TVLA-Programmen eingegangen. Bei einer Verzweigung kann man in Abhängigkeit vom Datenwert zu verschiedenen Programmpunkten verzweigen. Es könnte also der Wunsch entstehen, auch von ähnlichen Kantenzügen zu reden, wenn die Forderung nach paarweise gleichen Programmpunkten nicht erfüllt ist. Wenn man die bestehende Definition in dieser Richtung hin abzuschwächen möchte, bieten sich zwei Möglichkeiten an. Zum Ersten kann man in der Definition explizit formalisieren, dass, wenn es sich nicht um paarweise dieselben Programmpunkte handelt, es korrespondierende Programmpunkte in einer Verzweigung sein müssen. Ein solches Vorgehen erschwert in jedem Fall die Definition. Die zweite Möglichkeit besteht darin, überhaupt keine Forderung an die Programmpunkte zu stellen. Dann entsteht eine weite Fassung des Begriffs. Wir würden (beziehungsweise können) dann auch ähnliche Kantenzüge erhalten, bei denen korrespondierende abstrakte Zustände zu (völlig) verschiedenen Programmpunkten gehören. Diese Weitgefasstheit kann nicht dadurch gemildert werden, dass wir stattdessen zusätzlich fordern, dass bei ähnlichen Kantenzügen korrespondierende Kanten gleiche Kantenbeschriftungen tragen müssen. Der Sinn einer Verzweigung ist üblicherweise, dass verschiedene Aktionen ausgeführt werden. Wir brauchen nur den Suchalgorithmus für Suchbäume, vergleiche Abbildung 4.3 auf Seite 66, zu be-

trachten. Die Kanten, die vom Programmpunkt n4 zu den beiden Zweigen führen, haben die Beschriftung Less_Data_T(el, cur) oder Greater_Data_T(el, cur). Diese Variante der Verallgemeinerung des Ähnlichkeitsbegriffs ist insgesamt aber sicher zu bevorzugen, da sie einem fortgeschrittenen Anwender der Visualisierungswerkzeuge eine größere Flexibilität ermöglicht.

Wir haben in der Definition der Ähnlichkeit paarweise gleiche Programmpunkte gefordert. Das hat den Vorzug, dass ähnliche Kantenzüge auch intuitiv in dem Sinne ähnlich sind, dass auf die Daten(struktur) die gleichen Aktionen angewendet werden. Es kann bei speziellen Visualisierungszielen aber hilfreich sein, eine abgschwächte Variante zur Verfügung zu haben. Wenn wir in Definition 7.9 auf die Forderung nach paarweise gleichen Programmpunkten, also auf die Bedingung $m_i = n_i$, verzichten, dann nennen wir den so entstehenden Ähnlichkeitsbegriff *schwache Ähnlichkeit (bezüglich $\mathcal{D}$)*. Bei der ursprünglichen Ähnlichkeit können wir dann ergänzend von starker Ähnlichkeit sprechen. Diese Verallgemeinerung werden wir in der folgenden Diskussion nicht weiter verfolgen, wir verbleiben bei dem Ähnlichkeitsbegriff wie er in Definition 7.9 festgelegt ist.

In Satz 6.13 haben wir gezeigt, dass Ähnlichkeit für Shapegraphen eine Äquivalenzrelation ist. Diese Eigenschaft überträgt sich auf Kantenzüge:

7.10 Satz. *Die Ähnlichkeitsrelation für Kantenzüge ist eine Äquivalenzrelation.*

Der reduzierte Transitionsgraph

Bei der Entwicklung des Ähnlichkeitskonzepts für Shapegraphen haben wir in Kapitel 6 die von einem Partitionsdefinierer induzierte Partition einer Shapegraphenmenge eingeführt. Hier haben wir es formal nicht mehr mit Shapegaphen oder Shapegraphenmengen zu tun, sondern mit abstrakten Zuständen und Transitionsgraphen. Die Begrifflichkeit lässt sich leicht hierauf übertragen. Betrachten wir anstelle des Transitionsgraphen als Analyseausgabe eine Menge von abstrakten Zuständen, also gewissermaßen nur seine Ecken. Zwei abstrakte Zustände (m, S) und (n, T) seien genau dann ähnlich bezüglich eines Partitionsdefinierers $\mathcal{D}$, wenn $m = n$ und $S_i \sim_{\mathcal{D}} T_i$ gilt; sie müssen also zum gleichen Programmpunkt gehören. Für diese Ähnlichkeitsrelation auf abstrakten Zuständen verwenden wir ebenfalls die Notation $\sim_{\mathcal{D}}$. Die durch einen abstrakten Zustand (m, S) repräsentierte Klasse $[(m, S)]_{\sim_{\mathcal{D}}}$ ist die Menge aller zu (m, S) ähnlichen abstrakten Zustände. Die induzierte Partition ist dann die Menge aller dieser Klassen.

Wir sehen ähnliche abstrakte Zustände als nicht unterscheidbar an. Daher können wir die abstrakten Zustände jeder Klasse zu einer Einheit zusammenfassen. Wenn wir dies im Transitionsgraphen vornehmen, dann erhalten wir:

7.11 Definition. *Es sei $\mathcal{TG}$ der Transitionsgraph einer Analyseausgabe, des Weiteren sei $\mathcal{D}$ ein Partitionsdefinierer. Der* bezüglich $\mathcal{D}$ reduzierte Transitionsgraph

$\mathcal{RTG}_\mathcal{D}$ entsteht aus $\mathcal{TG}$, indem bezüglich $\sim_\mathcal{D}$ ähnliche abstrakte Zustände verschmolzen werden.

Im vorangegangenen Abschnitt, als wir uns mit der Visualisierung von Klassen befassten, haben wir uns auch dem Aspekt der Visualisierungskomplexität von Klassen gewidmet. Dabei haben wir zwischen der Anzahl, der primären Visualisierungskomplexität, und der Größe, der sekundären Visualisierungskomplexität, unterschieden. Diese Begriffe wollen wir auch allgemein zur Charakterisierung der Ausgabe verwenden. Hier haben wir es aber nicht mehr mit Shapegraphenmengen, sondern mit Transitionsgraphen zu tun. Daher bedürfen die Begriffe einer Klärung. Wenn wir die Anzahl der Shapegraphen als Anzahl abstrakter Zustände verstehen, dann wäre die Eckenanzahl des Transitionsgraphen die geeignete Entsprechung. Wir wollen darunter jedoch die strukturelle Größe der Ausgabe verstehen, wobei ein Shapegraph als Einheit betrachtet wird. Die primäre Visualisierungskomplexität ist die Größe des Transitionsgraphen. Unter der sekundären Visualisierungskomplexität verstehen wir die maximale Größe einer seiner Ecken, also die maximale Größe eines Shapegraphen. Allgemein wollen wir unter Visualisierungskomplexität das Folgende verstehen:

7.12 Definition. *Die* primäre Visualisierungskomplexität *eines Graphen ist die Anzahl seiner Ecken und Kanten, wobei wir seine gleichartigen Bestandteile als Einheiten auffassen. Die* sekundäre Visualisierungskomplexität *eines Graphen ist das Maximum der Größen seiner Bestandteile.*

Der Übergang von einem Transitionsgraphen zu einem reduzierten Transitionsgraphen führt (bei nichttrivialen Partitionsdefinierern) zu einer Verringerung der primären Visualisierungskomplexität (der Analyseausgabe). Die Ecken sind nun aber keine Shapegraphen mehr, sondern Klassen, die sekundäre Visualisierungskomplexität ist damit angestiegen. Primäre Visualisierungskomplexität wurde in sekundäre transformiert. Die Methoden zur Visualisierung einer Klasse in Abschnitt 7.1 erlauben es, die Visualisierungskomplexität der Klassen zu verringern. Klasseneinteilung und Klassenvisualisierung führen in Kombination zu einer Reduzierung der Visualisierungskomplexität im Vergleich zur Shapeanalyseausgabe.

Wir sind von der ursprünglichen Analyseausgabe, dem Transitionsgraphen, zu einer abgeleiteten Struktur, dem reduzierten Transitionsgraphen, übergegangen. Dabei haben wir eine Reduzierung der Visualisierungskomplexität im Sinne. Wir müssen noch untersuchen, wie sich die Informationen, die im Transitionsgraphen enthalten sind, zu denen verhalten, die im reduzierten Transitionsgraphen enthalten sind. Für das Folgende bezeichne $\mathcal{TG}$ den Transitionsgraphen einer Analyseausgabe. Ein Partitionsdefinierer sei festgelegt, so dass wir von Ähnlichkeit sprechen können. Der bezüglich dieses Partitionsdefinierers reduzierte Transitionsgraph sei mit $\mathcal{RTG}$ bezeichnet.

Jedem Kantenzug in $\mathcal{TG}$ entspricht ein Kantenzug in $\mathcal{RTG}$, den wir auch als reduzierten Kantenzug bezeichnen. Ähnlichkeit für Kantenzüge ist eine Äquivalenzrelation. Ähnliche Kantenzügen haben also denselben reduzierten Pfad, nichtähnliche Kantenzüge haben verschiedene. In diesem Sinne sind Kantenzüge in $\mathcal{RTG}$ Ähnlichkeitsklassen von Kantenzügen in $\mathcal{TG}$. Betrachten wir nun einen (reduzierten) Kantenzug in $\mathcal{RTG}$, etwa $K' = (\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3)$. Was können wir über die Kantenzüge in $\mathcal{TG}$ sagen, die K' als reduzierten Pfad haben? Wegen $[\mathcal{K}_1, \mathcal{K}_2\rangle \in \mathcal{RTG}$ existieren in $\mathcal{TG}$ Ecken $(n_1, S_1) \in \mathcal{K}_1$ und $(n_2, S_2) \in \mathcal{K}_2$ mit $[(n_1, S_1), (n_2, S_2)\rangle \in \mathcal{TG}$. Wegen $[\mathcal{K}_2, \mathcal{K}_3\rangle \in \mathcal{RTG}$ existieren in $\mathcal{TG}$ ebenfalls Ecken $(n_2, S_2') \in \mathcal{K}_2$ und $(n_3, S_3) \in \mathcal{K}_3$ mit $[(n_2, S_2'), (n_3, S_3)\rangle \in \mathcal{TG}$. Es ist aber denkbar und möglich, dass alle in Frage kommenden S_2 von allen in Frage kommenden S_2' verschieden sind. Folglich braucht es in $\mathcal{TG}$ überhaupt keinen Kantenzug zu geben, dessen reduzierter Pfad K' ist. Eine solche Situation ist unbefriedigend, wenn wir für die Visualisierung den reduzierten Transitionsgraphen anstelle des eigentlichen nutzen wollen.

Zu jeder Kante des reduzierten Transitionsgraphen existiert wenigstens eine Kante im Transitionsgraphen, die zu ihr reduziert. Aber wenn es für zwei reduzierte Kantenzüge aus $\mathcal{RTG}$ „Urbilder“ in $\mathcal{TG}$ gibt, dann braucht dies für deren Konkatenation nicht mehr zu gelten. Im Folgenden werden wir Situationen auszeichnen, in denen auch die Konkatenation zweier Kantenzüge aus $\mathcal{RTG}$ ein „Urbild“ besitzt. Sie sind im Hinblick auf die Visualisierung interessant. Die Ursachen des Problems können an zwei Stellen liegen: im Transitionsgraphen und im Partitionsdefinierer. Wir werden Konstellationen von Transitionsgraph(en) und Partitionsdefinierern auszeichnen, die sich im Hinblick auf Konkatenation gutartig verhalten. Dazu werden wir fordern, dass sich Kantenzüge verlängern lassen. Wir setzen:

7.13 Definition. *Es sei $\mathcal{TG}$ ein Transitionsgraph, $\mathcal{D}$ ein Partitionsdefinierer und $\mathcal{RTG}$ der bezüglich $\mathcal{D}$ reduzierte Transitionsgraph. Existiert zu jedem Kantenzug K' in $\mathcal{RTG}$ ein Kantenzug in $\mathcal{TG}$, der zu K' reduziert, dann nennen wir $\mathcal{D}$* wohlig *für $\mathcal{TG}$. In diesem Fall bezeichnen wir die Ähnlichkeitsrelation ebenfalls als wohlig und sprechen von wohlähnlich.*

Es sei $\mathcal{TG}$ ein Transitionsgraph und $\mathcal{D}$ ein für $\mathcal{TG}$ wohliger Partitionsdefinierer. Der bezüglich $\mathcal{D}$ reduzierte Transitionsgraph sei $\mathcal{RTG}$. Wir gehen so vor, dass wir ein Kriterium ableiten, welches den folgenden Schluss erlaubt: Zu jedem Kantenzug $K' = (\mathcal{K}_1, \ldots, \mathcal{K}_k)$ in $\mathcal{RTG}$ und zu jedem $(n_k, S_k) \in \mathcal{K}_k$ existiert ein Kantenzug in $\mathcal{TG}$, der zu K' reduziert und dessen Endecke (n_k, S_k) ist. Das wollen wir mittels vollständiger Induktion zeigen. – Zuerst betrachten wir den Fall, dass $K' = (\mathcal{K}_1, \mathcal{K}_2)$ ein Kantenzug der Länge 1 ist. Zu jeder Kante in $\mathcal{RTG}$, also auch zu K', existiert per Definition wenigstens eine Kante in $\mathcal{TG}$, die zu ihr reduziert. (Ihre Anfangsecke liegt in $\mathcal{K}_1$ und ihre Endecke in $\mathcal{K}_2$.) Um die gewünschte Aussage sicherzustellen, müssen wir folgende Bedingung fordern: Existiert eine Kante $[(m, S), (n, T)\rangle$ im Transitionsgraphen, wobei $(m, S) \in \mathcal{K}_1$ und $(n, T) \in \mathcal{K}_2$ ist, dann existiert zu jedem $(n, T) \in \mathcal{K}_2$ ein $(m, S) \in \mathcal{K}_1$, so dass $[(m, S), (n, T)\rangle \in \mathcal{TG}$

gilt. – Jetzt sei $K' = (\mathcal{K}_1, \ldots, \mathcal{K}_k)$ ein Kantenzug in $\mathcal{RTG}$ der Länge $k \geq 2$, ferner sei $(n_k, S_k) \in \mathcal{K}_k$. Wir zerteilen K' in zwei Kantenzüge $K'_1 = (\mathcal{K}_1, \mathcal{K}_2)$ und $K'_2 = (\mathcal{K}_2, \ldots, \mathcal{K}_k)$. Nach Induktionsvoraussetzung existiert in $\mathcal{TG}$ ein Kantenzug $K_2 = \big((n_2, S_2), \ldots, (n_k, S_k)\big)$ mit der geforderten Endecke, der zu K'_2 reduziert. Mit demselben Argument wie beim Induktionsanfang, oder unter erneuter Bezugnahme auf die Induktionsvoraussetzung, sehen wir, dass auch eine Ecke $(n_1, S_1) \in \mathcal{K}_1$ mit $[(n_1, S_1), (n_2, S_2)\rangle \in \mathcal{TG}$ existiert. Die Konkatenation der beiden Kantenzüge ergibt den gesuchten Pfad.

Bei dieser Überlegung haben wir (Teil-)Kantenzüge nach vorne hin verlängert. Wir haben einen Kantenzug K' in zwei Teilkantenzüge K'_1 und K'_2 zerlegt. Die Induktionsvoraussetzung sicherte, dass zu ihnen korrespondierende Kantenzüge K_1 und K_2 in $\mathcal{TG}$ existieren. Sie sicherte ebenfalls, dass es einen zu K_1 ähnlichen Kantenzug gibt, dessen Endecke gleich der Anfangsecke von K_2 ist. Wir können aber auch andersherum vorgehen. Wir beginnen mit K_1 und wählen einen zu K_2 ähnlichen Kantenzug, dessen Anfangsecke gleich der Endecke von K_1 ist. Dazu müssen wir folgendes sicherstellen: Ist $K' = (\mathcal{K}_1, \ldots, \mathcal{K}_k)$ ein Kantenzug in $\mathcal{RTG}$, dann existiert zu jedem abstrakten Zustand $(m, S) \in \mathcal{K}_1$ ein Kantenzug in $\mathcal{TG}$, der zu K' reduziert und dessen Anfangsecke (m, S) ist. Die dazu nötige Bedingung ist „symmetrisch“ zu der obigen.

Diese Überlegungen haben gezeigt:

7.14 Satz. *Es sei $\mathcal{TG}$ ein Transitionsgraph, $\mathcal{D}$ ein Partitionsdefinierer und $\mathcal{RTG}$ der bezüglich $\mathcal{D}$ reduzierte Transitionsgraph. Jede der folgenden Bedingungen impliziert, dass $\mathcal{D}$ wohlig für $\mathcal{TG}$ ist:*

1. *Existieren $(m, S) \in \mathcal{K}_1$ und $(n, T) \in \mathcal{K}_2$ mit $[(m, S), (n, T)\rangle \in \mathcal{TG}$, dann gilt $\forall (m, S) \in \mathcal{K}_1\ \exists (n, T) \in \mathcal{K}_2\ [(m, S), (n, T)\rangle \in \mathcal{TG}$.*
2. *Existieren $(m, S) \in \mathcal{K}_1$ und $(n, T) \in \mathcal{K}_2$ mit $[(m, S), (n, T)\rangle \in \mathcal{TG}$, dann gilt $\forall (n, T) \in \mathcal{K}_2\ \exists (m, S) \in \mathcal{K}_1\ [(m, S), (n, T)\rangle \in \mathcal{TG}$.*

Wenn $\mathcal{TG}$ ein Transitionsgraph und $\mathcal{D}$ ein für $\mathcal{TG}$ wohliger Partitionsdefinierer ist, dann stellt der bezüglich $\mathcal{D}$ reduzierte Transitionsgraph eine brauchbare Alternativstruktur für die Visualisierung dar und kann gut anstelle von $\mathcal{TG}$ verwendet werden. – Wenn wir den Algorithmus zur Suche eines Elementes in einem Suchbaum verwenden, eine Standardshapeanalyse mit der in Abschnitt 4.1 vorgestellten Analysespezifikation durchführen und den Partitionsdefinierer so wählen, dass Shapegraphen mit der gleichen Struktur des Suchweges einander ähnlich sind, dann erhalten wir eine wohlige Ähnlichkeitsrelation.

Die Beachtung von Ähnlichkeit(en) ist bei Lernvorgängen oft vorteilhaft: Hat man eine der zueinander ähnlichen Situationen verstanden, dann erschließen sich die anderen in der Regel leicht. Im vorangegangenen Kapitel haben wir deshalb ein Ähnlichkeitskonzept für Shapegraphen eingeführt und untersucht. In diesem Abschnitt haben wir dieses Ähnlichkeitskonzept von Shapegraphen auf Ausführungspfade, auf

Kantenzüge im Transitionsgraphen übertragen. Des Weiteren sind wir zu einem reduzierten Transitionsgraphen übergegangen. Wir haben Bedingungen herausgearbeitet und formalisiert, bei denen sich in der Visualisierung der reduzierte Transitionsgraph gut anstelle des ursprünglichen verwendet lässt.

7.3 Navigation im Transitionsgraphen

Bei der Visualisierung einer Shapeanalyseausgabe wird man sich nicht mit der Betrachtung einzelner Shapegraphen zufrieden geben wollen, man wird auch Kantenzügen im Transitionsgraphen folgen wollen. In diesem Abschnitt entwickeln wir Charakterisierungen für Shapegraphen, welche Hilfestellung bei der Navigation im Transitionsgraphen (und im reduzierten Transitionsgraphen) bieten. Will man einem Kantenzug im Transitionsgraphen folgen, dann hängt die Auswahl des Startprogrammpunktes und des Startshapegraphen auf der einen Seite und des jeweiligen Nachfolgeshapegraphen auf der anderen Seite von vielen Gesichtspunkten ab, die von den spezifischen Charakteristiken des Algorithmus bis zu den speziellen Zielen der aktuellen Visualisierungssitzung, wobei hier auch der Kenntnisstand des Benutzers eingeht, reichen. Die Charakterisierungen, die wir hier behandeln, elaborieren den Unterschied zwischen Shapegraphen, die spezielle(re) Heapsituationen beschreiben, und Shapegraphen, die allgemeine(re) Heapsituationen beschreiben. Wir entwickeln zwei Charakterisierungen. Die erste setzt methodisch auf dem Konzept der Einbettung auf. Die zweite Charakterisierung zeichnet Shapegraphen aus, die sich unter einer Folge von TVLA-Aktionen „nicht verändern“, die unter einer Folge von TVLA-Aktionen „fix“ bleiben. Letztere wurde für Schleifen in [Johannes u. a. 2005] eingeführt.

7.3.1 Allgemeine versus spezielle Shapegraphen

Dieser Abschnitt handelt allgemein von Charakterisierungen von Shapegraphen, die während einer Visualisierungssitzung die Navigation im Transitionsgraphen erleichtern. Dabei liegt unser Augenmerk auf Aspekten der Visualisierung von Kantenzügen (des Transitionsgraphen). Als Ausgangssituation setzen wir voraus, dass die Programmpunkte der Pfade gegeben sind, dass sie zuvor unter Verwendung des Kontrollflussgraphen ausgewählt worden sind. Uns geht es hier um eine Hilfestellung bei der Auswahl der Shapegraphen an den einzelnen Programmpunkten. Hierzu gehört als Erstes die Wahl eines Startshapegraphen für den Kantenzug. Ein abstrakter Zustand im Transitionsgraphen kann einen oder mehrere Nachfolger haben. Im ersten Fall kann in der Visualisierung sofort mit dem einzigen Nachfolger fortgefahren werden. Existieren jedoch mehrere Nachfolger, so gabelt sich der Pfad, und wir müssen einen geeigneten Nachfolger auswählen.

Zum einen kann die Gabelung durch den Kontrollfluss induziert werden. (Wir setzen voraus, dass TVLA-Programme in Normalform vorliegen, so wie sie in Abschnitt 4.3 beschrieben ist.) Bei Aktionenblöcken kann keine Gabelung auftreten, wohl aber bei Verzweigungen und Schleifen. Bei Verzweigungen führt der Verzweigungsblock zu der Gabelung, bei Schleifen besteht die Möglichkeit, sie entweder zu verlassen oder vom Ausgangsblock des Schleifenrahmens zur Eingangsecke seines Eingangsblocks zurückzuspringen. Diese kontrollflussbedingte Gabelung ist für unser Anliegen unproblematisch. In jedem dieser Fälle gilt, dass die Nachfolgeecken (Nachbarecken im Transitionsgraphen) zu verschiedenen Programmpunkten gehören. Wir haben als Ausgangssituation vorausgesetzt, dass die Programmpunkte, die der Kantenzug besuchen soll, im Voraus festliegen. Daher ist die Nachfolgeecke bei kontrollflussbedingter Gabelung (in der Regel[4]) eindeutig.

Zum anderen kann die Gabelung auch durch die Shapeanalyse selbst bedingt sein. Wir illustrieren dieses Phänomen anhand eines Beispiels, wobei wir uns der üblichen Analysespezifikation für Bäume bedienen. Wir betrachten die TVLA-Aktion Get_Sel_T, die im Suchalgorithmus auftritt, vergleiche Abschnitt 4.1.2. Wir wenden auf den Shapegraphen S aus Abbildung 7.10 die Aktion Get_Sel_T(cur, cur, left) an, sie entspricht der Zuweisung cur := cur.left. Es resultieren fünf Shapegraphen, nämlich $T_1, T_2, \ldots, T_5$ aus Abbildung 7.10. Das Phänomen, dass bei Anwendung einer TVLA-Aktion auf einen Shapegraphen mehr als ein Shapegraph resultieren kann, ist nicht baumspezifisch. Es tritt zum Beispiel ebenfalls bei der Aktion Get_Next_L bei Listen auf. Es ist also nicht immer korrekt, sich eine TVLA-Aktion als eine Funktion vorzustellen, die einen Shapegraphen in einen anderen transformiert; das Ergebnis kann eine (endliche) Menge von Shapegraphen sein.

Der Grund für dieses Verhalten liegt in der Fokussierung, die Bestandteil jeder Anwendung einer TVLA-Aktion ist, vergleiche hierzu Abschnitt 3.7 oder [Lev-Ami u. Sagiv 2000; Sagiv u. a. 2002]. Jeder Aktion kann eine Menge von Fokus-Formeln mitgegeben werden. Der Fokus-Algorithmus transformiert den Shapegraphen, auf den die Aktion angewendet wird, in eine endliche Menge von Shapegraphen, so dass die folgenden zwei Bedingungen erfüllt sind: 1. Sowohl der Ausgangsshapegraph als auch die Shapegraphen der Ausgabemenge beschreiben dieselben konkreten Strukturen. 2. Jede Fokusformel wertet für jeden Shapegraphen der Ausgabemenge immer (für jede Variablenbelegung) zu einem definiten Wahrheitswert aus.

Bei der Aktion Get_Sel_T benötigt man eine Fokusformel. Werfen wir zur Verdeutlichung einen Blick auf den Shapegraphen S in Abbildung 7.10. Er besteht aus

4. Eine kontrollflussbedingte Gabelung ist (bei deterministischen Programmen) an eine Verzweigung gebunden. Üblicherweise werden TVLA-Aktionen für Vergleiche als Filter realisiert: Shapegraphen, die die Filterbedingung(en) erfüllen, dürfen passieren, die anderen nicht. Bei Anwendung einer solchen Filteraktion existiert zu jedem Shapegrahen höchstens ein Nachfolger pro Programmpunkt.

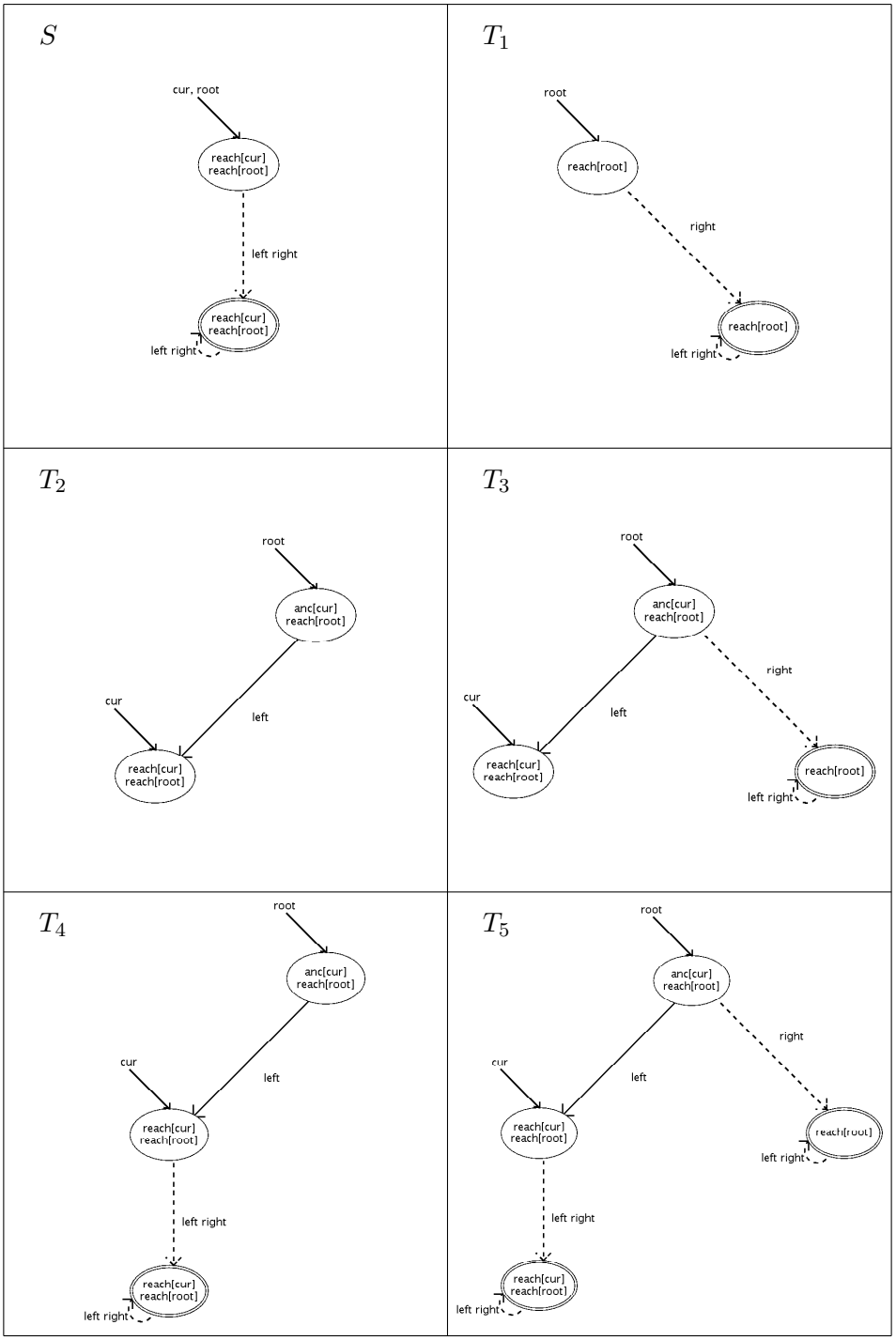

Abbildung 7.10: Anwendung von Get_Sel_T(cur, cur, left) auf S ergibt die fünf Shapegraphen $T_1, T_2, \ldots, T_5$

zwei Ecken: der Wurzel, die auch gleichzeitig die aktuelle Ecke ist, und einer Summary-Ecke, die den Rest des Baumes repräsentiert. Um cur auf ein Kind der Wurzel, beispielsweise auf das linke, zu setzen, muss dieses erst als Individuum aus der Summary-Ecke „herausmaterialisieren“ werden. (Prädikate für Zeigervariablen erhalten in der Analysespezifikation das Attribut „unique pointer“ und können deshalb nur für Individuenecken wahr sein.) Die Summary-Ecke muss durch (mehrere) Ecken ersetzt werden, die jeweils spezifischere Baumteile repräsentieren. Die Fokusformel, die bei der Anwendung der Aktion Get_Sel_T(cur, cur, left) verwendet wird, ist $\exists v_1, v_2 \colon \mathsf{cur}(v_1) \wedge \mathsf{left}(v_1, v_2) \wedge \mathsf{downStar}(v_2, v)$, wobei v eine freie Variable bezeichnet. Für den Shapegraphen S ist die Teilformel $\mathsf{cur}(v_1)$ genau dann wahr, wenn v_1 die Wurzel ist. Die Teilformel $\mathsf{left}(v_1, v_2)$ erzwingt, dass es entweder ein linkes Kind der Wurzel gibt oder nicht. Die Teilformel $\mathsf{downStar}(v_2, v)$ erzwingt eine Aufspaltung in einen linken und einen rechten Teilbaum der Wurzel. Die entsprechenden Baumteile können jeweils vorhanden sein oder nicht, wobei einzelne Kombinationen nicht möglich sind. Auf diese Weise resultieren die fünf Shapegraphen $T_1, T_2, \ldots, T_5$ in Abbildung 7.10.

Im ersten Abschnitt dieses Kapitels haben wir uns mit einer weiteren Form der Aufbereitung von Shapegraphenmengen (Klassen) beschäftigt. Eine der behandelten Methoden ist die Einbettungsvisualisierung. Sie wählt aus der Eingabemenge einige Shapegraphen aus, die sie als repräsentativ für die Klasse erklärt. Dabei handelt sich um die bezüglich der Einbettungsrelation maximalen Elemente. Ist ein Shapegraph S in einen Shapegraph T einbettbar, dann ist die Menge der von S beschriebenen konkreten Strukturen eine Teilmenge, der von T beschriebenen. In diesem Sinne können wir T als allgemeiner und S als spezieller interpretieren. Für beliebig gewählte Shapegraphen (eines Programmpunktes) verhält es sich aber häufig so, dass keine Einbettung vorliegt. Denken wir an den Suchalgorithmus für binäre Suchbäume und betrachten beispielsweise die (Teil-)Menge der Shapegraphen aus den Abbildungen 7.8, 7.9 und 7.10. Diese Shapegraphen treten alle am Programmpunkt n2, dem Schleifeneingangspunkt, auf. Aber nur wenige Shapegraphenpaare stehen bezüglich Einbettung in Relation. Die unmittelbare Nutzung der Einbettungsrelation als Hilfestellung bei der Shapegraphenauswahl während der Visualisierung von Pfaden im Transitionsgraphen ist daher in der Praxis nur bedingt hilfreich.

Im vorangegangenen Kapitel 6 haben wir ein Ähnlichkeitskonzept für Shapegraphen eingeführt. Shapegraphen, die bezüglich (durch einen Partitionsdefinierer) spezifizierbarer Merkmale nicht unterscheidbar sind, sehen wir als ähnlich an. Uns beschäftigt die Frage, ob ein Shapegraph T als allgemeiner als ein Shapegraph S bezeichnet werden kann, und wir wollen dies mittels einer Einbettungsbeziehung ausdrücken. Bei Verwendung des Ähnlichkeitskonzepts unterscheiden wir nicht zwischen ähnlichen Shapegraphen. Es sei S' ein zu S und T' ein zu T ähnlicher Shapegraph. Anstelle die Einbettungsbeziehung zwischen S und T zu betrachten, können wir stattdessen auch die Einbettungsbeziehung zwischen S' und T' betrachten. Wir könnten also sagen, dass T allgemeiner als S sei, wenn es einen zu S ähnlichen

Shapegraphen S' und einen zu T ähnlichen Shapegraphen T' (in der aktuell betrachteten Shapegraphenmenge) gibt, so dass S' in T' einbettbar ist. Allerdings würden wir damit die Existenz der Shapegraphen S' und T' fordern (müssen). Wir werden demgegenüber eine schwächere Bedingung verwenden.

Der Ähnlichkeitsbegriff beruht auf einem Partitionsdefinierer. Er besteht aus zwei Komponenten: einem Formelpartitionierer und einem Teilstrukturdefinierer. Bei der Motivation des Begriffs „allgemeiner“ haben wir uns von der Idee der Einbettung leiten lassen. Die Einbettungsrelation ist aber nur auf Shapegraphenmengen definiert. Bei der Formalisierung des Begriffs müssen wir auch den Formelpartitionierer mit einbeziehen. Wir setzen:

7.15 Definition. *Es sei $\mathcal{D} = (F, \varphi(v), P)$ ein Partitionsdefinierer. Ein Shapegraph T heiße* allgemeiner bezüglich $\mathcal{D}$ *als ein Shapegraph S, wenn:*

1. $\forall f \in F\colon W_S(f) \sqsubseteq W_T(f)$
2. $S_{(\varphi(v),P)} \sqsubseteq T_{(\varphi(v),P)}$

Ist T allgemeiner bezüglich $\mathcal{D}$ als S, dann schreiben wir $S \preccurlyeq_{\mathcal{D}} T$. Alternativ nennen wir S dann auch spezieller bezüglich $\mathcal{D}$ als T.

Man beachte, dass das Symbol $\sqsubseteq$ in der ersten Bedingung der Definition die Informationsordung auf der Wahrheitswertemenge $\{0, 1, \frac{1}{2}\}$ und in der zweiten Bedingung die Einbettungsrelation für Shapegraphen bezeichnet. In der Definition fordern wir für die Wahrheitswertetupel, dass das erste bezüglich der Informationsordung „allgemeiner“ als das zweite ist. Für die Shapegraphen fordern wir, dass sich die induzierten Teilstrukturen ineinander einbetten lassen, dass also die eine induzierte Teilstruktur „allgemeiner“ als die andere ist.

Der obige Begriff des Allgemeinerseins kann als eine abgeschwächte Form von Ähnlichkeit gesehen werden. Aus der Definition folgt:

7.16 Satz. *Es sei $\mathcal{D}$ ein Formelpartitionierer, ferner seien S und T zwei bezüglich $\mathcal{D}$ ähnliche Shapegraphen. Dann gilt $S \preccurlyeq_{\mathcal{D}} T$.*

Wir untersuchen einige Beispiele, wobei wir jedes Mal die Shapeanalyseausgabe für den Suchalgorithmus zugrunde legen. Als Erstes betrachten wir den Partitionsdefinierer $\mathcal{D} = (\emptyset, 1, P)$, wobei P die Menge aller Prädikate der Analysespezifikation bezeichne. Da der Formelpartitionierer die leere Menge ist, hat er keine Auswirkung auf die Partitionierung. Der Teilstrukturdefinierer bestimmt jeweils den ganzen Shapegraphen als induzierte Teilstruktur. Jede Klasse der induzierten Partition der Shapegraphenmenge eines Programmpunktes enthält nur einen einzigen Shapegraphen. (Wir setzen voraus, dass die Shapegraphen jedes Programmpunktes paarweise nichtisomorph sind.) In diesem Fall stimmt die Relation $\preccurlyeq_{\mathcal{D}}$ mit der Einbettungsrelation $\sqsubseteq$ überein. – Als Zweites wählen wir den Partitionsdefinierer so, dass die Partitionierung hinsichtlich der Struktur des Suchweges erfolgt. Wir

betrachten die beiden Shapegraphen T_2 und T_3 aus Abbildung 7.10, die beide am Schleifeneingangspunkt n2 vorkommen. In beiden Shapegraphen hat der Weg von der Wurzel zur aktuellen Ecke dieselbe Struktur, das heißt beide Shapegraphen sind in derselben Klasse. Alle Formeln des Formelpartitionierers werten für beiden Shapegraphen zum selben Wahrheitswert aus, außerdem stimmen ihre reduzierten Teilstrukturen überein. Damit gilt $T_2 \preccurlyeq_{\mathcal{D}} T_3$. Das gleiche Argument zeigt $T_4 \preccurlyeq_{\mathcal{D}} T_5$. – Als Drittes betrachten wir den unteren Shapegraphen in der rechten Spalte von Abbildung 7.8, er sei mit S_1 bezeichnet, und den oberen Shapegraphen in der linken Spalte von Abbildung 7.9, der S_2 heiße. Wie man sich leicht überzeugt gilt $S_1 \preccurlyeq_{\mathcal{D}} S_2$.

Die Allgemeiner-Relation aus Definition 7.15 ist zunächst nur für Shapegraphen definiert. Im Folgenden haben wir es jedoch mit abstrakten Zuständen zu tun. Die Allgemeiner-Relation lässt sich problemlos auf sie übertragen. Wir achten zusätzlich nur darauf, dass die Programmpunkte übereinstimmen. Wir definieren: Ein abstrakter Zustand (n, T) heiße allgemeiner bezüglich eines Partitionsdefinierers $\mathcal{D}$ als ein abstrakter Zustand (m, S), wenn $m = n$ und $S \preccurlyeq_{\mathcal{D}} T$ gilt. Wir schreiben in diesem Fall $(m, S) \preccurlyeq_{\mathcal{D}} (n, T)$.

Es sei ein Partitionsdefinierer $\mathcal{D} = (F, \varphi(v), P)$ gegeben. Wir partitionieren die Eckenmenge des Transitionsgraphen, also die Menge der abstrakten Zustände der Analyseausgabe, bezüglich $\mathcal{D}$ und betrachten zwei der entstandenen Klassen. Diese seien mit $\mathcal{K}_1$ und $\mathcal{K}_2$ bezeichnet, außerdem sei $(m, S) \in \mathcal{K}_1$ und $(n, T) \in \mathcal{K}_2$. Wir betrachten den Fall $(m, S) \preccurlyeq_{\mathcal{D}} (n, T)$. Dann gilt erstens $m = n$, zweitens ist für jede Formel f des Formelpartitionierers $W_S(f) \sqsubseteq W_T(f)$ und drittens gilt $S_{(\varphi(v),P)} \sqsubseteq T_{(\varphi(v),P)}$. Ist $(m, S') \in \mathcal{K}_1$ ein beliebiger (anderer) abstrakter Zustand aus $\mathcal{K}_1$, dann gilt per Definition $\forall f \in F\colon W_{S'}(f) = W_S(f)$ und $S'_{(\varphi(v),P)} \simeq S_{(\varphi(v),P)}$. Wir setzen immer voraussetzen, dass alle pro Programmpunkt von der Shapeanalyse berechneten Shapegraphen nichtisomorph sind; wenn wir uns zudem die Shapegraphen mit kanonischem Universum denken, dann gilt sogar $S'_{(\varphi(v),P)} = S_{(\varphi(v),P)}$. Ist $(n, T') \in \mathcal{K}_2$ ein beliebiger (anderer) abstrakter Zustand aus $\mathcal{K}_2$, dann gilt analog $\forall f \in F\colon W_{T'}(f) = W_T(f)$ und $T'_{(\varphi(v),P)} \simeq T_{(\varphi(v),P)}$, beziehungsweise $T'_{(\varphi(v),P)} = T_{(\varphi(v),P)}$. Wir setzen alles zusammen: Für jede Formel $f \in F$ gilt $W_{S'}(f) = W_S(f) \sqsubseteq W_T(f) = W_{T'}(f)$, und für die induzierten Teilstrukturen ist $S'_{(\varphi(v),P)} = S_{(\varphi(v),P)} \sqsubseteq T_{(\varphi(v),P)} = T'_{(\varphi(v),P)}$. Folglich gilt $S' \preccurlyeq_{\mathcal{D}} T'$.

Existiert ein Paar abstrakter Zustände aus $\mathcal{K}_1 \times \mathcal{K}_2$, das in Relation bezüglich der Allgemeiner-Relation steht, dann besteht die Relationsbeziehung für alle Paare aus $\mathcal{K}_1 \times \mathcal{K}_2$. Existiert umgekehrt ein Paar abstrakter Zustände aus $\mathcal{K}_1 \times \mathcal{K}_2$, für welches die Relationsbeziehung nicht besteht, dann kann sie für kein Paar bestehen. Die Relation $\preccurlyeq_{\mathcal{D}}$ ist also unabhängig von den Repräsentanten der Klassen. Wir können $\preccurlyeq_{\mathcal{D}}$ als eine Relation auf den durch $\mathcal{D}$ induzierten Klassen der Eckenmenge des Transitionsgraphen interpretieren. Damit ist $\preccurlyeq_{\mathcal{D}}$ eine (wohldefinierte) Relation auf den Ecken des bezüglich $\mathcal{D}$ reduzieren Transitionsgraphen.

Fassen wir nun die Allgemeiner-Relation als Relation auf der Menge der Klassen auf. Sie ist wegen Satz 7.16 reflexiv. Sie ist außerdem transitiv, was aus der Transitivität der Informationsordnung und aus Satz 7.1 folgt. Die Informationsordnung ist antisymmetrisch. Ist der Transitionsgraph durch eine Standardanalyse entstanden, sind also die Shapegraphen jedes Programmpunktes paarweise nichtisomorph und sind für jeden Shapegraphen die Wahrheitswerte aller Abstraktionsprädikate definit, dann ist nach Satz 7.1 auch die Einbettungsrelation antisymmetrisch, was dann die Antisymmetrie der Allgemeiner-Relation impliziert. Damit haben wir gezeigt:

7.17 Satz. *Es sei $\mathcal{M}$ eine Menge von abstrakten Zuständen, des Weiteren sei $\mathcal{D}$ ein Partitionsdefinierer. Die Relation $\preccurlyeq_{\mathcal{D}}$ ist eine reflexive und transitive Relation auf der Partition $\mathcal{M}/_{\sim_{\mathcal{D}}}$. Sind für jeden Programmpunkt die Shapegraphen der abstrakten Zustände paarweise nichtisomorph und haben in jedem Shapegraphen alle Abstraktionsprädikate definite Wahrheitswerte, so ist $\preccurlyeq_{\mathcal{D}}$ eine Ordnungsrelation auf $\mathcal{M}/_{\sim_{\mathcal{D}}}$.*

In diesem Abschnitt beschäftigen wir uns mit Techniken zur Hilfestellung bei der Navigation im Transitionsgraphen. Konkret geht es um die Verlängerung von Kantenzügen, um die Auswahl von Nachfolgeshapegraphen. In diesem Teilabschnitt haben wir die intuitive Vorstellung von einem allgemeineren versus einem spezielleren Shapegraphen mathematisch umgesetzt. Wir haben uns dabei methodisch auf die Einbettungsrelation gestützt. Sie alleine erweist sich aber nicht als ausreichend. Daher haben wir das Konzept der Einbettung mit dem Ähnlichkeitsbegriff kombiniert, woraus die Allgemeiner-Relation $\preccurlyeq_{\mathcal{D}}$ resultiert. Sie ist unabhängig von den speziellen Elementen der Klassen, sie ist eine Relation auf der durch den Partitionsdefinierer induzierten Partition der Menge der abstrakten Zustände. Damit erlaubt sie auch eine Klassifikation von Klassen hinsichtlich Allgemeinheit.

Die Allgemeiner-Relation erlaubt ein zweistufiges Vorgehen bei der Auswahl eines Nachfolgeshapegraphen. Als Erstes werden die Klassen betrachtet, in denen die Nachfolger liegen. Die Relation $\preccurlyeq_{\mathcal{D}}$ ordnet die Klassen nach Allgemeinheit. Unter diesem Gesichtspunkt wird zuerst eine Klasse ausgewählt, zum Beispiel eine bezüglich $\preccurlyeq_{\mathcal{D}}$ maximale. Existiert nur ein Nachfolgeshapegraph in der ausgewählten Klasse, dann ist die Bestimmung des Nachfolgers schon abgeschlossen. Existieren mehrere, dann gehen wir als Zweites zu der Klasse über und wählen hier einen Nachfolger. Wir können beispielsweise die Einbettungs- oder V-Einbettungsrelation verwenden und einen bezüglich einer dieser Relationen maximalen Shapegraphen wählen. Auch die Methoden, die wir im nächsten Teilabschnitt behandeln, können hierbei von Nutzen sein.

Betrachten wir als Beispiel das Szenario in Abbildung 7.10. Unser aktueller Shapegraph ist S, und seine Nachfolger sind $T_1, T_2, \ldots, T_5$. Den Partitionsdefinierer wählen wir so, dass Shapegraphen mit derselben Struktur des Suchweges ähnlich sind. Die Nachfolgegraphen liegen dann in zwei Klassen: Der Shapegraph T_1 liegt in einer, die

Shapegraphen T_2, T_3, T_4, T_5 in der zweiten. Zuerst müssen wir uns für eine dieser Klasse entscheiden. Da es in T_1 kein aktuelles Element (mehr) gibt, entscheiden wir uns, diese Klasse zu verwerfen. Wir gehen zur zweiten Klasse über. In dieser ist T_5 das bezüglich der V-Einbettung maximale Element, und wir könnten uns deshalb entschließen, es auszuwählen. In den Shapegraphen T_2 und T_3 ist das aktuelle Element cur ein Blatt des Baumes. Wenn wir weitere Iterationen der Suchschleife zu betrachten wünschen, wären sie eine schlechte Wahl. (Diese Art von Kriterien werden wir im folgenden Teilabschnitt behandeln.)

7.3.2 Semistabile Shapegraphen

Will man Kantenzügen im Transitionsgraphen folgen, dann steht man sowohl vor der Aufgabe, einen geeigneten Startshapegraphen als auch unter den Nachfolgeshapegraphen einen passenden auszuwählen. Die speziellen Ziele der aktuellen Visualisierungssitzung können die Anzahl geeigneter Nachfolger einschränken, die Auswahlaufgabe als solche besteht jedoch weiterhin. In diesem Abschnitt ist es unser Anliegen, Shapegraphen hinsichtlich ihrer Allgemeinheit zu klassifizieren und dadurch Hilfestellung bei der Auswahl zu bieten. Im Gegensatz zum vorangegangenen Teilabschnitt beziehen wir uns nicht auf den Ähnlichkeitsbegriff. Der Ansatz dieses Teilabschnitts basiert auf der Frage, inwieweit sich Shapegraphen bei Anwendung von TVLA-Aktionen verändern.

Semistabilität

Zur Motivation des Ansatzes betrachten wir eine Schleife in einem TVLA-Programm. Hier ist das Phänomen, um das es uns geht, deutlicher ausgeprägt als bei Aktionenblöcken und Verzweigungen. Wir betrachten als Beispiel die Suchschleife im Suchalgorithmus für Suchbäume, er ist in Abbildung 4.3 auf Seite 66 dargestellt. Wir berechnen den Transitionsgraphen und untersuchen einen Kantenzug, der die (Programmpunkte der) Schleife mehrfach durchläuft. Wir verwenden hierzu den Pfad, der sich aus den abstrakten Zuständen der rechten Spalten der beiden Abbildungen 7.8 und 7.9 (Seite 173f.) zusammensetzt. Es genügt, wenn wir uns darauf beschränken, jeweils die Situation am Programmpunkt n2 zu begutachten. Wenn wir die Shapegraphen der Reihe nach betrachten, dann stellen wir fest, dass sich am Anfang des Pfades ein Shapegraph recht stark von seinem Nachfolger unterscheidet, am Ende ist der Unterschied deutlich geringer. Der letzte Shapegraph, bezeichnen wir ihn als T_{ru}, unterscheidet sich von dem vorletzten nur im Wert des Prädikats left für die Summary-Ecke, die das Innere des Suchpfades darstellt. Führen wir eine weitere Schleifeniteration durch, wobei wir vom aktuellen Element cur wieder zu seinem rechten Kind verzweigen, dann kommt unter den Nachfolgegraphen auch T_{ru} selbst vor. Wenn wir den Pfad auf diese Weise fortsetzen, dann bleibt der Shapegraph also unverändert. Ein Shapegraph beschreibt eine Menge von konkreten Strukturen. Wenn sich ein Shapegraph bei Anwendung einer Aktion nicht

ändert, dann bedeutet dies, dass sich sowohl die abstrakte Situation als auch die durch sie beschriebenen konkreten Situationen in einem gewissen Sinne „stabil“ verhalten.

Wenn sich ein Shapegraph bei der Anwendung einer TVLA-Aktion stark ändert, dann bedeutet das nicht zwangsläufig, dass sich die von ihm beschriebenen konkreten Strukturen stark ändern müssen. Beim Suchalgorithmus durchlaufen wir nur einzelne Heapzellen, der Heap selbst, also die Zeigerstruktur und die Datenwerte, bleibt unverändert. Was sich ändert ist der „Blick“ auf den Heap. Bei den ersten Schleifeniterationen ist die Ausgabe der Shapeanalyse präzise genug, um festzustellen, dass das aktuelle Element ein Kind oder ein Enkel der Wurzel ist. In folgenden Iterationen liegt keine Information mehr über die Tiefe des aktuellen Elementes vor. Die Shapegraphen der ersten Schleifeniterationen beschreiben (in diesem Sinne) spezifischere Heapsituationen, die der späteren unspezifischere.

Das Phänomen, dass sich Shapegraphen in den ersten Iterationen von denen in späteren unterscheiden, korrespondiert zum Verhalten bei konkreten Strukturen. Die Situationen während der ersten Schleifeniterationen sind oft noch durch die Schleifeninitialisierung und den Iterationsbeginns geprägt, sie sind häufig spezifischer Natur. Wenn man aus den in allen Iterationen auftretenden Situationen welche auswählen soll, die man intuitiv als allgemeine Schleifeniterationen bezeichnen würde, dann gehören die Situationen der Anfangsiterationen normalerweise nicht dazu. Man wird eigentlich immer Situationen aus späteren Schleifeniterationen benennen. Jedoch wird man wohl auch keine Situation der letzten Iterationen nominieren, beim Beenden der Schleife liegen häufig wieder spezifischere Situationen vor.

Wenn wir uns von dem obigen Beispiel führen lassen, dann zeichnen sich unspezifische(re) Shapegraphen dadurch aus, dass sie sich unter einer Sequenz von TVLA-Aktionen nicht oder nur wenig verändern, die „sehr unspezifischen“ sollten sich dann gar nicht mehr ändern, sie sollten fix bleiben. Diese Sicht ist so nicht (ganz) korrekt. Wenn wir mit einem Shapegraphen S starten und einem Pfad durch die Schleife folgen, dann kann eine shapeanalysebedingte Gabelung stattfinden. Beim Suchalgorithmus ist dies bei den meistens Shapegraphen der Fall. Wenn wir eine komplette Schleifeniteration betrachten, dann wird es häufig mehrere „Nachfolgeshapegraphen“ für den Startshapegraphen S geben. Zu diesen kann S gehören oder nicht. Natürlich ist es auch möglich, dass S der einzige Nachfolgeshapegraph nach einer Schleifeniteration ist. Wir werden sagen, dass unspezifische Shapegraphen solche sind, die in einer Schleifenoperation in sich überführt werden können.

Bis zu einem gewissen Grad können wir das Begriffspaar spezifisch versus unspezifisch mit dem Begriffspaar speziell versus allgemein identifizieren. Auf jeden Fall sind die Begriffspaare korreliert. Shapegraphen, die auf die Art unspezifisch sind, wie wir sie in der obigen Diskussion motiviert haben, werden wir in der folgenden

formalen Entwicklung semistabil nennen. Das Konzept der semistabilen Shapegraphen wurden in [Johannes u. a. 2005] eingeführt und speziell für Schleifen untersucht.

Schreiten wir zur formalen Entwicklung des Begriffs des semistabilen Shapegraphen. TVLA-Programme sind aus drei Arten von Konstrukten zusammengesetzt: Aktionenblöcken, Verzweigungen und Schleifen, vergleiche Abschnitt 4.3. Der zu einem Programm mittels Shapeanalyse berechnete Transitionsgraph sei im Folgenden jeweils mit $\mathcal{TG}$ bezeichnet. – Als Erstes betrachten wir den Fall, dass das TVLA-Programm ein Aktionenblock ist, dieser sei (B, e, a). Wir wählen unter den Programmpunkten von B zwei aus, nennen wir sie s und z. Uns interessieren Kantenzüge von s nach z durch den Block B, sie sollen also nur Programmpunkte aus B besuchen. Wir reduzieren den Transitionsgraphen auf die Programmpunkte des Blocks: Der Graph $\mathcal{TG}(B)$ entstehe aus $\mathcal{TG}$, indem alle Ecken (abstrakten Zustände) samt inzidenter Kanten entfernt werden, deren Programmpunkt nicht zum Block B gehören. Da es sich bei B um einen Aktionenblock handelt, ist jeder Kantenzug von s nach z in $\mathcal{TG}(B)$ ein Weg. Die Abfolge der besuchten Programmpunkte ist durch das TVLA-Programm bestimmt. Kommt z vor s, dann gibt es keinen Kantenzug von s nach z. Der Fall $s = z$ ist graphentheoretisch trivial, denn von jeder Ecke existiert ein Weg der Länge null zu sich. Kantenzüge der Länge null wollen wir aber nicht gelten lassen, den Grund werden wir bei der Behandlung von Schleifen sehen.

Als Zweites betrachten wir den Fall, dass das TVLA-Programm eine einfache Verzweigung ist, sie sei durch den Block (B, e, a) gegeben. (Einfach bedeutet, dass die Äste der Verzweigung Aktionenblöcke sind.) Hier ist die Situation ähnlich wie beim Aktionenblock, Jeder Kantenzug in $\mathcal{TG}(B)$ zwischen zwei Ecken ist ein Weg. Jedoch brauchen bei zwei Wegen deren besuchte Programmpunkte nicht dieselben sein. Ein weiterer Punkt, den wir im Auge behalten wollen, ist, dass Verzweigungen terminale Programmpunkte enthalten dürfen. Sie dienen zur Simulation von RETURN-Anweisungen in Funktionen. Deshalb können wir uns bei unserer Betrachtung nicht darauf beschränken, nur Kantenzüge von der Eingangsecke e eines Blocks zu seiner Ausgangsecke a zu betrachten. – Als Drittes folgt der Fall, dass das TVLA-Programm eine einfache Schleife ist. (Der Schleifenrumpf besteht aus einem Aktionenblock.) Durch den Rücksprung vom Ausgangsblock des Schleifenrahmens zu seinem Eingangsblock entsteht ein Kreis im Kontrollflussgraphen, der normalerweise Kreise im Transitionsgraphen impliziert. In den folgenden Erörterungen werden wir einzelne Iterationen der Schleife gesondert betrachten, dazu ist es ebenfalls erforderlich, dass wir uns nicht auf Kantenzüge von der Eingangsecke einer Schleife zu seiner Ausgangsecke beschränken. – Als Viertes folgt der Fall eines (allgemeinen) TVLA-Programms. Strukturell entsteht nichts Neues: Es können terminale Programmpunkte im Inneren existieren, und Kantenzüge im Transitionsgraphen können Kreise enthalten.

Wir haben die verschiedenen Arten von Blöcken betrachtet, aus denen ein TVLA-

Programm besteht und auf die entstehenden Kantenzüge geschlossen. Wir erklären jetzt, was wir unter einem semistabilen Shapegraphen verstehen wollen:

7.18 Definition. *Es sei ein TVLA-Programms gegeben. Der mittels Shapeanalyse berechnete Transitionsgraph sei $\mathcal{TG}$. Es sei (B, e, a) ein (Teil-)Block des Programms, ferner seien s und z zwei Programmpunkte von B. Wir nennen einen Shapegraphen S* semistabil bezüglich $B_{s\to z}$, *wenn gilt:*

1. *Der abstrakte Zustand (s, S) ist eine Ecke in $\mathcal{TG}$.*
2. *Der abstrakte Zustand (z, S) ist eine Ecke in $\mathcal{TG}$.*
3. *Es existiert in dem auf den Block B reduzierten Transitionsgraphen $\mathcal{TG}(B)$ ein Kantenzug der Länge größer oder gleich eins von (s, S) nach (z, S).*

Ist S semistabil bezüglich $B_{e\to a}$, dann heiße S semistabil bezüglich B.

Normalerweise werden TVLA-Aktionen für Vergleiche als Filter realisiert: Shapegraphen, die die Filterbedingung(en) erfüllen, dürfen passieren, die anderen nicht. Besitzt eine solche Filter-Aktion keine Fokusformel, dann führt die Anwendung der Aktion zu keiner Veränderung des Shapegraphen. Wenn wir einen zweieckigen Block betrachten, dessen beide Ecken mit einer Kante verbunden sind, die mit einer solchen Filteraktion beschriftet ist, dann ist jeder passierende Shapegraph semistabil bezüglich dieses Blocks. – Betrachten wir im Suchalgorithmus für Binärbäume den Block B für die Suchschleife, der aus den Programmpunkten n2 bis n6 besteht. Der Shapegraph T_{ru}, der rechts unten in Abbildung 7.9 (Seite 174) dargestellt ist, tritt am Programmpunkt n2 auf. Wenn wir von der aktuellen Ecke cur zu ihrem rechten Kind verzweigen, dann ist einer der entstehenden Shapegraphen T_{ru}. Folglich ist T_{ru} semistabil bezüglich $B_{n2\to n2}$ (also sozusagen bezüglich einer Schleifeniteration). Ebenso sehen wir, dass beispielsweise auch die oberen beiden Shapegraphen in Abbildung 7.9 semistabil bezüglich $B_{n2\to n2}$ sind. Demgegenüber ist von den sechs Shapegraphen in Abbildung 7.8 keiner bezüglich $B_{n2\to n2}$ semistabil.

In der Definition 7.18 des Begriffs semistabil haben wir die Existenz von Kantenzügen gefordert. Wir können stattdessen auch verlangen, dass jeder Kantenzug in dem Block von (s, S) zu einem abstrakten Zustand mit Programmpunkt z in (z, S) endet. Dies entspricht der Vorstellung aus der einleitenden Motivation, dass ein Shapegraph unter einer Folge von TVLA-Aktionen in sich transformiert wird, dass er ein Fix-Shapegraph für diese Aktionenfolge ist. Solche Shapegraphen nennen wir *stabil bezüglich* $B_{s\to z}$. Wie wir oben dargelegt haben, ist die definierende Bedingung sehr restriktiv, sie wird eher selten erfüllt. Daher werden wir den Begriff des stabilen Shapgegraphen nicht weiter verfolgen.

Ist ein Shapegraph S semistabil bezüglich B und ist B' ein Teilblock von B, dann ist S nicht notwendigerweise semistabil bezüglich B'. Wir zeigen dies durch Angabe eines Gegenbeispiels. Wir legen die Spezifikation für Binärbäume zugrunde. Der Block B_l sei ein Aktionenblock und bestehe aus zwei Ecken. Sie seien durch eine

Kante verbunden, die mit der TVLA-Aktion Get_Sel_T(cur, cur, left) beschriftet sei. Der Block B_r sei wie B_l, jedoch sei die TVLA-Aktion hier Get_Sel_T(cur, cur, right). Wir betrachten wieder den Shapegraphen T_{ru}, der rechts unten in Abbildung 7.9 dargestellt ist. Er ist semistabil bezüglich B_r. Aber er ist nicht semistabil bezüglich B_l: Nach Ausführung der Aktion Get_Sel_T(cur, cur, left) gelangt man zur aktuellen Ecke cur von der Wurzel kommend mit einer left-Kante. Der Block B entstehe durch Konkatenation von B_l und B_r. Der Shapegraph T_{ru} ist semistabil bezüglich B: Wenn wir in T_{ru} erst zum linken und danach zum rechten Kind verzweigen, dann resultiert auch der Shapegraph T_{ru}. Aber wie schon festgestellt, ist T_{ru} nicht semistabil bezüglich des Teilblocks B_l.

Der Schleifentransitionsgraph

Der Begriff eines bezüglich $B_{s\to z}$ semistabilen Shapegraphen kann auf beliebige (Teil-)Programme angewendet werden. Es ist aus zwei Gründen jedoch nützlich, Schleifen diesbezüglich genauer zu studieren. Ein Kantenzug durch eine Schleife kann beliebig viele Iterationen enthalten, und es ist oft wünschenswert, jede Iteration einzeln zu betrachten. Zum Zweiten ist es oft wünschenswert, Kantenzüge in der Schleife zu verfolgen, die keine Schleifeniterationen sind. Wir erlauben in Verzweigungen terminale Programmpunkte. Enthält der Schleifenrumpf solche Verzweigungen, dann liegen in der Schleife terminale Programmpunkte vor. Im Suchalgorithmus für Suchbäume ist der Programmpunkt „found“ ein solcher. Das Aufbrechen der Schleife ermöglicht auch eine explizite Repräsentation solcher Pfade zu terminalen Ecken.

Um eine bessere Untersuchung einer Schleife zu ermöglichen, leiten wir aus dem Transitionsgraphen für die Schleife einen gerichteten Graphen als Hilfsstruktur ab. Die Idee ist folgende: Die Eckenmenge des Hilfsgraphen wird die Menge $\mathcal{M}$ der abstrakten Zustände eines geeignet gewählten Programmpunktes im Eingangsblock des Schleifenrahmens vermehrt um zwei weitere Ecken E (Eingang) und A (Ausgang). Wir ziehen eine gerichtete Kante von einem abstrakten Zustand aus $\mathcal{M}$ zu einem anderen, wenn dieser von jenem in genau einer Schleifeniteration erreichbar ist. Alle von außen direkt erreichbaren abstrakten Zustände werden Nachfolger von E. Alle abstrakten Zustände, aus denen heraus die Schleife in der nächsten Iteration beendet werden kann, erhalten eine Kante zu A.

Wir formalisieren dies. Es sei (B, e, a) eine Schleife, wir können sie als eigenständiges TVLA-Programm ansehen oder als Teil eines anderen Programms auffassen. Der mittels Shapeanalyse berechnete Transitionsgraph sei $\mathcal{TG}$. Mit $\mathcal{TG}(B)$ bezeichnen wir wieder den auf die zu B gehörenden Programmpunkte eingeschränkten Transitionsgraphen. Der geeignete Programmpunkt, von dem wir die abstrakten Zustände verwenden, ist derjenige, zu dem der Rücksprung vom Ausgangsblock des Schleifenrahmens erfolgt. Der Eingangsblock des Schleifenrahmens ist eine Konkatenation eines Aktionenblockes, er dient zur Simulation von Schleifeninitialisierungen, und eines Verzweigungsblocks; vergleiche Abschnitt 4.3, etwa Abbildung 4.8 auf Seite 82.

Die geeignete Ecke ist damit $e_v^{(E)}$. Beim Suchalgorithmus für Binärbäume benötigt die Suchschleife keine Initialisierung, der Eingangsblock besteht nur aus dem Verweigungsblock. Der Schleifenrücksprung erfolgt in so einem Fall zum Schleifeneingangspunkt. In diesem Beispiel ist also n2 als Programmpunkt zu verwenden, was wir in den Beispielen bisher auch immer getan haben.

Es bezeichne $\mathcal{M}$ die Menge der abstrakten Zustände des Transitionsgraphen mit Programmpunkt $m = e_v^{(E)}$. Die Eckenmenge des Hilfsgraphen sei $\mathcal{M} \cup \{E, A\}$. Seine Kantenmenge setze sich aus drei Arten von Kanten zusammen:

- Für zwei Ecken (m, S) und (m, T) aus $\mathcal{M}$ existiere genau dann eine Kante $[(m, S), (m, T)\rangle$, wenn es in $\mathcal{TG}(B)$ einen Kantenzug von (m, S) nach (m, T) gibt, der mindestens die Länge 1 hat[5] und dessen innere Ecken nicht zum Programmpunkt m gehören. (Das heißt, der abstrakte Zustand (m, T) sei von (m, S) in genau einem Schleifendurchlauf erreichbar.)
- Für jeden abstrakten Zustand $(m, S) \in \mathcal{M}$ der von der Eingangsecke e der Schleife direkt[6] erreichbar ist, existiere eine Kante $[E, (m, S)\rangle$.
- Für jeden abstrakten Zustand $(m, S) \in \mathcal{M}$ existiere eine Kante $[(m, S), A\rangle$, wenn von (m, S) aus in der folgenden Iteration der Schleifenausgangsprogrammpunkt a oder ein eventuell vorhandener terminaler (schleifenverlassender) Programmpunkt erreichbar ist, wenn es also einen Kantenzug von (m, S) zu einem abstrakten Zustand (n, T) in $\mathcal{TG}(B)$ gibt, so dass $n = a$ oder n ein terminaler Programmpunkt ist und alle inneren Ecken von $e_v^{(E)}$ verschieden sind.

Wir fassen zusammen und benennen den Hilfsgraphen:

7.19 Definition. *Es sei B eine Schleife eines TVLA-Programms und $\mathcal{TG}$ der für das Programm berechnete Transitionsgraph. Der auf die vorstehende Weise entstehende (gerichtete) Graph heiße der* Schleifentransitionsgraph *für B bezüglich $\mathcal{TG}$.*

Man beachte, dass der Schleifentransitionsgraph in der Regel Schlingen besitzt. Im Gegensatz zum Transitionsgraphen enthält der Schleifentransitionsgraph keine Beschriftungen. – Eine Schleife B hat genau eine Ausgangsecke, und sie kann im Inneren terminale Programmpunkte enthalten. Es gibt also verschiedene Arten, die Schleife zu beenden. In der obenstehenden Definition haben wir zwischen diesen verschiedenen Arten nicht unterschieden. Möchte man jedoch mehrere Arten des Schleifenendes unterscheiden, dann kann man anstelle der einen Ecke A für jede der

5. Eine einzelne Ecke ist ein Kantenzug der Länge null. Auf diese Art entstehende Schlingen sind unerwünscht.
6. Direkt erreichbar soll hier ohne Schleifeniteration bedeuten. Formal: Es existiert in $\mathcal{TG}(B)$ ein Kantenzug von der Eingangsecke e zu $e_v^{(E)}$, der nur Programmpunkte des dem im Eingangsblock des Schleifenrahmens dem Verzweigungsblock vorangehenden Aktionenblockes enthält, dessen (eventuell vorhandene) innere Ecken von $e_v^{(E)}$ verschieden sind.

Arten, die man zu unterscheiden wünscht, eine Ecke hinzunehmen und die Kanten zu ihnen entsprechend ziehen.

Wir betrachten eine Ecke des Schleifentransitionsgraphen, die eine Schlinge besitzt. Bei ihr kann es sich per Definition nicht um E oder A handeln. Sie ist ein abstrakter Zustand des verwendeten Programmpunktes des Schleifenrahmens, nennen wir ihn (m, S). Die Existenz der Schlinge ist äquivalent dazu, dass S in genau einer Schleifeniteration in sich überführt werden kann. In diesem Fall ist S semistabil bezüglich $B_{m\to m}$. Es ist aber möglich, dass für einen bezüglich $B_{m\to m}$ semistabilen Shapegraphen S mehr als eine Schleifeniteration nötig sind, um zu (m, S) zurückzukehren. Dann liegt ein geschlossener Kantenzug im Schleifentransitionsgraphen vor – es braucht sich aber nicht um einen Kreis zu handeln –, von dem (m, S) eine Ecke ist. Liegt umgekehrt (m, S) auf einem geschlossenen Kantenzug im Schleifentransitionsgraphen, dann ist S semistabil bezüglich $B_{m\to m}$. Damit haben wir Folgendes gezeigt:

7.20 Satz. *Es sei B eine Schleife eines TVLA-Programms, der Programmpunkt zu dem der Schleifenrücksprung erfolgt sei m, des Weiteren sei $\mathcal{TG}$ der für das Programm berechnete Transitionsgraph. Ein Shapegraph S ist genau dann semistabil bezüglich $B_{m\to m}$, wenn erstens der abstrakte Zustand (m, S) im Schleifentransitionsgraphen existiert und zweitens (m, S) eine Schlinge besitzt oder auf einem geschlossenen Kantenzug liegt.*

Ineinander transformierbare Shapegraphen

Der Begriff semistabil ist eine Charakterisierung für einen einzelnen Shapegraphen beziehungsweise für einen abstrakten Zustand im Hinblick auf einen Block. Wir ändern jetzt unsere Blickrichtung und betrachten anstatt eines einzelnen Shapegraphen Paare von Shapegraphen beziehungsweise von abstrakten Zuständen. Diese neue Blickrichtung wird uns im Folgenden erlauben, einen abstrahierten Transitionsgraphen abzuleiten, den wir für die Navigation „im Großen“ verwendet werden. Wir setzen:

7.21 Definition. *Es sei B eine Schleife eines TVLA-Programms und m der Programmpunkt, zu dem der Schleifenrücksprung erfolgt, des Weiteren sei $\mathcal{TG}(B)$ der auf B reduzierte Transitionsgraph. Zwei in $\mathcal{TG}(B)$ am Programmpunkt m vorkommende, nicht notwendig verschiedene Shapegraphen S und T heißen* ineinander transformierbar*, wenn gilt:*

- *Ist $S = T$, dann hat die Ecke (m, S) im Schleifentransitionsgraphen eine Schlinge.*
- *Ist $S \neq T$, dann liegen S und T im Schleifentransitionsgraphen auf einem geschlossenen Kantenzug.*

Wir nennen dann auch die abstrakten Zustände (m, S) und (m, T) ineinander transformierbar.

Sind für eine Schleife B zwei Shapegraphen des Rücksprungprogrammpunktes m ineinander transformierbar, dann sind sie beide bezüglich $B_{m\to m}$ semistabil. Ist umgekehrt ein Shapegraph S semistabil bezüglich $B_{m\to m}$ und existiert im Schleifentransitionsgraphen ein geschlossener Kantenzug, auf dem S liegt, dann sind S und jeder Shapegraph dieses geschlossenen Kantenzuges ineinander transformierbar. Beide Begriffe hängen sehr stark zusammen. Semistabile Shapegraphen wollten wir als allgemeine Shapegraphen interpretieren. Mithin repräsentieren auch ineinander transformierbare Shapegraphen allgemeine Shapegraphen.

Transformierbarkeit können wir auch als ein Form von Ähnlichkeit ansehen. Von dieser Ähnlichkeit wollen wir abstrahieren: Ineinander transformierbare Shapegraphen wollen wir nicht unterscheiden. Sind im Schleifentransitionsgraphen zwei Shapegraphen S und T (abstrakte Zustände) ineinander transformierbar, dann verschmelzen wir sie. Der Fall, dass beide Shapegraphen gleich sind, ist trivial. Wir setzen also verschiedene Shapegraphen voraus. In diesem Fall existiert ein geschlossener Kantenzug auf dem S und T liegen. In graphentheoretischer Terminologie bedeutet das nichts anderes, als dass beide Shapegraphen zu derselben starken Zusammenhangskomponente des Schleifentransitionsgraphen gehören. Wir berechnen die starken Zusammenhangskomponenten und verschmelzen die Shapegraphen einer jeden Komponente zu einer einzelnen Ecke und erhalten so einen weiteren (Hilfs-) Graphen:

7.22 Definition. *Gegeben sei eine Schleife und ein aus dem Transitionsgraphen abgeleiteter Schleifentransitionsgraph. Werden die abstrakten Zustände einer jeden starken Zusammenhangskomponente zu jeweils einer Ecke verschmolzen, dann entsteht ein gerichteter Graph, welcher der* Reihenfolgegraph *der Schleife (bezüglich des Transitionsgraphen) heiße.*

Der Reihenfolgegraph einer Schleife ist azyklisch. Wir können ihn als Graphen einer Ordnungsrelation (auf der Menge seiner Ecken) auffassen. Zwei Ecken stehen in Relation, wenn sie entweder gleich sind, oder es einen Kantenzug von der ersten zur zweiten gibt. Nach Definition ist diese Relation reflexiv. Da eine Konkatenation zweier Kantenzüge ihre Endecken verbindet, ist die Relation transitiv. Aus der Azyklizität folgt ihre Antisymmetrie. Diese Relation ist also eine Ordnungsrelation. Sie, und damit auch der Reihenfolgegraph, codiert die Reihenfolge, in der nicht ineinander transformierbare abstrakte Zustände bei Schleifeniterationen vorkommen. Der Reihenfolgegraph stellt eine abstrahierte, und damit auch abstrakte, Sicht von $\mathcal{TG}(B)$ dar. Er eignet sich damit für die abstrahierte Navigation durch die Schleife (genauer durch $\mathcal{TG}(B)$). Wenn wir während einer Visualisierungssitzung eine Schleife studieren und wir alle ihre unterschiedlichen Situationen betrachten wollen, dann sollten wir den Reihenfolgegraphen komplett traversieren.

Die obigen Überlegungen, wobei besonders die Ausführungen zur Schleife zu berücksichtigen sind, lassen sich auf beliebige TVLA-Programme übertragen. Wir berechnen mittels einer Shapeanalyse einen Transitionsgraphen $\mathcal{TG}$. Wenn wir nur einen

Teil des Programms untersuchen wollen, dann schränken wir $\mathcal{TG}$ auf die zugehörigen Programmpunkte ein. Dafür ist es nicht einmal nötig, dass es sich bei dem Teilprogramm um eine Konkatenation von Teilblöcken handelt. Wir verallgemeinern Definition 7.21 und nennen zwei abstrakte Zustände von $\mathcal{TG}$ *ineinander transformierbar*, wenn es in $\mathcal{TG}$ sowohl einen Kantenzug der Länge größer gleich eins vom ersten zum zweiten als auch vom zweiten zum ersten gibt. Ineinander transformierbare abstrakte Zustände fassen wir als ähnlich auf und wir wollen sie nicht unterscheiden. Wir berechnen die starken Zusammenhangskomponenten von $\mathcal{TG}$ und schrumpfen jede starke Zusammenhangskomponente zu einer Ecke zusammen. Der so entstehende Reihenfolgegraph induziert eine Ordnungsrelation, die die Reihenfolge respektiert, in der abstrakte Zustände auf Kantenzügen durch das Programm vorkommen. Er dient uns zur abstrahierten Navigation im Transitionsgraphen.

In diesem Kapitel haben wir die Überlegungen zum Ähnlichkeitsbegriff fortgeführt, den wir in Kapitel 6 eingeführt haben. Im ersten Abschnitt haben wir Methoden entwickelt, wie die einzelnen Klassen einer durch einen Ähnlichkeitsbegriff für Shapegraphen induzierten Partition (der Shapegraphenmenge eines Programmpunktes) weiter für die Visualisierung aufbereitet werden können. Im zweiten Abschnitt sind wir von Shapegraphen zu Pfaden im Transitionsgraphen übergegangen. Bei der Visualisierung (der Shapeanalyseausgabe) eines Algorithmus wird man nicht nur einzelne Shapegraphen oder die Menge der Shapegraphen eines einzelnen Programmpunktes studieren wollen, sondern auch Sequenzen von Shapegraphen. Im Hinblick auf dieses Ziel haben wir den Ähnlichkeitsbegriff von Shapegraphen auf Pfade im Transitionsgraphen übertragen. Außerdem haben wir einen reduzierten Transitionsgraphen eingeführt und studiert. In diesem Abschnitt galt unser Augenmerk ebenfalls dem Transitionsgraph und seinen Kantenzügen. Wir haben Charakterisierungen für Shapegraphen beziehungsweise für abstrakte Zustände entwickelt, die als Hilfestellung bei der Navigation im Transitionsgraphen dienen. Im ersten Teilabschnitt haben wir eine Charakterisierung im Hinblick auf die Unterscheidung spezieller versus allgemeiner Shapegraph entwickelt, die sich methodisch als Kombination von Einbettung und Ähnlichkeit darstellt. In diesem Teilabschnitt haben wir allgemeine Shapegraphen als solche verstanden, die unter einer Folge von TVLA-Aktionen in sich transformiert werden können. Damit eng verwandt ist die Vorstellung, dass zwei abstrakte Zustände allgemeiner Art sind, wenn sie ineinander transformiert werden können. Durch Identifikation ineinander transformierbarer abstrakter Zustände haben wir aus dem Transitionsgraphen einen Reihenfolgegraphen abgeleitet, der zur abstrahierten Navigation im Transitionsgraphen dient.

8 Symmetrie

> Feinerer Art ist schon jene Freude, welche beim Anblick alles Regelmäßigen und Symmetrischen [...] entsteht.
>
> *(Friedrich Nietzsche)*

In dieser Arbeit sind wir stark an Ähnlichkeitsbegriffen für Shapegraphen interessiert. In Kapitel 6 haben wir mittels einer Klasseneinteilungen ein parametrisches Ähnlichkeitskonzept entwickelt. Es gestattet, eine Menge von Shapegraphen, zum Beispiel die eines Programmpunktes, zu strukturieren. Der Ähnlichkeitsbegriff wird unter Verwendung der für die Analyse verwendeten Prädikate spezifiziert. Shapegraphen, die hinsichtlich dieser Ähnlichkeitsvorstellung nicht unterscheidbar sind, werden in Klassen zusammengefasst. Die Shapegraphen einer Klasse sind einander ähnlich, wir fassen sie als nicht wesentlich verschieden auf. Bei der Visualisierung kann die Aufmerksamkeit damit beispielsweise auf die Unterschiede zwischen Klassen gelenkt werden, während die Gemeinsamkeiten der Shapegraphen innerhalb einer Klasse ausgeblendet werden (können). So reduziert sich die Komplexität der zu betrachtenden Analyseausgabe. Eine Strukturierung der Analyseausgabe erweist sich auf diese Art für die Visualisierung als hilfreich.

In diesem Kapitel entwickeln wir ein weiteres Ähnlichkeitskonzept. Im Gegensatz zum vorgenannten fordern wir aber weder eine Übereinstimmung in Teilstrukturen noch im Auswertungsverhalten von Formeln. Der Ansatz beruht auf einem „zueinander ähnlichen“, einem „zueinander symmetrischen“ Aussehen der Shapegraphen. Wieder geht es nicht um ein starres Konzept. Wie schon bei der Klasseneinteilung beziehen wir den Symmetriebegriff auf die Prädikate der Analysespezifikation. Damit kann auch die Symmetrievorstellung verändert werden, und sie kann so flexibel den Wünschen und Anforderung während einer Visualisierungssitzung angepasst werden. Wir lassen uns bei diesem Ähnlichkeitsbegriff vom (zueinander symmetrischen) Aussehen der gezeichneten Shapegraphen inspirieren. Hierin liegt die Ursache, weswegen wir über diese Art von Ähnlichkeit als Symmetrie sprechen.

Ein Shapegraph beschreibt eine Heapsituation, ein dazu symmetrischer Shapegraph beschreibt eine dazu symmetrische Heapsituation. Beim Studium eines Algorithmus ist es zweckmäßig, zunächst nur eine dieser zueinander symmetrischen Situationen

zu betrachten. Dann kann aus der Tatsache, dass es sich um zueinander symmetrische Heapsituationen handelt, die andere (in der Regel) leicht erschlossen werden. Die Untersuchung des Algorithmus – und auch die Visualisierung – kann also besser gesteuert werden, wenn wir zueinander symmetrische Heapsituationen erkennen und herausfiltern können. Dieser Effekt verstärkt sich, wenn wir nicht nur Heapsituationen an Programmpunkten, sondern Kantenzüge im Transitionsgraphen betrachten. Wenn wir die abstrakte Programmausführung entlang eines Kantenzuges verstanden haben, erschließt sich in der Regel auch die abstrakte Programmausführung entlang eines dazu symmetrischen Kantenzuges.

Doch was bedeutet der Begriff *Symmetrie*? Das Wort entstammt dem Griechischen und bedeutet „Ebenmaß“, in adjektivistischem Gebrauch „gleichmäßig“. Das „Deutsche Wörterbuch“, das von Jacob und Wilhelm Grimm herausgegeben wurde, vergleiche [Grimm 2004], schreibt unter anderem dazu: „allgemein das ebenmäszige verhältnis der teile eines körpers, eines kunstwerkes zueinander und zum ganzen, die harmonische übereinstimmung der teile eines ganzen[1] [...] vielfältig erweitert und übertragen auf die gestalt anderer kunstformen und erscheinungen, in denen sich eine ‚regelmäszigkeit, gleichartigkeit des aufbaus, harmonische gesetzmäszigkeit, äuszere und innere ordnung, ein gleichgewicht der teile‘ u. dgl. kundtut“. Dieses Zitat sollte zum einen unsere intuitive Vorstellung des Wortes Symmetrie präzisieren, zum anderen soll es uns ein wenig als Leitbild für dieses Kapitel dienen. Im nächsten Abschnitt werden wir den Begriff Symmetrie für Shapegraphen konkretisieren und formalisieren.

8.1 Symmetrische Shapegraphen

In den einleitenden Absätzen dieses Kapitels haben wir eine allgemeine Vorstellung des Begriffs *Symmetrie* angegeben. Es stellt sich nun sofort die Frage, was er für Shapegraphen bedeuten solle. Im Symmetriebegriff soll sich das „zueinander symmetrische Aussehen“ der gezeichneten Shapegraphen widerspiegeln. Wir haben schon angedeutet, dass auch dieses Ähnlichkeitskonzept parametrisch sein soll: Es geht um Symmetrie bezüglich eines auf die (Prädikate der) Analysespezifikation Bezug nehmenden Spezifikators. Wir werden das Konzept anhand eines Beispiels entwickeln und es anschließend formalisieren.

Motivation

Wir betrachten als Beispiel wie gewöhnlich die Suche nach einem Element in einem binären Suchbaum. Der Suchalgorithmus ist in Abschnitt 4.1 auf Seite 66 beschrieben. Wir beschränken uns zunächst auf „Basisprädikate“, auf die grundlegenden Prädikate zur Beschreibung von Binärbäumen und verzichten insbesondere auf Ordnungsprädikate. Zwei der Shapegraphen, die in der Suchschleife, genauer beim

1. Dieser erste Teil ist ein Zitat aus Christian Wolffs „Mathematisches Lexicon“ (Leipzig, 1716).

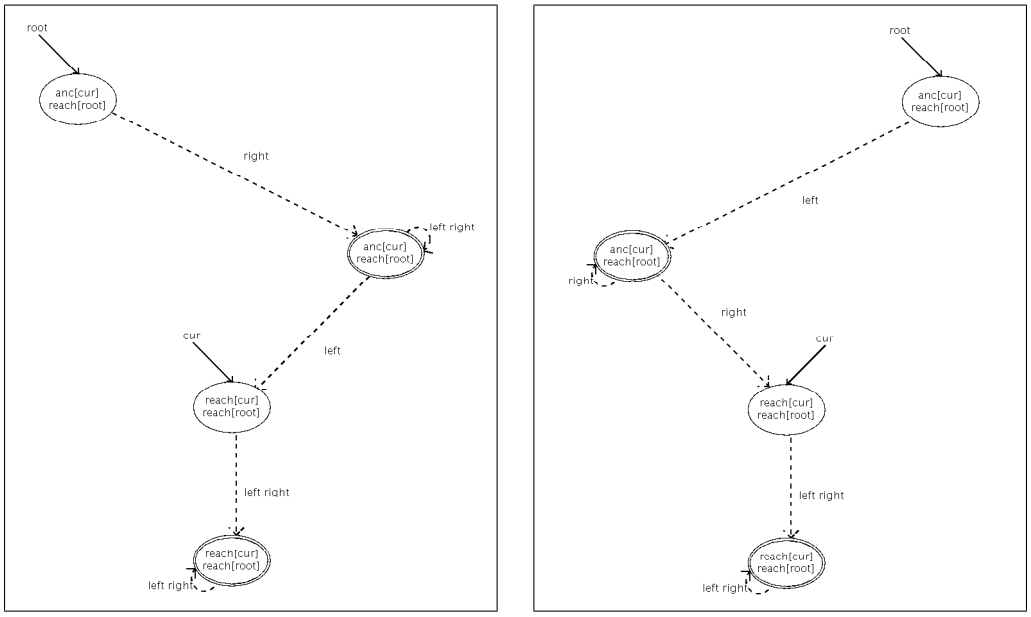

Abbildung 8.1: Zwei bezüglich $\{(\mathsf{left}, \mathsf{right})\}$ zueinander symmetrische Shapegraphen

Schleifeneingangsprogrammpunkt n2, auftreten, sind in Abbildung 8.1 gezeigt. Eine Fokussierung auf die graphische Darstellung der Shapegraphen zeigt: 1. Beide Shapegraphen sind an einer Vertikalen gespiegelt. 2. Die Beschriftungen der Kanten mit left und right sind vertauscht. Wenn wir von der graphischen Darstellung abstrahieren – wir lassen insbesondere die Eckennamen außer Acht –, dann unterscheiden sich beide Shapegraphen nur dadurch, dass die Prädikate left und right vertauscht sind. In diesem Sinne wollen wir die Shapegraphen als zueinander symmetrisch (bezüglich left und right) auffassen.

Wir haben Symmetrie mit der Vertauschung von Prädikaten beziehungsweise Prädikatswerten in Zusammenhang gebracht. Dies wird letztendlich auch die Weise sein, wie wir Symmetrie für Shapegraphen definieren werden. Das mag zunächst überraschen. Ein erster Ansatz, der als recht intuitives Vorgehen erscheint, wäre die Negation der Wahrheitswerte der für die Symmetrie ausgewählten Prädikate. Wir würden dann zwei Shapegraphen S und T zueinander symmetrisch nennen, wenn für jedes der betrachteten Prädikate p die Wahrheitswerte von p^S mit denen von $\neg\, p^T$ übereinstimmen. Wenn wir zum Beispiel in Abbildung 8.1 den linken Shapegraphen betrachten und uns auf die Prädikate left und right für diejenigen Argumente (Paare von Ecken) beschränken, die den Kanten im gezeichneten Shapegraphen entsprechen, dann erhalten wir (für diese Argumente) durch Negieren der Wahrheitswerte in der Tat die entsprechenden Prädikatswerte des rechten Shapegraphen aus Abbildung 8.1. Doch im Allgemeinen, für beliebige Argumente, führt eine Negation nicht zum gewünschten Ergebnis. Betrachten wir dazu wieder den

linken Shapegraph aus Abbildung 8.1, den wir S nennen. Es bezeichne u_{root} seine Wurzel, also die Ecke auf die der Zeiger root zeigt, und u_{cur} diejenige Ecke, auf die cur zeigt. Wenn wir die Wahrheitswerte der left- und right-Prädikate negieren, der so entstehende Shapegraph sei mit T bezeichnet, dann erhalten wir unter anderem $W_{T,\beta\left[\frac{v_1 \quad v_2}{u_{\mathsf{root}} u_{\mathsf{cur}}}\right]}\big(\mathsf{left}(v_1, v_2)\big) = 1$ und $W_{T,\beta\left[\frac{v_1 \quad v_2}{u_{\mathsf{root}} u_{\mathsf{cur}}}\right]}\big(\mathsf{right}(v_1, v_2)\big) = 1$. Beides ist keinesfalls erwünscht. Eine Negation von Wahrheitswerten führt somit nicht zu einem angemessenen Symmetriebegriff.

Fahren wir mit dem Suchbeispiel fort. Wir sind wieder an Symmetrie bezüglich der Prädikate left und right interessiert. Jetzt haben wir die Shapeanalyse um Prädikate ergänzt, die Ordnungseigenschaften der Datenelementen beschreiben. Wir können uns vorstellen, dass wir binäre Prädikate dle (data less or equal) und dg (data greater) hinzugefügt haben, vergleiche Abschnitt 4.1. (Wenn wir auch den Fall erlauben wollen, dass im Baum gleiche Datenwerte vorhanden sein dürfen, dann verwenden wir das Prädikat dge (data greater or equal) anstelle von dg.) Wenn wir die Prädikate left und right vertauschen, dann werden linke Teilbäume zu rechten und umgekehrt. Allerdings sind die Datenelemente der linken Teilbäume jetzt alle größer als die der rechten. Uns schwebt jedoch vor, dass die „normale“ Ordnung eines Suchbaums erhalten bleibt, dass also eine Inorder-Traversierung des Baums eine der Größe nach aufsteigend sortierte Aufzählung der Datenwerte ergibt. Wenn zwei Shapegraphen bezüglich left und right zueinander symmetrisch sind, dann sollten (müssen) Prädikate, die Ordnungseigenschaften beschreiben, mitvertauscht werden.

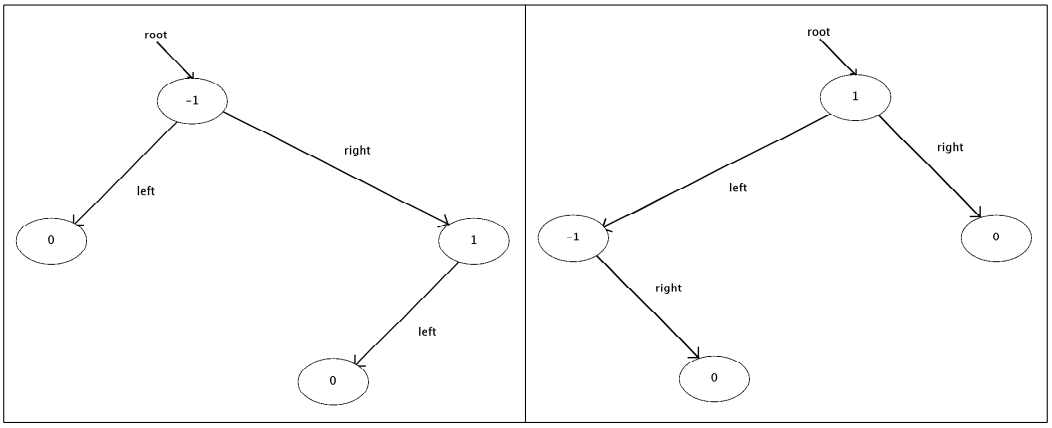

Abbildung 8.2: Zwei bezüglich left und right zueinander symmetrische AVL-Bäume

Als Nächstes ergänzen wir die Analysespezifikation um Prädikate, die zur Untersuchung von AVL-Bäumen benötigt werden, vergleiche [Parduhn 2005]. Zum Ersten benötigen wir Prädikate, die Balancierungsinformationen beschreiben. Wir verwenden die unären Prädikate mtwo, mone, zero, pone und ptwo, die für jede Ecke die Höhendifferenz ihres linken minus ihres rechten Teilbaums repräsentieren. Die Prädikate mtwo und ptwo stehen für eine Höhendifferenz von mindestens zwei. Sinnigerweise

kann für jede Ecke von diesen Prädikaten höchstens eines wahr sein. Zum Zweiten verwenden wir ein unäres Prädikat balanced, das für eine Ecke v angibt, ob der Teilbaum mit Wurzel v AVL-balanciert ist. In Abbildung 8.2 sind zwei AVL-Bäume mit jeweils vier Ecken abgebildet. Beide Bäume sind bezüglich left und right zueinander symmetrisch. Jedoch haben korrespondierende Ecken in den beiden Bäumen zum Teil unterschiedliche Balanceinformationen. Das bedeutet, dass wir die Prädikate für Balancierungsinformationen ebenfalls mitvertauschen müssen.

In den vorangegangenen Beispielen haben wir eine Vorstellung von Symmetrie bezüglich der Prädikate left und right entwickelt. Dabei haben wir gesehen, dass die Prädikate der Analysespezifikation hinsichtlich Symmetrie nicht unabhängig voneinander sind. Wir können Prädikate, bezüglich derer wir Symmetrie betrachten wollen, nicht beliebig auswählen; oft müssen weitere Prädikatenpaare hinzugenommen werden. Damit ist dieses Ähnlichkeitskonzept nicht so frei nutzbar wie das Konzept der Klasseneinteilung. Für die Auswahl der Symmetrie definierenden Prädikate ist ein Verständnis der (Abhängigkeiten in der) Analysespezifikation nötig.

In diesem Kapitel werden wir als Beispiel häufig Symmetrie bezüglich der Prädikate left und right betrachten, wobei wir uns aber nicht explizit auf eine bestimmte Analysespezifikation beziehen. Wir sagen dann, dass wir Symmetrie bezüglich (left, right) betrachten, oder wir legen gar als Symmetriedefinierer $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right})\}$ zugrunde. Wir implizieren mit dieser Formulierung aber, dass der Symmetriedefinierer je nach Analysespezifikation entsprechend sinnvoll ergänzt ist. Wir legen fest:

8.1 Vereinbarung. Wenn wir in den folgenden Beispielen Symmetrie bezüglich eines Prädikatenpaares (p, q) betrachten, dann denken wir uns den Symmetriedefinierer je nach Analysespezifikation entsprechend sinnvoll ergänzt, so dass die beabsichtigte Symmetriebetrachtung möglich ist.

Wenn wir eine Teilmenge der Prädikate der Anaylsespezifikation, bezüglich derer wir Symmetrie betrachten wollen, auswählen, dann ist a priori überhaupt nicht sichergestellt, dass in der Analyseausgabe solcherart Symmetrie vorhanden ist. Wir benötigen daher zusätzlich einen Mechanismus zur Beurteilung, ob und inwieweit die Analyseausgabe Symmetrie bezüglich der ausgewählten Prädikate aufweist. Diesen Aspekt werden wir in diesem Abschnitt qualitativ mitbehandeln. In Abschnitt 8.3 werden wir Symmetrieeigenschaften quantifizieren. Damit kann praktikabel beurteilt werden, wie ausgeprägt die Symmetrie bezüglich vorgegebenen Prädikate ist.

In den vorangegangenen Beispielen haben wir (primär) immer Symmetrie bezüglich der Prädikate left und right betrachtet. Sie repräsentieren die Zeigerstruktur des Heaps. Bei der Visualisierung von Shapegraphen gehen wir, solange keine speziellen Anforderungen ein anderes Vorgehen nahelegen, von der Prämisse aus, dass die

graphentheoretische Struktur der Shapegraphen die Zeigerstruktur des Heaps widerspiegeln soll, vergleiche die Strukturerhaltungsforderung auf Seite 61. Daher widmen wir den Prädikaten, die Zeiger repräsentieren, viel Aufmerksamkeit. Folglich hatten wir auch in den Beispielen Symmetrien betrachtet, bei denen die Zeigerprädikate beteiligt sind. Aber natürlich ist das Symmetriekonzept nicht auf sie beschränkt. Fortgeschrittene Anwender können es auch gewinnbringend einsetzen, wenn sie andere Prädikate für die Symmetriebetrachtung heranziehen.

Symmetrie für Shapegraphen

Nun wenden wir uns der Formalisierung des Symmetriekonzeptes zu. Wir betrachten Symmetrie bezüglich ausgewählter Prädikate. Daher benötigen wir einen Spezifikator. Wir setzen:

8.2 Definition. *Es seien $p_1, q_1, p_2, q_2, \ldots, p_n, q_n$, $n \geq 1$, (verschiedene) Prädikate der Analysespezifikation so, dass für $1 \leq i \leq n$ die Prädikate p_i und q_i gleiche Stelligkeit haben. Dann heiße $\mathcal{D}_\Sigma = \{(p_1, q_1), \ldots, (p_n, q_n)\}$ ein* Symmetriedefinierer.

In den einleitenden Beispielen haben wir Symmetrie mittels Vertauschung von Prädikaten (beziehungsweise Prädikatswerten) motiviert. Diese Sichtweise benutzen wir auch bei der formalen Definition. Was der Begriff der Vertauschung bedeuten soll, erklären wir zunächst gesondert, also ohne Bezugnahme auf den Symmetriebegriff.

8.3 Definition. *Es seien S und T zwei Shapegraphen; das Universum von S sei U_S, jenes von T sei U_T. Es existiere eine Bijektion $f : U_S \to U_T$ vom Universum von S ins Universum von T.*

1. *Es sei p ein Prädikat der Analysespezifikation. Die Bijektion f* respektiere *p, wenn:*
   a) *Ist p ein nullstelliges Prädikat, dann gelte $W_{S,\beta}(p) = W_{T,\beta}(p)$.*
   b) *Ist p ein einstelliges Prädikat, dann gelte für alle $u \in U_S$*
   $$W_{S,\beta}\left[\tfrac{v}{u}\right]\big(p(v)\big) = W_{T,\beta}\left[\tfrac{v}{f(u)}\right]\big(p(v)\big).$$
   c) *Ist p ein zweistelliges Prädikat, dann gelte für alle $u_1, u_2 \in U_S$*
   $$W_{S,\beta}\left[\tfrac{v_1\, v_2}{u_1 u_2}\right]\big(p(v_1, v_2)\big) = W_{T,\beta}\left[\tfrac{v_1 \quad v_2}{f(u_1) f(u_2)}\right]\big(p(v_1, v_2)\big).$$

   Es sei P eine Menge von Prädikaten. Die Bijektion f respektiere P, wenn sie jedes Prädikat $p \in P$ respektiert.
2. *Es seien p und q zwei Prädikate der Analysespezifikation. Die Bijektion f* respektiere *(p, q), wenn:*
   a) *Sind p und q nullstellige Prädikate, dann gelte $W_{S,\beta}(p) = W_{T,\beta}(q)$ und $W_{S,\beta}(q) = W_{T,\beta}(p)$.*

b) Sind p und q einstellige Prädikate, dann gelte für alle $u \in U_S$

$$W_{S,\beta}\left[\frac{v}{u}\right]\big(p(v)\big) = W_{T,\beta}\left[\frac{v}{f(u)}\right]\big(q(v)\big)$$

und

$$W_{S,\beta}\left[\frac{v}{u}\right]\big(q(v)\big) = W_{T,\beta}\left[\frac{v}{f(u)}\right]\big(p(v)\big).$$

c) Sind p und q zweistellige Prädikate, dann gelte für alle $u_1, u_2 \in U_S$

$$W_{S,\beta}\left[\frac{v_1\, v_2}{u_1 u_2}\right]\big(p(v_1, v_2)\big) = W_{T,\beta}\left[\frac{v_1 \quad v_2}{f(u_1) f(u_2)}\right]\big(q(v_1, v_2)\big)$$

und

$$W_{S,\beta}\left[\frac{v_1\, v_2}{u_1 u_2}\right]\big(q(v_1, v_2)\big) = W_{T,\beta}\left[\frac{v_1 \quad v_2}{f(u_1) f(u_2)}\right]\big(p(v_1, v_2)\big).$$

Zunächst notieren wir einige für die Respektierung gültige Aussagen. Die Beweise folgen entweder unmittelbar oder mit nur leichtem technischen Aufwand aus der Definition, weswegen wir auf sie verzichten können. Die Ausgangssituation sei wie in der Definition. Es seien S und T zwei Shapegraphen, mit f bezeichnen wir eine Bijektion des Universums von S aufs Universum von T und es seien p und q Prädikate der Analysespezifikation. Es gilt:

1. Eine Bijektion respektiert (p, q) genau dann, wenn sie (q, p) respektiert. (Hier nutzen wir die Tatsache, dass die Definition der Respektierung symmetrisch formuliert ist.)
2. a) Respektiert f ein Prädikat p, dann respektiert auch die Umkehrabbildung f^{-1} das Prädikat p.
   b) Respektiert f ein Prädikatenpaar (p, q), dann respektiert auch die Umkehrabbildung f^{-1} das Paar (p, q).

Bevor wir mit dem Symmetriekonzept fortfahren, verweilen wir kurz bei der Respektierung und untersuchen diesen Begriff weiter. Wenn eine Bijektion f zwischen den Universen zweier Shapegraphen alle Prädikate der Analysespezifikation respektiert, dann bedeutet das nichts anderes, als dass f ein Isomorphismus zwischen den beiden Shapegraphen ist. Ist umgekehrt f ein Isomorphismus zwischen zwei Shapegraphen, dann muss f notwendigerweise jedes Prädikat der Analysespezifikation respektieren. Dies zeigt:

8.4 Satz. *Es sei S ein Shapegraph mit Universum U_S und T ein Shapegraph mit Universum U_T, des Weiteren sei $f : U_S \to U_T$ eine Bijektion. Die Funktion f ist genau dann ein Isomorphismus, wenn f jedes Prädikat der Analysespezifikation respektiert.*

Objekte gleicher Art vergleicht man oft hinsichtlich ihrer Ähnlichkeit beziehungsweise Verschiedenheit miteinander. Dabei bezeichnet man mit Gleichheit eine sehr starke Ausprägung, einen sehr hohen Grad an Ähnlichkeit. Im Begriff Isomorphie drückt sich in der Mathematik ganz allgemein eine Gleichheitsvorstellung aus. Dies trifft natürlich auch auf Shapegraphen zu. Wenn eine Bijektion zwischen den Universen zweier Shapegraphen alle Prädikate der Analysespezifikation respektiert, dann ist sie ein Isomorphismus. Wenn wir stattdessen nur verlangen, dass sie eine Teilmenge der Prädikate respektiert, dann stellt dies eine schwächere Forderung dar. Die Ähnlichkeit zweier Shapegraphen ist dann auch schwächer ausgeprägt, beziehungsweise wir können sie als schwächer ausgeprägt auffassen.

Wir können diesen Ansatz nutzen, um einen parametrischen Ähnlichkeitsbegriff zu definieren. Als Spezifikator verwenden wir eine Teilmenge P der Prädikate der Analysespezifikation. Wir nennen zwei Shapegraphen ähnlich bezüglich P, wenn es eine Bijektion zwischen ihren Universen gibt, die P respektiert. Es ist nicht schwer zu zeigen, dass die so definierte binäre Relation eine Äquivalenzrelation ist. Damit ordnet sich dieser Ähnlichkeitsbegriff in die Herangehensweise in Kapitel 6 ein. Wir nehmen aber davon Abstand, dieser Art von Ähnlichkeit einen Namen zu gegeben. Der Grund liegt darin, dass wir ihr schon begegnet sind.

Betrachten wir dazu zwei Shapegraphen S und T, die bezüglich P ähnlich seien. Wir leiten aus S einen Shapegraphen S' ab. Er habe das gleiche Universum wie S und enthalte alle Prädikate aus P mit den Wahrheitswerten von S. Alternativ können wir uns auch vorstellen, dass S' aus S durch Entfernen aller Prädikate, die nicht in P enthalten sind, hervorgeht. Ebenso leiten wir T' aus T ab. Die fortgelassenen beziehungsweise entfernten Prädikate haben keinen Einfluss auf die Ähnlichkeit: Die Shapegraphen S und T sind genau dann ähnlich (bezüglich P), wenn es S' und T' sind. Eine Bijektion f des Universums von S ins Universum von T ist selbstverständlich auch eine (ebensolche) Bijektion für S' und T'. Wenn f alle Prädikate aus P respektiert, dann bedeutet das, in S' und T' sind keine anderen Prädikate vorhanden, dass f ein Isomorphismus von S' auf T' ist. Andererseits ist S' der durch den Teilstrukturdefinierer $(1, P)$ induzierte Teilgraph von S. Ebenso gilt $T' = T_{(1,P)}$. Insgesamt sind S und T also schwach ähnlich bezüglich $(1, P)$, vergleiche Definition 6.8 auf Seite 116. Setzen wir für die Umkehrung voraus, dass S und T schwach ähnlich bezüglich $(1, P)$ seien. Dann existiert nach Definition ein Isomorphismus f von $S_{(1,P)}$ auf $T_{(1,P)}$. Unter Berücksichtigung von $S' = S_{(1,P)}$ und $T' = T_{(1,P)}$ zeigt dies, dass f alle Prädikate aus P respektiert. Diese Ähnlichkeitsvorstellung ist folglich ein Spezialfall der schwachen Ähnlichkeit. Die Eckenauswahlformel ist immer 1, sie wählt stets alle Ecken eines Shapegraphen aus. Wir fassen zusammen:

8.5 Satz. *Es sei P eine Teilmenge der Prädikate der Analysespezifikation. Zwei Shapegraphen sind genau dann schwach ähnlich bezüglich des Teilstrukturdefinierers $(1, P)$, wenn es eine Bijektion zwischen ihren Universen gibt, die P respektiert.*

Wenden wir uns nach diesem Einschub jetzt der Symmetrie zu. Aufbauend auf den vorangegangenen Begriffen erklären wir, wann Shapegraphen symmetrisch sind:

8.6 Definition. *Es sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Zwei Shapegraphen S und T heißen* zueinander symmetrisch bezüglich $\mathcal{D}_\Sigma$*, wenn:*

1. *Es existiere eine Bijektion f zwischen den Universen von S und T.*
2. *Die Bijektion f respektiere jedes $(p, q) \in \mathcal{D}_\Sigma$.*
3. *Die Bijektion f respektiere jedes Prädikat p der Analysespezifikation, das nicht in $\mathcal{D}_\Sigma$ enthalten ist.*

Die auf diese Weise erklärte binäre Relation schreiben wir als $\mathrm{sym}_{\mathcal{D}_\Sigma}$.

Die beiden Shapegraphen in Abbildung 8.1 auf Seite 199 sind zueinander symmetrisch bezüglich $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right})\}$. – Wenn der Zusammenhang zum Symmetriedefinierer klar ist und keine Unklarheiten oder Verwechslungen zu befürchten sind, dann schreiben wir anstelle von zueinander symmetrisch bezüglich $\mathcal{D}_\Sigma$ kürzer zueinander symmetrisch. Ebenso schreiben wir gelegentlich sym, was stellvertretend für $\mathrm{sym}_{\mathcal{D}_\Sigma}$ steht. Im Folgenden gehen wir immer, wenn wir nicht explizit eine Einschränkung formulieren, davon aus, dass $\mathrm{sym}_{\mathcal{D}_\Sigma}$ über (dem Kreuzprodukt) der Menge aller Shapegraphen bezüglich der verwendeten Analysespezifikation erklärt ist. Gilt $(S, T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ für zwei Shapegraphen S und T, dann nennen wir T auch den *symmetrischen Partner* von S.

In den Beispielen zu Beginn des Abschnitts haben wir gesehen, dass Prädikate bezüglich Symmetrie oft nicht unabhängig voneinander sind und bei der Spezifikation von Symmetriedefinierern oft andere Prädikate mitberücksichtigt werden müssen. Wenn wir eine Analysespezifikation für Bäume verwenden, die neben den Strukturprädikaten auch Prädikate zur Beschreibung von Ordnungseigenschaften auf den Datenelementen enthält, dann müssen wir zusätzlich $(\mathsf{dle}, \mathsf{dg})$ in $\mathcal{D}_\Sigma$ aufnehmen. (Erlauben wir gleiche Datenwerte im Baum, dann müssen wir das Prädikat dge anstelle von dg verwenden.) Im Falle von AVL-Bäumen müssen wir darüber hinaus die Prädikate zur Beschreibung von Balanceinformationen mitberücksichtigen. Zur Betrachtung von Symmetrieeigenschaften bezüglich left und right ist der vollständige Symmetriedefinierern dann $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right}), (\mathsf{dle}, \mathsf{dg}), (\mathsf{pone}, \mathsf{mone}), (\mathsf{ptwo}, \mathsf{mtwo})\}$.

Wir betrachten normalerweise keine einzelnen Shapegraphen. Auch betrachten wir nicht die Menge aller Shapegraphen bezüglich der verwendeten Analysespezifikation, sondern Teilmengen von ihr. Shapegraphenmengen mit einer stark ausgeprägten Symmetrie, mit einem besonders hohen Anteil zueinander symmetrischer Shapegraphen, zeichnen wir aus:

8.7 Definition. *Es sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Eine Menge $\mathcal{M}$ von Shapegraphen heiße* symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$*, wenn für jeden Shapegraphen $S \in \mathcal{M}$ ein nicht notwendig von S verschiedener Shapegraph $T \in \mathcal{M}$ mit $(S, T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ existiert.*

Betrachten wir als Beispiel die Suche nach einem Element in einem binären Suchbaum. Wir legen die Basisspezifikation zugrunde und wählen als Symmetriedefinierer $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right})\}$. (Bei einer erweiterten Analysespezifikation ist der Symmetriedefinierer geeignet zu ergänzen, vergleiche die Vereinbarung in diesem Abschnitt hierzu.) Das TVLA-Programm sei jenes aus Abbildung 4.3 (Seite 66). Wir verwenden die bekannten Startshapegraphen und wir führen die übliche Shapeanalyse aus. Dann ist die Shapegraphenmenge jedes Programmpunktes symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$.

Aus praktischen Gesichtspunkten kann man es als wünschenswert erachten, den Symmetriebegriff allgemeiner zu formulieren. Denken wir zum Beispiel an Prädikate, die Ordnungseigenschaften auf den Datenelementen beschreiben. Dazu haben wir die binären Prädikate dle (data less or equal) und dg (data greater) eingeführt. Für eine befriedigende Spezifikation von Suchbäumen genügt bereits dle. Zur Nutzung des Symmetriekonzepts im Zusammenhang mit $(\mathsf{left}, \mathsf{right})$ benötigen wir allerdings das dazu „symmetrische“ Prädikat dg. Es kann also der Fall eintreten, dass man ein Prädikat p bei der Symmetriebetrachtung verwenden will, aber es kein zu p „symmetrisches“ Prädikat gibt. Man kann die Definition verallgemeinern und im Symmetriedefinierer neben Einträgen der Form (p, q) auch solche der Form $(p, \neg q)$ erlauben. In diesem Fall fordert man, dass die Wahrheitswerte von p mit denen von $\neg q$ korrespondieren müssen. Im Falle von dle können wir dann $(\mathsf{dle}, \neg\, \mathsf{dle})$ in den Symmetriedefinierer aufnehmen. Diese zusätzliche Flexibilität kann sich (insbesondere für fortgeschrittene Anwender) als nützlich verweisen. Andererseits kann man sich auf den Standpunkt stellen, dass alle benötigten Prädikate, die man in der Visualisierung wünscht oder benötigt, bereits in der Analysespezifikation enthalten sein müssen. Wir haben in diesem Kapitel darauf verzichtet, diese (und etwaige andere) Verallgemeinerung mit in die Definition und Betrachtung aufzunehmen.

Eigenschaften der Symmetrierelation

Als Nächstes untersuchen wir Eigenschaften der Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$. Wir beginnen mit:

8.8 Satz. *Für jeden Symmetriedefinierer $\mathcal{D}_\Sigma$ ist die Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$ symmetrisch.*

Beweis. Es gelte $(S, T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. Nach Definition existiert eine Bijektion f vom Universum von S auf das Universum von T, die den Symmetriedefinierer $\mathcal{D}_\Sigma$ respektiert. Die Umkehrabbildung f^{-1} bildet das Universum von T aufs Universum von S ab. Sie respektiert ebenfalls $\mathcal{D}_\Sigma$. Damit gilt $(T, S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. □

Eine weitere wichtige Eigenschaft einer Relation ist die Reflexivität, und wir wollen feststellen, ob sie auch für $\mathrm{sym}_{\mathcal{D}_\Sigma}$ zutrifft. Betrachten wir dazu ein Beispiel. Es sei S der linke Shapegraph aus Abbildung 8.1 (Seite 199). Der Symmetriedefinierer ist $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right})\}$. Nehmen wir an, dass $\mathrm{sym}_{\mathcal{D}_\Sigma}$ reflexiv wäre. Dann gäbe es eine

Bijektion f des Universums von S auf sich mit den in der Definition genannten Eigenschaften. Da in $\mathcal{D}_\Sigma$ keine Abstraktionsprädikate enthalten sind, respektierte f jedes Abstraktionsprädikat. Damit wäre f die Identität. Bezeichnen wir mit u_1 die Wurzel von S und mit u_2 die Summary-Ecke zwischen root und cur. Nach Definition gälte dann insbesondere

$$W_{S,\beta\left[\frac{v_1\,v_2}{u_1\,u_2}\right]}\big(\mathsf{left}(v_1,v_2)\big) = W_{S,\beta\left[\frac{v_1\,v_2}{u_1\,u_2}\right]}\big(\mathsf{right}(v_1,v_2)\big).$$

Die Wahrheitswerte sind aber 0 auf der linken Seite des Gleichheitszeichen und $\frac{1}{2}$ auf der rechten. Wir erhalten einen Widerspruch. Dies zeigt $(S,S) \notin \mathrm{sym}_{\mathcal{D}_\Sigma}$. Die Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$ ist also in der Regel nicht reflexiv.

Es kann jedoch der Fall eintreten, dass für einen Shapegraphen S die Bedingung $(S,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ gilt. Solche Shapegraphen wollen wir auszeichnen:

8.9 Definition. *Ein Shapegraph S heiße* symmetrisch bezüglich $\mathcal{D}_\Sigma$*, wenn* $(S,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ *gilt.*

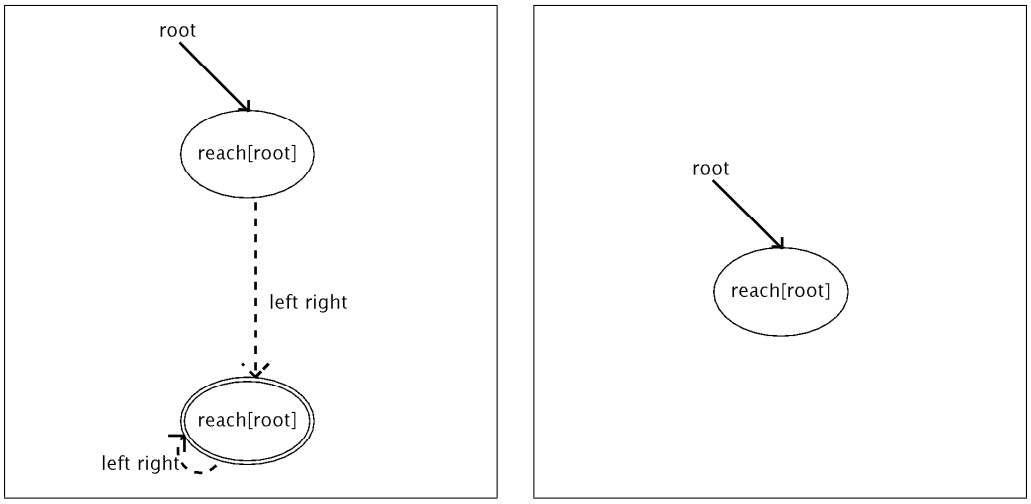

Abbildung 8.3: Zwei bezüglich $\big\{(\mathsf{left},\mathsf{right})\big\}$ symmetrische Shapegraphen

Es sei S ein Shapegraph. Die Identität vom Universum von S auf sich respektiert jedes Prädikat der Analysespezifikation. Folglich ist jeder Shapegraph S bezüglich $\mathcal{D}_\Sigma = \emptyset$ symmetrisch. Abbildung 8.3 zeigt zwei Beispiele für bezüglich $\mathcal{D}_\Sigma = \big\{(\mathsf{left},\mathsf{right})\big\}$ symmetrische Shapegraphen. Sie treten als (potentielle) Startshapegraphen bei den Analysen von Baumalgorithmen auf. Dabei ist der nur aus einer Ecke bestehende Shapegraph trivial, die left- und right-Prädikate werten für alle Argumente zum Wahrheitswert 0 aus.

In unseren Beispielen betrachten wir das Symmetriekonzept vorrangig für Symmetriespezifikationen, die (left, right) enthalten. In diesem Fall sind symmetrische

Shapegraphen selten. Für je zwei Ecken u_1 und u_2 eines symmetrischen Shapegraphen S gilt dann $W_{S,\beta}\left[\frac{v_1\,v_2}{u_1\,u_2}\right]\big(\mathsf{left}(v_1,v_2)\big) = W_{S,\beta}\left[\frac{v_1\,v_2}{u_1\,u_2}\right]\big(\mathsf{right}(v_1,v_2)\big)$. Wenn es im Shapegraphen eine Individuenecke u gibt, so dass ihr linkes und rechtes Kind zu verschiedenen Ecken des Shapegraphen gehören, dann kann S nicht symmetrisch sein. Wenn wir das linke Kind mit u_l bezeichnen, dann gälte insbesondere $W_{S,\beta}\left[\frac{v_1\,v_2}{u\,u_l}\right]\big(\mathsf{left}(v_1,v_2)\big) \geq \frac{1}{2}$, während andererseits $W_{S,\beta}\left[\frac{v_1\,v_2}{u\,u_l}\right]\big(\mathsf{right}(v_1,v_2)\big) = 0$ wäre. Das stünde im Widerspruch zur Symmetrie. Diese Situation tritt bei Verwendung der Standardspezifikation häufig auf. Sobald wir von der Wurzel ausgehend einen Baum traversieren, werden die Teilbäume, deren Wurzeln das linke und rechte Kind der Baumwurzel sind, im Shapegraphen nicht durch dieselben Ecken repräsentiert. Mit der gleichen Überlegung sehen wir, dass ein Shapegraph nicht symmetrisch sein kann, wenn er neben der Wurzel eine weitere Individuumsecke besitzt. Der dort hineinführende Strukturzeiger ist entweder ein left- oder ein right-Zeiger.

Es gilt die folgende Bedingung:

8.10 Satz. *Es sei S ein Shapegraph. Gilt $p^S = q^S$ für alle $(p,q) \in \mathcal{D}_\Sigma$, dann ist S symmetrisch.*

Beweis. Es sei S ein Shapegraph. Des Weiteren bezeichne f die identische Abbildung des Universums von S auf sich. Sie ist ein Automorphismus und respektiert damit jedes Prädikat der Analysespezifikation. Es sei $(p,q) \in \mathcal{D}_\Sigma$. Die Bedingung $p^S = q^S$ bedeutet nichts anderes, als dass f das Paar (p,q) respektiert, vergleiche Definition 8.3. Folglich ist S symmetrisch. □

Wir erhalten weitere Einsicht in die Symmetrierelation, wenn wir sie auf Transitivität hin untersuchen. Dazu sei ein Symmetriedefinierer $\mathcal{D}_\Sigma$ gegeben, und es seien $(R,S),(S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. Die Universen der drei Shapegraphen seien U_R, U_S und U_T. Nach Definition existieren Bijektionen g_1 (von U_R auf U_S) und g_2 (von U_S auf U_T), die jedes Prädikatenpaar in $\mathcal{D}_\Sigma$ und jedes Prädikat, dass nicht in $\mathcal{D}_\Sigma$ enthalten ist, respektieren. Die Komposition $f = g_2 \circ g_1$ ist dann eine Bijektion von U_R auf U_T. Es ist sofort einsichtig, dass auch f jedes Prädikat respektiert, dass nicht in $\mathcal{D}_\Sigma$ enthalten ist. Wir untersuchen die Situation für ein Paar $(p,q) \in \mathcal{D}_\Sigma$. Wir betrachten nur den Fall, dass p und q einstellige Prädikate sind; die anderen Fälle verhalten sich analog. Nach Definition gilt (unter anderem)

$$\forall u \in U_R\colon \quad W_{R,\beta}\left[\frac{v}{u}\right]\big(p(v)\big) = W_{S,\beta}\left[\frac{v}{g_1(u)}\right]\big(q(v)\big).$$

Des Weiterem gilt (unter anderem)

$$\forall \bar{u} \in U_S\colon \quad W_{S,\beta}\left[\frac{v}{\bar{u}}\right]\big(q(v)\big) = W_{T,\beta}\left[\frac{v}{g_2(\bar{u})}\right]\big(p(v)\big).$$

Da es sich bei g_1 um eine Bijektion zwischen den Universen handelt, ist diese Gleichung äquivalent zu

$$\forall u \in U_R\colon \quad W_{S,\beta\left[\frac{v}{g_1(u)}\right]}\big(q(v)\big) = W_{T,\beta\left[\frac{v}{g_2(g_1(u))}\right]}\big(p(v)\big).$$

Zusammen erhalten wir

$$\forall u \in U_R\colon \quad W_{R,\beta\left[\frac{v}{u}\right]}\big(p(v)\big) = W_{T,\beta\left[\frac{v}{f(u)}\right]}\big(p(v)\big).$$

Diese Gleichung besagt nichts anderes, als dass f das Prädikat p respektiert. Wenn wir in diesem Gedankengang die Rollen von p und q vertauschen, dann erhalten wir dieselbe Gleichungskette für q; die Bijektion f respektiert auch das Prädikat q. Damit haben wir gezeigt, dass f jedes Prädikat der Analysespezifikation respektiert. Die Shapegraphen R und T sind folglich isomorph. – Alle von uns betrachteten Shapegraphen entstammen einer Shapeanalyse. Die von TVLA berechnete Ausgabe ist derart, dass für jeden Programmpunkt des untersuchten Programms die Shapegraphen paarweise nichtisomorph sind. Daher ist es sinnvoll, wenn wir uns bei der Entwicklung des Symmetriebegriffs auf Mengen paarweise nichtisomorpher Shapegraphen einschränken. In diesem Fall folgt aus $(R,S),(S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ schon $R = T$, die Shapegraphen R und T sind identisch.

Aus dem vorangegangenen Gedankengang können wir zweierlei ableiten. Zum Ersten sehen wir, dass es (in der Analyseausgabe an jedem Programmpunkt) zu jedem Shapegraphen S höchstens einen Shapegraphen T gibt, der mit S in Relation steht. Dazu betrachten wir $(S,T),(S,T') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. Wegen der Symmetrie von $\mathrm{sym}_{\mathcal{D}_\Sigma}$ ist dies äquivalent zu $(T,S),(S,T') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. Die vorangegangene Überlegung zeigt $T = T'$. Der Fall $(T,S),(T',S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ lässt sich wegen der Symmetrie von $\mathrm{sym}_{\mathcal{D}_\Sigma}$ hierauf reduzieren, den verbleibenden Fall $(T,S),(S,T') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ haben wir schon behandelt. Zum Zweiten sehen wir, dass $\mathrm{sym}_{\mathcal{D}_\Sigma}$ im Allgemeinen nicht transitiv ist. Denn dann müsste mit $(S,T),(T,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ auch $(S,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ gelten. Wir haben aber schon festgestellt, dass $\mathrm{sym}_{\mathcal{D}_\Sigma}$ im Allgemeinen nicht reflexiv ist; der linke Shapegraph aus Abbildung 8.1 diente als Gegenbeispiel.

Symmetrieinduzierte Strukturierung

Zu jedem Shapegraphen gibt es bis auf Isomorphie also höchstens einen Shapegraphen, so dass beide hinsichtlich $\mathrm{sym}_{\mathcal{D}_\Sigma}$ in Relation stehen. (Alternativ könnten wir uns $\mathrm{sym}_{\mathcal{D}_\Sigma}$ auch als eine partielle Funktion vorstellen, die einem Shapegraphen seinen symmetrischen Partner zuordnet; doch wir vermeiden diese Sicht.) Damit lässt sich das Symmetriekonzept gut mit der Vorstellung einer Paarbildung in Einklang bringen. Wir werden sie im Folgenden entwickeln. Es sei noch einmal daran erinnert, dass die Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$ (auf dem Kreuzprodukt) der Menge aller Shapegraphen bezüglich der zugrundeliegenden Analysespezifikation erklärt ist.

Bei einem vorgegebenen Symmetriedefinierer kann man natürlich zu jedem Shapegraphen S gemäß Definition rein formal eine logische Struktur T konstruieren, so

dass S und T die Bedingungen in Definition 8.6 erfüllen. Es braucht sich bei dieser Struktur T aber nicht um einen legalen Shapegraphen bezüglich der verwendeten Analysespezifikation zu handeln. Betrachten wir diesen Sachverhalt an einem Beispiel. Wir haben es wieder mit binären Suchbäumen zu tun und es sei $\mathcal{D}_\Sigma = \{(\mathsf{reach}[\mathsf{root}], \mathsf{reach}[\mathsf{cur}])\}$. Der Shapegraph S sei derart, dass root und cur auf verschiedene Ecken des Graphen zeigen. Die Wurzel sei mit u bezeichnet, sie ist die eindeutig bestimmte Ecke u mit $W_{S,\beta\left[\frac{v}{u}\right]}\big(\mathsf{root}(v)\big) = 1$. Sie ist ebenfalls in T die Wurzel des Baumes. In T, also nach der „Vertauschung“ von reach[root] und reach[cur], gilt (für die Wurzel) $W_{T,\beta\left[\frac{v}{u}\right]}\big(\mathsf{reach}[\mathsf{root}](v)\big) = 0$. Nun ist reach[root] ein Instrumentationsprädikat, und es ist durch die Formel

$$\mathsf{reach}[\mathsf{root}](v) := \exists v_1 \colon \mathsf{root}(v_1) \wedge \mathsf{downStar}(v_1, v)$$

definiert. Demgemäß muss für die Wurzel u natürlich $W_{T,\beta\left[\frac{v}{u}\right]}\big(\mathsf{reach}[\mathsf{root}](v)\big) = 1$ gelten. Folglich ist T kein (legaler) Shapegraph bezüglich der verwendeten Analysespezifikation.

Und selbst wenn durch formale Vertauschung der Prädikatswerte ein legaler Shapegraph bezüglich der verwendeten Analysespezifikation entsteht, so gibt es a priori keinen Grund, warum es ein Shapegraph in der Art sein sollte, wie wir sie von TVLA erhalten. Stellen wir uns vor, der Symmetriedefinierer bestehe aus Paaren einstelliger Prädikate, von denen jeweils eines ein Abstraktionsprädikat und das andere ein Nichtabstraktionsprädikat ist. Es ist nicht sehr schwer dies so einrichten, dass bei einem ausgewählten Shapegraphen nach der formalen Vertauschung der Wahrheitswerte Ecken existieren, die die gleichen Wahrheitswerte bei den Abstraktionsprädikaten haben. Der bei TVLA zur Anwendung kommende Blur-Mechanismus verhindert dort aber gerade die Existenz solcher Ecken.

Und selbst wenn durch formale Vertauschung der Prädikatswerte ein (legaler) Shapegraph entsteht, der auch in der TVLA-Ausgabe auftritt, gibt es a priori keinen Grund, warum er am selben Programmpunkt vorkommen sollte. Um dies zu sehen, untersuchen wir wieder das Suchbeispiel. Der Algorithmus ist in Abbildung 4.3 auf Seite 66 dargestellt. Wir betrachten erneut die beiden Shapegraphen in Abbildung 8.1 auf Seite 199. Der linke Shapegraph sei mit S, der rechte mit T bezeichnet. Beide sind bezüglich $\mathcal{D}_\Sigma = \{(\mathsf{left}, \mathsf{right})\}$ symmetrisch. Beide Shapegraphen treten am Programmpunkt n2, das ist der Schleifeneingangspunkt, auf. Nach dem Vergleich der Datenwerte verzweigt der Programmfluss, falls der Algorithmus dort nicht terminiert, entweder zum Programmpunkt n5 oder zu n6.

Die beiden Äste der Verzweigung sind Aktionenblöcke, die jeweils nur einen einzigen Programmpunkt enthalten. Die dort jeweils ausgeführte Aktion führt schon aus dem Block hinaus, zum Schleifeneingangspunkt zurück. Das liegt darin begründet, dass der Algorithmus in verschmolzener Normalform spezifiziert ist. Damit wir die Auswirkung der Aktion im jeweiligen Aktionenblock beobachten können, müssen wir den Algorithmus leicht modifizieren und für die herausführenden Übergänge von den

Blöcken zur Schleife eine kantenverbundene Konkatenation realisieren. Dazu führen wir in jedem Block einen weiteren Programmpunkt ein, diese seien n5′ und n6′. Die beiden Zeile des TVLA-Kontrollflussgraphen zu den Programmpunkten n5 und n6 ersetzen wir durch:

n5	Get_Sel_T(*cur*, *cur*, left)	n5′
n5′	concat()	n2
n6	Get_Sel_T(*cur*, *cur*, right)	n6′
n6′	concat()	n2

Hierbei bezeichnet concat() eine leere TVLA-Aktion, vergleiche Definition 4.4 auf Seite 77.

Zum Programmpunkt n5 gelangen wir, wenn wir in der Datenstruktur mit einer Verzweigung zum linken Kind fortfahren. Alle Shapegraphen, die beim Programmpunkt n5′ berechnet werden, zeichnen sich dadurch aus, dass der Strukturzeiger, der in die neue aktuelle Ecke cur hineinführt, ein left-Zeiger ist, dass also $\exists v_1, v_2 \colon \mathsf{cur}(v_2) \wedge \mathsf{left}(v_1, v_2)$ zu einem Wahrheitswert größer oder gleich $\frac{1}{2}$ auswertet und damit $\exists v_1, v_2 \colon \mathsf{cur}(v_2) \wedge \mathsf{right}(v_1, v_2)$ zum Wahrheitswert 0. Bei den Shapegraphen am Programmpunkt n6′ handelt es sich demgegenüber um solche, bei denen der Strukturzeiger, der in die neue aktuelle Ecke cur hineinführt, ein right-Zeiger ist. Der Shapegraph S tritt beim Programmpunkt n5′ auf, der Shapegraph T jedoch nicht. Dafür tritt T beim Programmpunkt n6′ auf, aber S dort nicht.

Nach diesen einleitenden Betrachtungen konkretisieren und formalisieren wir die Paarbildung. Hierzu bezeichne $\mathcal{M}$ eine Menge paarweise nicht isomorpher Shapegraphen, etwa die eines Programmpunktes. Wir setzen der Einfachheit halber wieder voraus, dass $\mathcal{M}$ nicht leer und endlich sei. Das Symmetriekonzept können wir dazu nutzen, um $\mathcal{M}$ eine zusätzliche Struktur aufzuprägen. Die auf diese Weise resultierende Menge bezeichnen wir mit $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$. Formal besteht sie aus einzelnen Shapegraphen sowie aus einelementigen und zweielementigen Mengen von Shapegraphen. Jeder Shapegraph aus $\mathcal{M}$ ist entweder als Shapegraph direkt in $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$ enthalten oder er ist Element genau einer der Teilmengen.

Für einen Shapegraphen $S \in \mathcal{M}$ kann es, wie die vorangegangenen Überlegungen gezeigt haben, einen Shapegraph $T \in \mathcal{M}$ mit $(S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ geben oder nicht. Man beachte, dass T nicht von S verschieden sein braucht. Im ersten Fall fassen wir S und T zu einer Menge zusammen und nehmen diese in $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$ auf, im zweiten Fall nehmen wir S direkt in $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$ auf. Es gelte also

$$\begin{aligned} \mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}} = &\big\{ S \colon \forall T \in \mathcal{M} \ (S,T) \notin \mathrm{sym}_{\mathcal{D}_\Sigma} \big\} \cup \\ &\big\{ \{S,T\} \colon \exists T \in \mathcal{M} \ (S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma} \big\}. \end{aligned} \tag{8.1}$$

Ist $S \in \mathcal{M}$ ein symmetrischer Shapegraph, gilt also $(S,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$, dann ist $\{S\} \in \mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$. Damit sind, wie wir es bereits angedeutet haben, die Elemente

von $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$ Shapegraphen sowie ein- und zweielementige Teilmengen von Shapegraphen von $\mathcal{M}$. – Wir fassen zusammen:

8.11 Definition. *Es sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer, des Weiteren sei $\mathcal{M}$ eine endliche und nichtleere Menge von paarweise nichtisomorphen Shapegraphen. Der Symmetriedefinierer $\mathcal{D}_\Sigma$ bedingt gemäß Gleichung* (8.1) *eine Struktur auf $\mathcal{M}$ wir nennen $\mathcal{M}_{\mathrm{sym}_{\mathcal{D}_\Sigma}}$ von $\mathcal{M}$ die durch $\mathcal{D}_\Sigma$ induzierte Strukturierung von $\mathcal{M}$.*

In Definition 8.7 haben wir Mengen mit einem hohen Grad an Symmetrie ausgezeichnet. Wir erweitern diesen Begriff auf Paare von Mengen:

8.12 Definition. *Es seien $\mathcal{M}$ und $\mathcal{N}$ zwei Mengen von Shapegraphen, und es sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Wir nennen das Paar $(\mathcal{M},\mathcal{N})$* symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$*, wenn es eine Bijektion $f\colon \mathcal{M} \to \mathcal{N}$ gibt, so dass für alle $S \in \mathcal{M}$ die Bedingung $(S, f(S)) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ gilt.*

Die in der Definition genannte Forderung ist äquivalent zu: Für alle $S \in \mathcal{M}$ existiert ein $T \in \mathcal{N}$ mit $(S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ und für alle $T \in \mathcal{N}$ gibt es ein $S \in \mathcal{M}$ mit $(T,S) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$. – Ein Beispiel: Wir betrachten die Suche nach einem Element in einem binären Suchbaum, wobei wir den erweiterten Algorithmus zugrunde legen. Wir verwenden die Basisspezifikation und die bekannten Startshapegraphen und führen die übliche Shapeanalyse durch. Als Symmetriedefinierer verwenden wir $\mathcal{D}_\Sigma = \big\{(\mathsf{left},\mathsf{right})\big\}$. Es bezeichne $\mathcal{M}$ die Menge der Shapegraphen am Programmpunkt n5′ und $\mathcal{N}$ die Menge der Shapegraphen am Programmpunkt n6′. Dann ist $(\mathcal{M},\mathcal{N})$ symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$.

In Kapitel 6 haben wir ein Ähnlichkeitskonzept für Shapegraphen eingeführt. Es gestattet, Mengen von Shapegraphen zu partitionieren, so dass Shapegraphen einer Klasse einander ähnlich sind. Handelt es sich bei den Mengen in Definition 8.12 um Klassen einer Partition, dann wollen wir diesen Fall mit einem eigenen Namen auszeichnen. Wir setzen:

8.13 Definition. *Es seien $\mathcal{C}_1$ und $\mathcal{C}_2$ zwei Klassen, der gemäß Satz 6.14 (Seite 125) hergestellten Partitionen der Shapegraphenmengen der Programmpunkte. Des Weiteren sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Die Klassen $\mathcal{C}_1$ und $\mathcal{C}_2$ heißen* zueinander symmetrische Klassen bezüglich $\mathcal{D}_\Sigma$*, wenn $(\mathcal{C}_1,\mathcal{C}_2)$ symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$ ist.*

Es sei darauf hingewiesen, dass in der Definition nicht vorausgesetzt wird, dass $\mathcal{C}_1$ und $\mathcal{C}_2$ zum selben Programmpunkt gehören müssen.

Der hier behandelte Ansatz zur Algorithmenvisualisierung basiert auf der Visualisierung der abstrakten Programmausführung. Dazu berechnen wir im Vorfeld der Visualisierung mittels einer statischen Programmanalyse Beschreibungen der auftretenden Heapsituationen. Das Problem, mit dem wir uns bei diesem Ansatz auseinandersetzen müssen, ist die Größe der Analyseausgabe, die eine direkte Visualisierung

erschwert. Wir begegnen dem Problem, indem wir die Analyseausgabe mit zusätzlicher Struktur versehen. Auf die so präparierte Ausgabe kann sich die eigentliche Visualisierung dann stützen. Das hier vorgestellte Symmetriekonzept ordnet sich in diesen Rahmen ein. In Kapitel 6 haben wir ein Ähnlichkeitskonzept für Shapegraphen behandelt. Auch dieses folgt der Idee, die Analyseausgabe zu strukturieren. Beide Methoden verhalten sich prinzipiell ähnlich. Beide Ansätze sind parametrisch. Durch Änderung eines Spezifikators kann die Strukturierung gesteuert werden. Der Spezifikator stützt sich in beiden Fällen direkt auf die (Prädikate der) Analysespezifikation. Beide Methoden gestatten eine qualitative Änderung der Strukturierung. Je nach den im Spezifikator aufgenommenen Prädikaten, kann die Strukturierung an anderen Eigenschaften (der Shapegraphen) ausgerichtet werden. Nur das Ähnlichkeitskonzept erlaubt die Quantität der strukturierten Ausgabe nahezu beliebig zu variieren, beim Symmetriekonzept bewegt sich der quantitative Aspekt auf dem Niveau von einzelnen Shapegraphen oder einelementigen Mengen von Shapegraphen bis zu zweielementigen Teilmengen.

In diesem Abschnitt haben wir einen Symmetriebegriff für Shapegraphen entwickelt und formalisiert. Er basiert auf der anschaulichen Vorstellung von Symmetrie in einem gezeichneten Graphen. Das Konzept erlaubt, eine Menge von Shapegraphen, etwa die eines Programmpunktes des zu visualisierenden Algorithmus, zu strukturieren. Das Symmetriekonzept ist parametrisch; es wird mittels eines Spezifikators gesteuert, der sich direkt auf die Prädikate der Analysespezifikation stützt. Durch die Strukturierung der Analyseausgabe kann die Visualisierung effektiver gestaltet und besser auf die Bedürfnisse eines Anwenders während der aktuellen Visualisierungssitzung zugeschnitten und abgestimmt werden.

8.2 Symmetrische Ausführungspfade

Im vorangegangenen Abschnitt haben wir ein Symmetriekonzept definiert und untersucht. Unser Augenmerk lag auf den Shapegraphen, es ging uns um zueinander symmetrische und symmetrische Shapegraphen. Mit dem erreichten Stand der Dinge wollen wir uns nicht begnügen. Wir sehen unseren Ansatz zur Algorithmenvisualisierung in einem weiten Sinne als ein Mittel zum Verständnisgewinn über Algorithmen. Das Studium eines Algorithmus sollte aber nicht mit der Untersuchung der an den einzelnen Programmpunkten auftretenden (Heap-)Situationen enden. Vielmehr ist es angeraten, auch Folgen von Aktionen, Sequenzen aufeinanderfolgender Situationen zu betrachten. Auf diese Weise erlangt man ein besseres Verständnis darüber, wie die Aktionen des Algorithmus bestehende Situationen in andere transformieren. Zu diesem Zweck übertragen wir das Symmetriekonzept auf Folgen von Aktionen.

In diesem Abschnitt geht es nicht mehr (primär) um die Strukturierung der Menge der Shapegraphen eines Programmpunktes. Wir wollen eine Sequenz von Aktionen

im TVLA-Programm und die zu ihr korrespondierenden Sequenzen von Shapegraphen untersuchen. Daher können wir uns nicht mit einer TVLA-Ausgabe in Form von Mengen von Shapegraphen begnügen. Wir benötigen für diese Betrachtung vielmehr einen Transitionsgraphen als Analyseausgabe, vergleiche Definition 3.7 auf Seite 54. Seine Ecken sind abstrakte Zustände, sie stellen sich formal als Paare aus einem Programmpunkt und einem Shapegraphen dar. Wenn klar ist, um welchen Programmpunkt es sich jeweils aktuell handelt, dann identifizieren wir den abstrakten Zustand gewöhnlich mit dem in ihm enthaltenen Shapegraphen. Die Kanten des Transitionsgraphen sind mit TVLA-Aktionen beschriftet, es sind die Aktionen aus dem TVLA-Programm. Wir verwenden TVLA-Programme in Normalform, so wie sie in Abschnitt 4.3 eingeführt wurden. Insbesondere sind die Programme deterministisch: Alle Kanten zwischen den Shapegraphen zweier Programmpunkte sind mit derselben TVLA-Aktion beschriftet. Deshalb können wir die Kantenbeschriftung in diesem Abschnitt unberücksichtigt lassen. (Diese Entscheidung wird sich später als nützliche Vereinfachung herausstellen.)

Symmetrie für Ausführungspfade

In dem diesen Abschnitt einleitenden Absatz haben wir von Sequenzen von Aktionen oder Situationen gesprochen. Damit haben wir nichts anderes als Kantenfolgen im TVLA-Programm beziehungsweise im Transitionsgraphen gemeint. Im vorangegangenen Abschnitt dieses Kapitels haben wir eine Symmetrievorstellung für Shapegraphen eingeführt. Wir haben erklärt, wann zwei Shapegraphen bezüglich eines vorgegebenen Symmetriedefinierers als zueinander symmetrisch anzusehen sind. In diesem Abschnitt geht es darum, diesen Symmetriebegriff auf Kantenfolgen im Transitionsgraphen zu übertragen. Jede dieser Kantenfolgen ist auch eine Kantenfolge im TVLA-Kontrollflussgraphen. Es ist zweckmäßig, wenn wir in der Erörterung dem strukturellen Aufbau von TVLA-Programmen folgen, wie er in Abschnitt 4.3 festgelegt wurde.

Als Erstes betrachten wir den Fall des Aktionenblocks. Er bestehe aus den Programmpunkten $n_1, n_2, \ldots, n_k$, wobei $k \geq 1$ sei. Wir betrachten zwei Wege W_1 und W_2 „durch den Block“, die beide in seiner Eingangsecke n_1 beginnen und in seiner Ausgangsecke n_k enden. Genaugenommen handelt es sich dabei natürlich um Wege im Transitionsgraphen. Die Programmpunkte und die Kantenbeschriftungen mit TVLA-Aktionen sind die des Aktionenblocks. Deshalb haben wir diese sinnbildliche (aber nicht ganz korrekte) Bezeichnung verwendet. Der Weg W_1 verlaufe entlang der Ecken $(n_1, S_1), (n_2, S_2), \ldots, (n_k, S_k)$, der Weg W_2 entlang $(n_1, T_1), (n_2, T_2), \ldots, (n_k, T_k)$. Wir setzen hierbei nicht voraus, dass für $1 \leq i \leq k$ die Shapegraphen S_i und T_i verschieden sein müssen. Aufgrund der Voraussetzungen gelten zwei Dinge, die wir explizit herausstellen wollen: 1. Für jedes i, $1 \leq i \leq k$, gehören die Shapegraphen S_i und T_i zum selben Programmpunkt. 2. Für jedes i, $1 \leq i \leq k-1$, haben die Kanten $[(n_i, S_i), (n_{i+1}, S_{i+1})\rangle$ und $[(n_i, T_i), (n_{i+1}, T_{i+1})\rangle$ dieselbe TVLA-Aktion als Beschriftung. (Die Programme sind in Normalform.) Damit

weichen beide Wege in struktureller Hinsicht nicht stark voneinander ab. Wir erklären nun, wann sie zueinander symmetrisch seien. Dazu sei ein Symmetriedefinierer $\mathcal{D}_\Sigma$ vorgegeben. Die Wege W_1 und W_2 seien *zueinander symmetrisch bezüglich* $\mathcal{D}_\Sigma$, wenn für jedes i, $1 \leq i \leq k$, die Shapegraphen S_i und T_i zueinander symmetrisch bezüglich $\mathcal{D}_\Sigma$ sind.

Wir notieren einige Eigenschaften. Für das Folgende sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer und es seien W_1 und W_2 zwei zueinander symmetrische Wege mit jeweils $k \geq 1$ Ecken.

1. Ist $k = 1$, dann sind W_1 und W_2 zueinander symmetrisch – sie bestehen jeweils nur aus einer Ecke –, wenn der einzige Shapegraph von W_1 zu dem einzigen Shapegraphen von W_2 symmetrisch ist. Der Symmetriebegriff für Wege ist damit eine Erweiterung des Symmetriebegriffs für Shapegraphen.
2. Gilt $S_i = T_i$ für ein $1 \leq i \leq k$, dann sind die Shapegraphen S_i und T_i wegen $(S_i, T_i) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ symmetrisch.
3. Gilt für jedes i, $1 \leq i \leq k$, dass der Shapegraph S_i (beziehungsweise T_i) nicht symmetrisch ist, dann folgt $S_i \neq T_i$. In diesem Fall sind die Wege W_1 und W_2 disjunkt.

Als Zweites betrachten wir den Fall einer einfachen Verzweigung. Sie zeichnet sich dadurch aus, dass die Äste – das sind die Blöcke, in die verzweigt wird – Aktionenblöcke sind. Die Wege, die wir uns im Folgenden vorstellen, mögen im Eingangspunkt der Verzweigung starten und in ihrem Ausgangspunkt enden. Wir können bei der Verzweigung konkret an die Verzweigung in unserem stets verwendeten Suchbeispiel denken. Die Aktionenblöcke bestehen jeweils nur aus einem Programmpunkt. Damit wir die Auswirkung der Aktionen im jeweiligen Block untersuchen können, verwenden wir das modifizierte TVLA-Programm aus Abschnitt 8.1, wo wir in jedem Aktionenblock einen zusätzlichen Programmpunkt, n5′ beziehungsweise n6′, eingefügt haben. Je nachdem wie der Vergleich der Datenwerte des aktuellen Elementes mit dem gesuchten Element ausfällt, verzweigen wir vom aktuellen Baumelement zu seinem linken oder rechten Kind. Wir haben schon festgestellt, dass Shapegraphen, die am Programmpunkt n5′ auftreten, keinen symmetrischen Partner bei n5′ haben, wohl aber bei n6′. (Wir betrachten Symmetrie bezüglich left und right.) Ebenso haben Shapegraphen am Programmpunkt n6′ keinen symmetrischen Partner an diesem Programmpunkt, wohl aber bei n5′. Wenn wir einen Weg durch die Verzweigung vorliegen haben, dann wird ein dazu symmetrischer Weg durch einen anderen Ast der Verzweigung verlaufen. Korrespondierende Ecken werden zu verschiedenen Programmpunkten gehören. Damit sind normalerweise auch die Kanten mit verschiedenen TVLA-Aktionen beschriftet. Bei der Definition von zueinander symmetrischen Wegen müssen wir, wenn eine Verzweigung involviert ist, im Vergleich zum Aktionenblock die Voraussetzungen abschwächen. Wir werden uns dafür entscheiden, die Bedingung an die Programmpunkte fallen zu lassen.

Als Drittes betrachten wir eine einfache Schleife. Sie zeichnet sich dadurch aus, dass der Schleifenrumpf ein Aktionenblock ist. Hier haben wir es in der Regel nicht mehr mit Wegen zu tun, wir können nur noch allgemein von Kantenzügen sprechen. Wenn wir (im Transitionsgraphen) einen Kantenzug durch die Programmpunkte einer Schleife betrachten, der in der Schleifeneingangsecke beginnt und in seiner Ausgangsecke endet, dann können durch einen eventuellen Rücksprung vom Ausgangsblock zum Eingangsblock des Schleifenrahmens Kreise entstehen. Der Kantenzug braucht nicht eckendisjunkt zu sein. – Abschließend betrachten wir den Fall des TVLA-Programms. Es entsteht durch Konkatenation von Aktionenblöcken, Verzweigungen und Schleifen, wobei Verzweigungen und Schleifen anstelle der Aktionenblöcke TVLA-Programme enthalten dürfen. Hier verhält es sich ebenso wie bei Schleifen: Wir können nur allgemein von Kantenzügen ausgehen.

Nachdem wir die Idee von zueinander symmetrischen Kantenzügen elaboriert haben, formalisieren wir sie:

8.14 Definition. *Es sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Des Weiteren sei K_1 ein Kantenzug im Transitionsgraphen entlang der Ecken $(m_1, S_1), (m_2, S_2), \ldots, (m_k, S_k)$ und K_2 ein Kantenzug entlang der Ecken $(n_1, T_1), (n_2, T_2), \ldots, (n_k, T_k)$, wobei $k \geq 1$ sei. Wir nennen K_1 und K_2* zueinander symmetrisch bezüglich $\mathcal{D}_\Sigma$, *wenn $(S_i, T_i) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ für jedes i, $1 \leq i \leq k$, gilt.*

Betrachten wir als Beispiel erneut die Suche nach einem Element in einem binären Suchbaum. Das TVLA-Programm ist in Abbildung 4.3 auf Seite 66 gezeigt, wobei wir wie oben die Modifikation aus Abschnitt 8.1 zugrunde legen. In den Abbildungen 8.4 und 8.5 sehen wir zwei bezüglich left und right zueinander symmetrische Kantenzüge durch den Transitionsgraphen, jede der Spalten zeigt einen Kantenzug. Beide Kantenzüge bestehen jeweils aus 23 Ecken, der linke beispielsweise aus: n1, n2, n3, n4, n5, n5' n2, n3, n4, n5, n5' n2, n3, n4, n5, n5' n2, n3, n4, n5, n5', n2, found. Zwischen den Programmpunkten n2, n3, n4 und n5 beziehungsweise n6 finden nur Vergleiche statt. Die entsprechenden TVLA-Aktionen führen zu keiner Veränderung am Shapegraphen. Daher wurden aufeinanderfolgende Ecken des Kantenzuges mit gleichen Shapegraphen zusammengefasst dargestellt. Die Programmpunkte, zu denen ein Shapegraph gehört, sind jeweils mit angegeben. Beide Kantenzüge beginnen im Startshapegraphen der Analyse und enden im Endprogrammzustand „notfound“ des TVLA-Programms. Die Suchschleife wird jeweils vier Mal durchlaufen.

Wenn wir zwei Wege im Transitionsgraphen auf Symmetrie hin untersuchen, einer verlaufe entlang der Ecken $(m_1, S_1), (m_2, S_2), \ldots, (m_k, S_k)$ und der andere entlang $(n_1, T_1), (n_2, T_2), \ldots, (n_k, T_k)$, dabei sei $k \geq 1$, dann muss für jedes i, $1 \leq i \leq k$, die Bedingung $(S_i, T_i) \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ erfüllt sein. Diese Forderung kann unter Umständen bei einigen Anwendungen zu restriktiv sein. Stellen wir uns zwei Wege vor, die vom Startprogrammpunkt zum Endprogrammpunkt verlaufen. Beide Wege beginnen mit

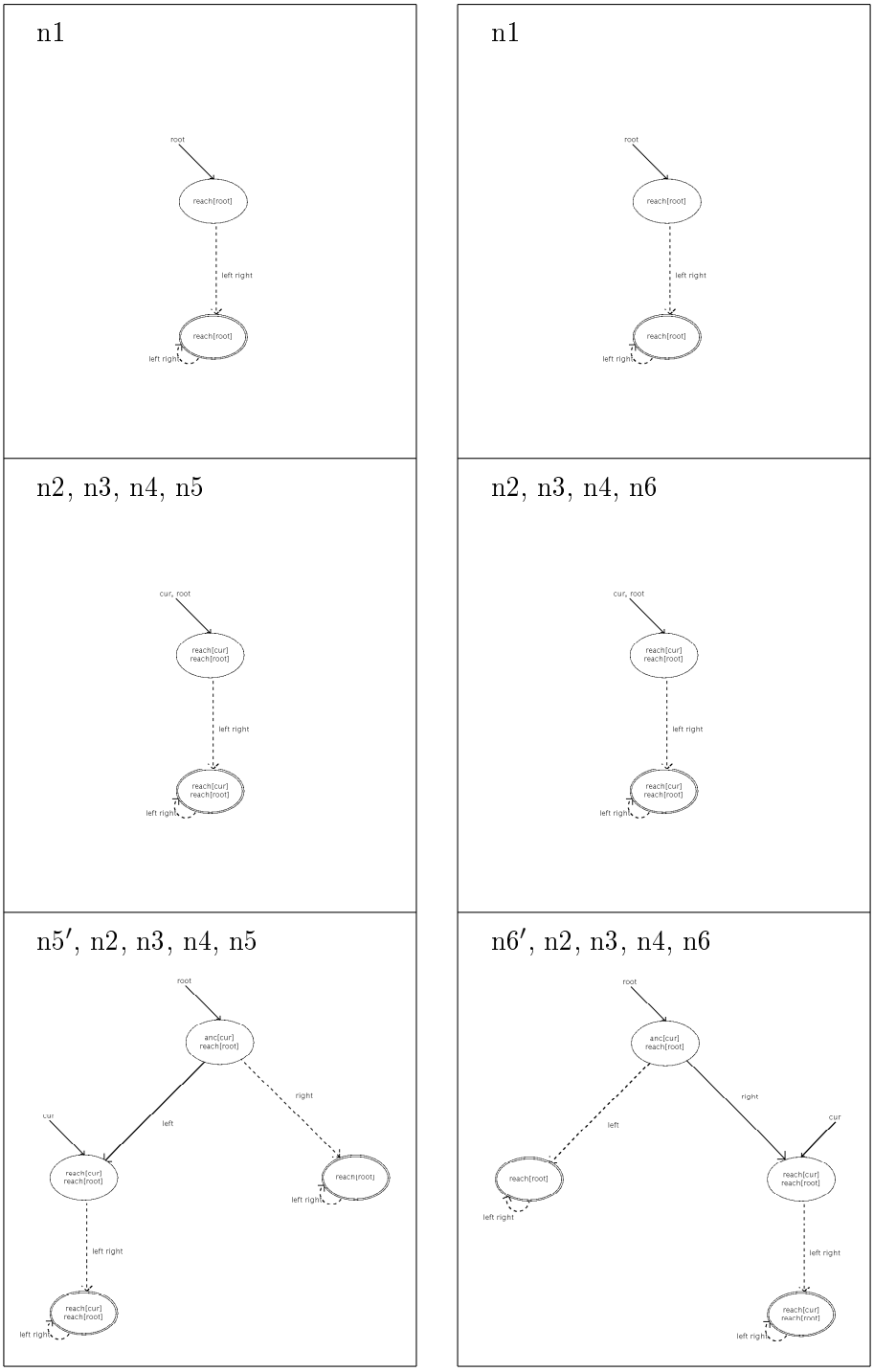

Abbildung 8.4: Zwei bezüglich left und right zueinander symmetrische Kantenzüge im Transitionsgraphen, Teil 1

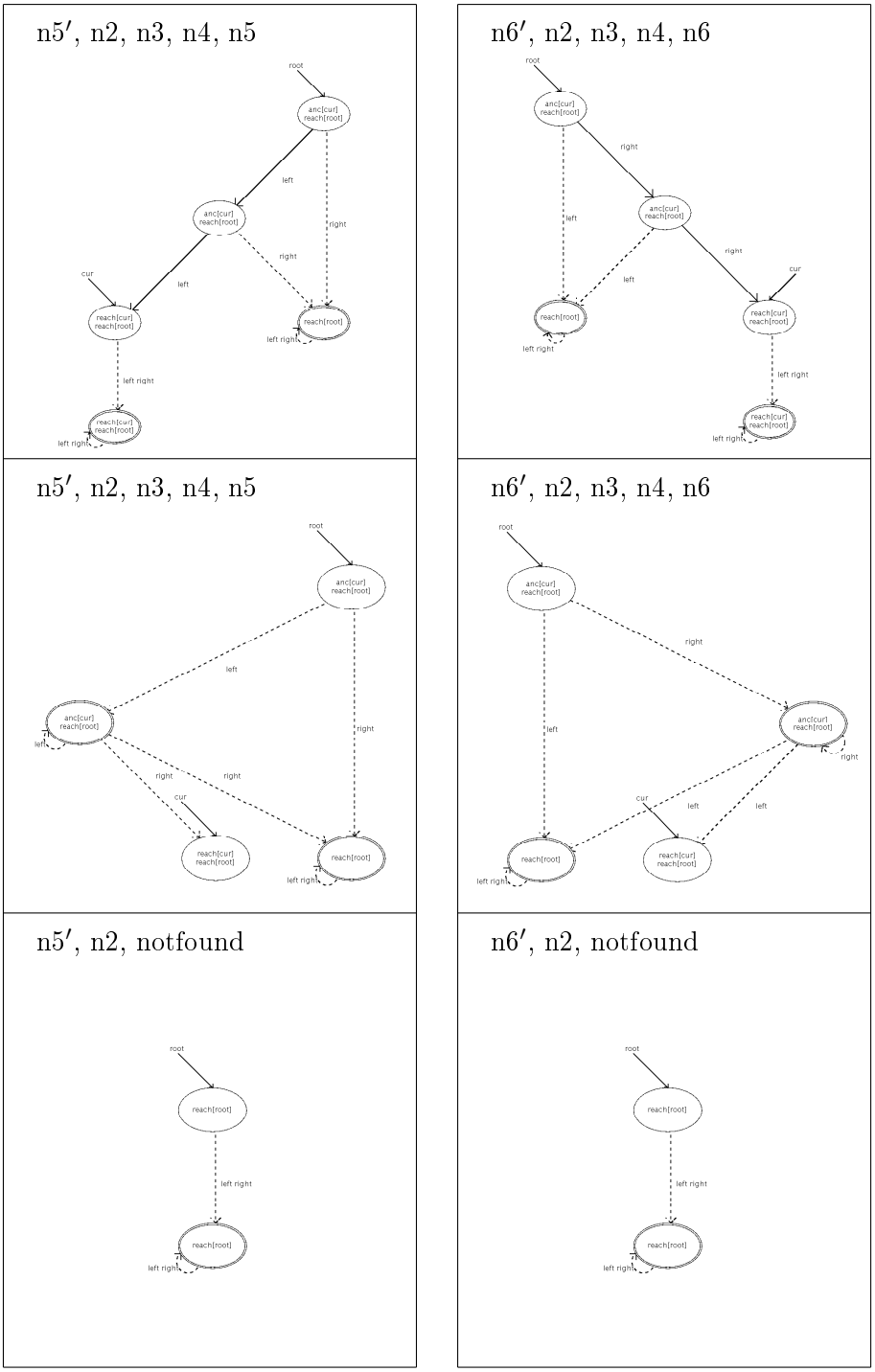

Abbildung 8.5: Zwei bezüglich left und right zueinander symmetrische Kantenzüge im Transitionsgraphen, Teil 2

einem der für die Analyse vorgegebenen Startshapegraphen, vielleicht sogar mit demselben. Wir haben nichts über die Analysespezifikation (oder den Algorithmus) gesagt, aber es gibt keine zwingenden Gründe anzunehmen, dass diese Shapegraphen zueinander symmetrisch (beziehungsweise symmetrisch) bezüglich eines gegebenen Symmetriedefinierer $\mathcal{D}_\Sigma$ sein müssen. Es könnte beispielsweise mit einer der ersten TVLA-Aktionen eine „Initialisierung“ stattfinden, in deren Folge dann erst, gewissermaßen als „zufälliges“ Nebenresultat, Symmetrie herrscht. Ebenso wäre es denkbar, dass die Shapegraphen bei Programmendpunkten die Symmetrieforderung verletzen können. In einem solchen Fall wird es sinnvoll sein, die Definition abzuschwächen. Wir würden dann auf eine Symmetrieforderung zwischen S_1 und T_1, und gegebenenfalls auch zwischen S_k und T_k, verzichtet.

Im weiteren Verlauf der Untersuchung werden wir diese Abschwächung aber nicht behandeln. Zum einen wollen wir der Klarheit der Darstellung wegen auf eine Extrabehandlung der Start- und Endprogrammpunkte verzichten. Zum anderen tritt der Fall bei den in dieser Arbeit hauptsächlich als Beispiele herangezogenen Baumalgorithmen und der Betrachtung von Symmetrie bezüglich (left, right) nicht auf. Die verwendeten Startshapegraphen sind symmetrisch, vergleiche Abbildung 8.3. Denken wir konkret an den Suchalgorithmus. Bei einer erfolglosen Suche ist der Shapegraph des Programmende-Programmpunktes „notfound“ entweder der linke Shapegraph aus Abbildung 8.3, dieser ist symmetrisch, oder er ist ihm „sehr ähnlich“. Die Summary-Ecke kann eine Individuenecke sein; anstelle der Kanten, die sowohl mit left als auch mit right beschriftet sind, kann nur eine dieser Beschriftungen vorliegen – es sind aber nicht alle Kombinationen möglich. Diese Shapegraphen besitzen alle einen symmetrischen Partner am Programmpunkt „notfound“. Bei einer erfolgreichen Suche endet das Programm am Programmpunkt „found“ mit einem Shapegraphen, der einen symmetrischen Partner an diesem Programmpunkt besitzt.

Symmetrieabgeschlossenheit für Ausführungspfade

Als wir uns in Abschnitt 8.1 mit zueinander symmetrischen Shapegraphen beschäftigten, haben wir in Definition 8.7 Shapegraphenmengen mit einem besonders hohen Grad an Symmetrie als symmetrieabgeschlossen herausgestellt. In Definition 8.12 haben wir diesen Begriff auf Paare von Mengen erweitert. In diesem Abschnitt beschäftigen wir uns mit Kantenzügen (im Transitionsgraphen). Deshalb wollen wir den Begriff symmetrieabgeschlossen von Shapegraphen(mengen) auf „Kanten“ übertragen.

Für die folgende Diskussion legen wir den Transitionsgraphen als Analyseausgabe zugrunde. Ein Symmetriedefinierer $\mathcal{D}_\Sigma$ sei spezifiziert. Wir betrachten zwei Mengen $\mathcal{M}$ und $\mathcal{N}$ von abstrakten Zuständen, die jeweils zum gleichen Programmpunkt m beziehungsweise n gehören. Gemäß unseren Voraussetzungen sind alle Kanten des Transitionsgraphen von Ecken aus $\mathcal{M}$ zu Ecken aus $\mathcal{N}$ mit der gleichen TVLA-Aktion beschriftet. Der einfacheren Darstellung wegen identifizieren wir einen abstrakten

Zustand mit seinem Shapegraphen. Da wir einen hohen Grad von Symmetrie begrifflich fassen wollen, soll dieser hohe Grad auch für die Shapegraphenmengen erfüllt sein. Wir fordern, dass sowohl $\mathcal{M}$ als auch $\mathcal{N}$ symmetrieabgeschlossen seien. Des Weiteren benötigen wir eine Bedingung, die sich auf Kanten bezieht. Es sei $S \in \mathcal{M}$ und $T \in \mathcal{N}$ beliebig. Der symmetrische Partner von S in $\mathcal{M}$ sei mit S' bezeichnet und derjenige von T in $\mathcal{N}$ mit T'. Weiter fordern wir, dass die Kante $[S,T\rangle$ genau dann (im Transitionsgraphen) existiere, wenn die Kante $[S',T'\rangle$ existiert. (Man beachte, dass S und T für abstrakte Zustände stehen; $[S,T\rangle$ bezeichnet also eine Kante vom Programmpunkt m zum Programmpunkt n.) Diese Bedingung gelte selbstverständlich für jedes $S \in \mathcal{M}$, $T \in \mathcal{N}$. Wir sagen dann, dass $[\mathcal{M},\mathcal{N}\rangle$ *symmetrieabgeschlossen bezüglich* $\mathcal{D}_\Sigma$ sei. Wir verzichten darauf, dies sogleich in einer Definition herauszuheben, stattdessen verallgemeinern wir den Begriff.

In Verzweigungen können zueinander symmetrische Wege durch verschiedene Äste verlaufen. Daher müssen wir den Begriff entsprechend erweitern. Es gelten die Voraussetzungen wie oben. Anstelle von zwei Shapegraphenmengen – eigentlich handelt es sich um Mengen von abstrakten Zuständen – betrachten wir jetzt derer vier. Sie seien $\mathcal{M}$, $\mathcal{M}'$, $\mathcal{N}$ und $\mathcal{N}'$, wobei es sich bei jeder Menge wieder um Shapegraphen (abstrakte Zustände) desselben Programmpunktes handele. Als Erstes fordern wir, dass $(\mathcal{M},\mathcal{M}')$ und $(\mathcal{N},\mathcal{N}')$ symmetrieabgeschlossen (bezüglich $\mathcal{D}_\Sigma$) seien. Die zweite Bedingung bezieht sich wieder auf die Kanten. Für jedes $S \in \mathcal{M}$, sein symmetrischer Partner in $\mathcal{M}'$ sei S', und jedes $T \in \mathcal{N}$, sein symmetrischer Partner in $\mathcal{N}'$ sei T', gelte, dass die Kante $[S,T\rangle$ genau dann existiere, wenn $[S',T'\rangle$ existiert. – Wir fassen dies in einer Definition zusammen:

8.15 Definition. *Es sei $\mathcal{TG}$ ein Transitionsgraph und $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Des Weiteren sei $\mathcal{M}$ eine Teilmenge der Shapegraphen des Programmpunktes m, $\mathcal{M}'$ eine Teilmenge der Shapegraphenmenge von m', $\mathcal{N}$ ein Teilmenge der Shapegraphenmenge von n und $\mathcal{N}'$ eine Teilmenge der Shapegraphenmenge von n'. Wir nennen $[(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle$* symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$ *genau dann, wenn:*

1. *Die Paare $(\mathcal{M},\mathcal{N})$ und $(\mathcal{M}',\mathcal{N}')$ sind symmetrieabgeschlossen bezüglich $\mathcal{D}_\Sigma$.*
2. *Für alle Shapegraphen $S \in \mathcal{M}$, $S' \in \mathcal{M}'$, $T \in \mathcal{N}$ und $T' \in \mathcal{N}'$ mit $(S,S') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ und $(T,T') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ gilt $[S,T\rangle \in \mathcal{TG} \Leftrightarrow [S',T'\rangle \in \mathcal{TG}$. (Hierbei bezeichnet $[S,T\rangle$ eine Kante vom Programmpunkt m zum Programmpunkt n und entsprechend $[S',T'\rangle$ von m' nach n'.)*

Betrachten wir als Beispiel den Suchalgorithmus in der erweiterten Darstellung mit den Programmpunkten n5′ und n6′. Wir legen die Basisspezifikation, die bekannten Startshapegraphen und die übliche Shapeanalyse zugrunde. Symmetrie betrachten wir bezüglich left und right. Wählen wir für $\mathcal{M}$ die Shapegraphenmenge des Programmpunktes n5, für $\mathcal{M}'$ die des Programmpunktes n6, für $\mathcal{N}$ die von n5′ und für $\mathcal{N}'$ die von n6′, dann ist $[(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle$ symmetrieabgeschlossen.

Was bedeutet dieser Begriff nun für Kantenzüge im Transitionsgraphen? Wir betrachten einen Kantenzug entlang der Programmpunkte $m_1, m_2, \ldots, m_k$, $k \geq 1$, und einen dazu symmetrischen Kantenzug entlang der Programmpunkte $n_1, n_2, \ldots, n_k$. Die Programmpunkte m_i und n_i können für $1 \leq i \leq k$ durchaus gleich sein. Für $1 \leq i \leq k$ sei nun $\mathcal{M}_i$ eine Teilmenge, der am Programmpunkt m_i auftretenden Shapegraphen, und $\mathcal{N}_i$ eine Teilmenge, der am Programmpunkt n_i vorkommenden. Für $1 \leq i \leq k-1$ sei $[(\mathcal{M}_i, \mathcal{M}'_i), (\mathcal{N}_i, \mathcal{N}'_i)\rangle$ symmetrieabgeschlossen. Dann gibt es zu jedem Kantenzug entlang der Programmpunkte $m_1, m_2, \ldots, m_k$, durch die Shapegraphenmengen $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_k$ einen dazu symmetrischen Kantenzug entlang der Programmpunkte $n_1, n_2, \ldots, n_k$, durch die Shapegraphenmengen $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$; das gilt auch umgekehrt. Wir haben es dann mit zwei zueinander symmetrischen *Strängen* von Kantenzügen zu tun. Definition 8.15 erlaubt es also, die Betrachtung von zueinander symmetrischen Kantenzügen auf Mengen von zueinander symmetrischen Kantenzügen (durch dieselben Programmpunkte) auszudehnen.

Dieser Sachverhalt lässt sich für die Visualisierung nutzen. Zueinander symmetrische Shapegraphen (und zueinander symmetrische Kantenzüge) beschreiben zueinander symmetrische Situationen (und Folgen von zueinander symmetrischen Situationen). Um einen Algorithmus zu verstehen, ist es daher (häufig) ausreichend, nur eine dieser zueinander symmetrischen (Folgen von) Situationen zu untersuchen. Von zwei zueinander symmetrischen Strängen brauchen wir nur einen zu visualisieren und können beispielsweise den zweiten aus dem Transitionsgraphen herausfiltern oder ausblenden. Die Symmetrieabgeschlossenheit aus Definition 8.15 ist eine Kohärenzbedingung. Sie sichert, dass die Stränge gleichartig sind, dass zwischen ihnen keine Unstimmigkeiten vorherrschen.

Betrachten wir wieder die Suche nach einem Element in einem binären Suchbaum, wir legen den erweiterten Algorithmus zugrunde und verwenden die übliche Analyse. Für einen Programmpunkt n bezeichne $\mathcal{M}_{\mathrm{n}}$ die Shapegraphenmenge des Programmpunktes. Wir betrachten einerseits den Strang, die Menge der Kantenzüge entlang Shapegraphen aus $\mathcal{M}_{\mathrm{n1}}, \mathcal{M}_{\mathrm{n2}}, \mathcal{M}_{\mathrm{n3}}, \mathcal{M}_{\mathrm{n4}}, \mathcal{M}_{\mathrm{n5}}, \mathcal{M}_{\mathrm{n5'}}, \mathcal{M}_{\mathrm{n2}}, \mathcal{M}_{\mathrm{n3}}, \mathcal{M}_{\mathrm{found}}$ und andererseits den Strang entlang Shapegraphen aus $\mathcal{M}_{\mathrm{n1}}, \mathcal{M}_{\mathrm{n2}}, \mathcal{M}_{\mathrm{n3}}, \mathcal{M}_{\mathrm{n4}}, \mathcal{M}_{\mathrm{n6}}, \mathcal{M}_{\mathrm{n6'}}$, $\mathcal{M}_{\mathrm{n2}}, \mathcal{M}_{\mathrm{n3}}, \mathcal{M}_{\mathrm{found}}$. Für diese Mengen ist Definition 8.15 erfüllt. Beide Stränge sind zueinander symmetrisch.

Wir sind aber nicht darauf beschränkt, jeweils die ganze Shapegraphenmenge eines Programmpunktes zu wählen. In Kapitel 6 haben wir ein Ähnlichkeitskonzept für Shapegraphen entwickelt. Es gestattet, eine Menge von Shapegraphen gemäß eines spezifizierten Ähnlichkeitsbegriffs zu partitionieren, so dass alle Shapegraphen innerhalb einer Klasse einander ähnlich sind. Wir können bei der Symmetriebetrachtung für die Mengen auch Klassen der Partitionen (der Shapegraphenmengen der Programmpunkte) wählen. Liegt jeweils Symmetrieabgeschlossenheit gemäß Definition 8.15 vor, dann haben wir es mit zueinander symmetrischen Strängen zu tun,

die durch die verschiedenen Klassen verlaufen. Kantenzüge im selben Strang sind einander ähnlich, vergleiche Abschnitt 7.2.

Als wir anfingen, den Symmetriebegriff von Shapegraphen auf Kantenzüge zu übertragen, hatten wir uns entschieden, die Beschriftungen der Kanten mit TVLA-Aktionen nicht zu berücksichtigen. Zueinander symmetrische Kantenzüge durch verschiedene Äste einer Verzweigung durchlaufen naturgemäß unterschiedliche Programmpunkte. Damit können korrespondierende Kanten in zueinander symmetrischen Kantenzügen mit unterschiedlichen TVLA-Aktionen beschriftet sein. Hätten wir die TVLA-Aktionen bei der Entwicklung des Konzepte mitberücksichtigen wollen, dann hätten wir Paare von Aktionen als zueinander symmetrisch klassifizieren können und dann zusätzlich verlangen müssen, dass im Falle verschiedener Programmpunkte die Kanten mit zueinander symmetrischen TVLA-Aktionen beschriftet sind. Eine solch zusätzliche Forderung würde die Anwendung des Symmetriekonzepts erschweren. Die entwickelte Begriffsbildung erlaubt demgegenüber, den umgekehrten Weg zu gehen. Wir betrachten die Shapegraphenmengen $\mathcal{M}$, $\mathcal{M}'$, $\mathcal{N}$ und $\mathcal{N}'$ von vier Programmpunkten. Nach Voraussetzung haben alle Kanten von Shapegraphen aus $\mathcal{M}$ zu Shapegraphen aus $\mathcal{N}$ dieselbe Beschriftung mit einer TVLA-Aktion „aktion1“, ebenso haben alle Kanten von $\mathcal{M}'$ nach $\mathcal{N}'$ dieselbe Beschriftung „aktion2“. Ist nun $[(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle$ symmetrieabgeschlossen bezüglich eines spezifizierten Symmetriedefinierers, dann können wir „aktion1“ und „aktion2“ als *zueinander symmetrische Aktionen* für die zugehörigen Programmpunkte auffassen. Gilt dies für alle relevanten Programmpunkte, zwischen denen die Kantenbeschriftungen „aktion1“ und „aktion2“ vorkommen, dann handelt es sich um zueinander symmetrische Aktionen für das ganze Programm (bezüglich der verwendeten Shapeanalyse). Nicht nur, dass wir anhand der Untersuchung zueinander symmetrischer Kantenzüge Aussagen über die Symmetrie im TVLA-Programm erhalten können, sie erlaubt uns ferner Rückschlüsse auf in der Analysespezifikation enthaltener Symmetrie anzustellen. Auf eine explizite Definition von zueinander symmetrischen Aktionen verzichten wir.

Dieses Kapitel handelt von Symmetrieeigenschaften in Bezug auf Shapegraphen. In Abschnitt 8.1 haben wir ein Symmetriebegriff für Shapegraphen eingeführt und untersucht. Auch haben wir ihn auf Mengen von Shapegraphen erweitert. In diesem Abschnitt ging es darum, dieses Symmetriekonzept auf „Kanten“ im Transitionsgraph zu übertragen. Shapegraphen beschreiben einzelne (Heap-)Situationen. Wir wollen aber im Zuge der Visualisierung dem Programmfluss folgen und mithin Folgen von Situationen beobachten. Diesen Sequenzen im TVLA-Kontrollflussgraphen entsprechen (Mengen von) Kantenzügen im von der Shapeanalyse berechneten Transitionsgraphen. Wir sind dem strukturellen Aufbau von TVLA-Programmen gefolgt und haben die entstehenden Kantenzüge beschrieben. Darauf aufbauend haben wir zueinander symmetrische Kantenzüge definiert. Wie für Shapegraphen haben wir auch für „Kantenmengen“ den Begriff der Symmetrieabgeschlossenheit eingeführt, um einen besonders hohen Grad an Symmetrieeigenschaften auszuzeichnen. Ist sie erfüllt, dann können Kantenzüge zu Strängen zusammengefasst werden. In der Vi-

sualisierung können dann beispielsweise jeweils die symmetrischen Partnerstränge unberücksichtigt bleiben. Die Untersuchung dieser Symmetrieeigenschaften in einem Transitionsgraphen für einen Algorithmus erlaubt also (bis zu einem gewissen Grad) Aussagen über die im TVLA-Programm innewohnende Symmetrie anzustellen. Des Weiteren erlaubt die Begriffsbildung, wir haben hierzu zueinander symmetrische TVLA-Aktionen ausgezeichnet, Rückschlüsse auf die in der Analysespezifikation immanente Symmetrie anzustellen.

8.3 Symmetriemaße

Dieses Kapitel handelt von Symmetriebetrachtungen bei Shapegraphen. Zueinander symmetrische Shapegraphen beschreiben zueinander symmetrische Heapsituationen. Hat ein Anwender eine dieser Situationen verstanden, dann erschließen sich die dazu symmetrischen in der Regel leicht und können bei der Visualisierung zum Beispiel herausgefiltert werden. Im ersten Abschnitt des Kapitels haben wir das Konzept eingeführt, wir haben den Begriff der zueinander symmetrischen Shapegraphen definiert und untersucht. Im zweiten Abschnitt haben wir den Symmetriebegriff auf Kantenzüge im Transitionsgraphen erweitert. Die Behandlung des Symmetriekonzepts erfolgte dabei jeweils auf einer qualitativen Ebene.

Bei der Anwendung offenbaren sich zwei Problembereiche. Zum einen sind die Prädikate der Analysespezifikation hinsichtlich des Symmetriebegriffs nicht unabhängig voneinander. Wir haben in Abschnitt 8.1 anhand der dort betrachteten Beispiele gesehen, dass bei einer Symmetriebetrachtung hinsichtlich (left, right) auch Prädikate, die Ordnungseigenschaften beschreiben, bei der Spezifikation des Symmetriedefinierers mitberücksichtigt werden müssen. Zum anderen braucht in der Analyseausgabe keine oder nur wenig Symmetrie bezüglich des gewählten Symmetriedefinierers vorzuliegen. In beiden Fällen wünscht man einen Mechanismus, um beurteilen zu können, inwieweit der Symmetriedefinierer glücklich gewählt und nutzbar ist. Deshalb fügen wir der Untersuchung an dieser Stelle einen quantitativen Aspekt hinzu. Wir wollen eine *Messende*, ein Maß, für den Grad der Symmetrie einer Shapegraphenmenge beziehungsweise für den Grad der Symmetrie zwischen zwei Shapegraphenmengen. Mit ihrer Hilfe können wir die Frage beantworten, ob für einen gegebenen Symmetriedefinierer überhaupt Symmetrie in der Shapeanalyseausgabe vorliegt und wie stark sie ausgeprägt ist.

Symmetriemessende für Shapegraphenmengen

Wir folgen der Darstellung in Abschnitt 8.1. Zunächst betrachten wir die Situation für einen Programmpunkt einer Anweisungsfolge. Es geht hier also um Symmetrieeigenschaften einer Shapegraphenmenge. Danach betrachten wir den Fall einer Verzweigung. Der zu einem Shapegraphen bei einem Programmpunkt im Then-Teil der

Verzweigung symmetrische Shapegraph kann beim korrespondierenden Programmpunkt im Else-Teil vorkommen. Hier betrachten wir mithin Symmetrieeigenschaften zwischen Shapegraphenmengen. Beginnen wir also als Erstes mit der Behandlung der Symmetrieeigenschaften einer Shapegraphenmenge. Für einen Teil ihrer Shapegraphen existiert ein symmetrischer Partner in dieser Menge, für andere nicht. Dies erfassen wir quantitativ mittels:

8.16 Definition. *Es sei $\mathcal{M}$ eine endliche und nichtleere Menge von Shapegraphen, des Weiteren sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Wir setzen*

$$\mu_{\mathcal{D}_\Sigma}(\mathcal{M}) = \frac{\left|\{S \in \mathcal{M} \colon \exists T \in \mathcal{M} \ (S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}\}\right|}{|\mathcal{M}|} \tag{8.2}$$

und nennen $\mu_{\mathcal{D}_\Sigma}$ eine Symmetriemessende *bezüglich $\mathcal{D}_\Sigma$ für $\mathcal{M}$.*

Wenn der Zusammenhang zum Symmetriedefinierer klar ist und keine Unklarheiten oder Verwechslungen zu befürchten sind, dann schreiben wir wie auch sonst anstelle von $\mu_{\mathcal{D}_\Sigma}$ kürzer μ. – Bezüglich des trivialen Symmetriedefinierers $\mathcal{D}_\Sigma = \emptyset$ ist jeder Shapegraph symmetrisch. Daher gilt für jede nichtleere Shapegraphenmenge $\mathcal{M}$ die Aussage $\mu_\emptyset(\mathcal{M}) = 1$. Für eine Menge $\mathcal{M}$ von Shapegraphen gilt $\mu(\mathcal{M}) = 1$ genau dann, wenn sie symmetrieabgeschlossen ist, vergleiche Definition 8.7.

Im Zähler von Gleichung (8.2) zählen wir die Anzahl der Shapegraphen aus $\mathcal{M}$, die einen symmetrischen Partner in $\mathcal{M}$ haben. Die von $\mathrm{sym}_{\mathcal{D}_\Sigma}$ induzierte Strukturierung einer Menge $\mathcal{M}$, vergleiche Definition 8.11, besteht aus einzelnen Shapegraphen, sie haben keinen symmetrischen Partner in $\mathcal{M}$, einelementigen Mengen von symmetrischen Shapegraphen und zweielementigen Mengen von zueinander symmetrischen Shapegraphen. Wenn wir uns die Menge $\mathcal{M}$ so strukturiert denken, dann werden in Gleichung (8.2) die einzelnen Shapegraphen nicht gezählt, die einelementigen Mengen werden einfach und die zweielementigen Mengen werden zweifach gezählt. Der Zähler in Gleichung (8.2) ist eine nichtnegative, der Nenner eine positive ganze Zahl; zudem ist der Zähler nicht größer als der Nenner. Somit ist der Bildbereich von $\mu_{\mathcal{D}_\Sigma}$ die Menge der rationalen Zahlen im Intervall $[0, 1]$. Wegen der Normierung ist $\mu_{\mathcal{D}_\Sigma}(\mathcal{M})$ folglich der Anteil der Shapegraphen aus $\mathcal{M}$ mit einem symmetrischen Partner in $\mathcal{M}$. Damit ist die Messende μ eine der einfachsten denkbaren quantitativen Beschreibungen für den Grad der Ausprägung der Symmetrie innerhalb einer Shapegraphenmenge. Doch reicht sie für unsere Zwecke aus, so dass wir davon Abstand nehmen können, andere Funktionen zu diskutieren.

Bisher haben wir die Symmetrieeigenschaften (innerhalb) einer Menge quantitativ beurteilt. Die Mengen, die wir im Blick haben, sind die von TVLA berechneten Shapegraphenmengen zu den Programmpunkten eines Algorithmus. Die Betrachtung einer Menge ist angemessen, wenn es sich um die Shapegraphenmenge eines Programmpunktes einer Anweisungsfolge handelt. Aber schon in Abschnitt 8.1 haben wir gesehen, dass der zu einem Shapegraphen gehörende symmetrische Partner

nicht unbedingt am gleichen Programmpunkt aufzutreten braucht. Bei einem If-Then-Else-Konstrukt kann der symmetrische Partner eines Shapegraphen einen anderen Ast der Verzweigung „durchlaufen“. Beispielsweise können Shapegraphen, die bei den Programmpunkten im Then-Zweig auftreten, symmetrische Partner haben, die bei den korrespondierenden Programmpunkten des Else-Zweiges vorkommen. Daher ist es notwendig, dass wir eine Notation für eine Messende zur Verfügung haben, in der zwei Mengen berücksichtigt werden. Sie soll quantifizieren, welcher Anteil der Shapegraphen der ersten Menge einen symmetrischen Partner in der zweiten Menge hat. Hierzu erweitern wir Definition 8.16 und legen fest:

8.17 Definition. *Es seien $\mathcal{M}$ und $\mathcal{N}$ zwei nichtleere, endliche und nicht notwendig disjunkte Mengen von Shapegraphen, des Weiteren sei $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Wir setzen*

$$\mu_{\mathcal{D}_\Sigma}(\mathcal{M},\mathcal{N}) = \frac{\left|\{S \in \mathcal{M} \colon \exists T \in \mathcal{N}\ (S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}\}\right|}{|\mathcal{M}|}$$

und nennen $\mu_{\mathcal{D}_\Sigma}$ eine Symmetriemessende *bezüglich $\mathcal{D}_\Sigma$ für $(\mathcal{M},\mathcal{N})$.*

Auch hier schreiben wir anstellen von $\mu_{\mathcal{D}_\Sigma}$ kürzer μ, wenn der Zusammenhang zum Symmetriedefinierer klar ist und keine Unklarheiten oder Verwechslungen zu befürchten sind. Es braucht auch nicht zu irritieren, dass wir in den Definitionen 8.16 und 8.17 dieselbe Bezeichnung für die Messende verwenden. Wegen der unterschiedlichen Anzahl der Argumente brauchen wir keine Verwirrung zu befürchten. Im Übrigen ist $\mu(\mathcal{M})$ in Definition 8.16 identisch mit $\mu(\mathcal{M},\mathcal{M})$ in Definition 8.17. (Somit können wir $\mu(\mathcal{M})$ auch einfach als abkürzende Schreibweise für $\mu(\mathcal{M},\mathcal{M})$ auffassen.) – Der Wert $\mu(\mathcal{M},\mathcal{N})$ gibt den Anteil der Shapegraphen aus $\mathcal{M}$ an, die einen symmetrischen Partner in $\mathcal{N}$ haben. Die im Anschluss an Definition 8.16 gemachten Bemerkungen gelten entsprechend. Symmetrische Shapegraphen werden in $\mu(\mathcal{M},\mathcal{N})$ gezählt, wenn sie in $\mathcal{M} \cap \mathcal{N}$ liegen.

Die Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$ ist symmetrisch. Es könnte sich die Frage aufdrängen, ob diese Eigenschaft sich auf μ überträgt, ob μ symmetrisch in seinen Argumenten ist, ob also $\mu(\mathcal{M},\mathcal{N}) = \mu(\mathcal{N},\mathcal{M})$ gilt. Zunächst stellen wir fest:

8.18 Satz. *Es seien $\mathcal{M}$ und $\mathcal{N}$ zwei nichtleere, endliche und nicht notwendig disjunkte Mengen von Shapegraphen. Dann gilt*

$$|\mathcal{M}| \cdot \mu(\mathcal{M},\mathcal{N}) = |\mathcal{N}| \cdot \mu(\mathcal{N},\mathcal{M}).$$

Beweis. Nach Definition 8.17 gilt

$$\begin{aligned} |\mathcal{M}| \cdot \mu(\mathcal{M},\mathcal{N}) &= \left|\{S \in \mathcal{M} \colon \exists T \in \mathcal{N}\ (S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}\}\right| \\ &= \left|\{T \in \mathcal{N} \colon \exists S \in \mathcal{M}\ (S,T) \in \mathrm{sym}_{\mathcal{D}_\Sigma}\}\right| \\ &= |\mathcal{N}| \cdot \mu(\mathcal{N},\mathcal{M}). \end{aligned}$$

Hierbei nutzen wir die Tatsache, dass die Relation $\mathrm{sym}_{\mathcal{D}_\Sigma}$ symmetrisch ist. □

Betrachten wir zwei nichtleere und endliche Mengen $\mathcal{M}$ und $\mathcal{N}$ von Shapegraphen, die nicht notwendig disjunkt sind. Ist $\mu(\mathcal{M},\mathcal{N}) = \mu(\mathcal{N},\mathcal{M})$, dann gilt nach dem vorangegangenen Satz notwendigerweise $|\mathcal{M}| = |\mathcal{N}|$. Setzen wir umgekehrt $|\mathcal{M}| = |\mathcal{N}|$ voraus, dann folgt mit der gleichen Argumentation $\mu(\mathcal{M},\mathcal{N}) = \mu(\mathcal{N},\mathcal{M})$. Damit haben wir gezeigt:

$$\mu(\mathcal{M},\mathcal{N}) = \mu(\mathcal{N},\mathcal{M}) \;\Leftrightarrow\; |\mathcal{M}| = |\mathcal{N}|\,.$$

Hieraus ersehen wir, dass μ im Allgemeinen nicht symmetrisch in seinen Argumenten ist.

Die Definition von $\mu(\mathcal{M},\mathcal{N})$ in Definition 8.17 als eine Funktion von zwei Argumenten haben wir damit motiviert, dass wir die Ausprägung der Symmetrie zwischen den Ästen einer Verzweigung quantitativ erfassen wollen. Der zu einem Shapegraphen bei einem Programmpunkt in einem Ast der Verzweigung symmetrische Partner kann sich beim korrespondierenden Programmpunkt in einem anderen Ast der Verzweigung befinden. Doch ist die resultierende Messende nicht auf diese Anwendung beschränkt.

In Kapitel 6 haben wir mittels eines Ähnlichkeitsbegriffs eine Partition einer Shapegraphenmenge definiert. Damit können wir die von TVLA berechneten Shapegraphenmengen so partitionieren, dass die Shapegraphen einer Klasse bezüglich des spezifizierten Ähnlichkeitsbegriffs einander ähnlich sind. Betrachten wir als Beispiel die Suche nach einem Element in einen Suchbaum, wobei wir wie üblich die Standardspezifikation zugrunde legen und für die Analyse die üblichen Startshapegraphen – wir sehen sie in Abbildung 8.3 – verwenden. Den Ähnlichkeitsbegriff wählen wir so, dass Shapegraphen mit der gleichen Struktur des Suchweges, also des Weges von der Wurzel zur aktuellen Ecke, als ähnlich angesehen werden; vergleiche die Erläuterungen in Abschnitt 6.3. Symmetrie interessiert uns hinsichtlich des Prädikatenpaares $(\mathsf{left}, \mathsf{right})$. Wir betrachten die Situation am Programmpunkt n2, das ist der Schleifeneingangspunkt der Suchschleife. Wenn wir die symmetrischen Shapegraphen außer Acht lassen, dann hat jeder Shapegraph einen symmetrischen Partner (an diesem Programmpunkt). Allerdings befinden sich zwei zueinander symmetrische Shapegraphen in verschiedenen Klassen der Partition. Wir können die Symmetriemessende μ auch dazu verwenden, um den Grad der Symmetrie zwischen Klassen einer Partition einer Shapegraphenmenge zu begutachten und zu beurteilen. Wir können mittels μ also auch feststellen, ob und inwieweit das aktuell gewählte Klassen- und Symmetriekonzept miteinander harmonisieren.

Symmetriemessende für Transitionen

In Abschnitt 8.1 haben wir den Begriff der zueinander symmetrischen Shapegraphen entwickelt, wobei wir uns dort auch für Symmetrie zwischen Shapegraphenmengen interessiert haben. In Abschnitt 8.2 haben wir das Konzept auf Kantenzüge im Transitionsgraphen übertragen. Die Untersuchung erfolgte in beiden Abschnitten auf einer qualitativen Ebene. In diesem Abschnitt erschließen wir (uns) quantitative

Aspekte des Untersuchungsgegenstandes. Bisher haben wir die Gedankengänge aus Abschnitt 8.1 quantitativ behandelt. Ein wesentlicher Kern des Abschnitts 8.2 ist der Begriff der Symmetrieabgeschlossenheit in Definition 8.15. Ihr wollen wir uns zum Abschluss dieses Abschnitts widmen. Wir können uns mit einer Skizzierung begnügen, da es nach den bisherigen Ausführungen klar ist, wie eine quantitative Form hierzu aussehen sollte.

Wie auch im ganzen Abschnitt 8.2 benötigen wir als Analyseausgabe einen Transitionsgraphen. Anstelle von abstrakten Zuständen werden wir im Folgenden wieder vereinfachend von Shapegraphen sprechen. Wir betrachten vier Shapegraphenmengen $\mathcal{M}$, $\mathcal{M}'$, $\mathcal{N}$ und $\mathcal{N}'$, wobei die Shapegraphen jeder Menge zum selben Programmpunkt gehören. Da es hier letztendlich um „Kanten“ geht, setzen wir partielle Symmetrieabgeschlossenheit zwischen den Mengen voraus. Sie kann mit den schon eingeführten Messenden geprüft werden. Wir fordern $\mu(\mathcal{M},\mathcal{M}') = \mu(\mathcal{N},\mathcal{N}') = 1$. Für einen Shapegraphen $S \in \mathcal{M}$ bezeichne S' den symmetrischen Partner von S in $\mathcal{M}'$, entsprechen sei $T' \in \mathcal{N}'$ der symmetrische Partner eines Shapegraphen $T \in \mathcal{N}$. Für jedes S und T existieren die symmetrischen Partner, aber aus der Existenz einer Kante $[S,T\rangle$ folgt nichts über die Existenz der Kante $[S',T'\rangle$, die wir die zu $[S,T\rangle$ *symmetrische Kante* nennen wollen. Wir definieren $\mu\left([(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle\right)$ als den Anteil der Kanten zwischen Shapegraphen (abstrakten Zuständen) aus $\mathcal{M}$ zu Shapegraphen aus $\mathcal{N}$, die eine dazu symmetrische Kante zwischen $\mathcal{M}'$ und $\mathcal{N}'$ besitzen.

8.19 Definition. *Es sei $\mathcal{TG}$ ein Transitionsgraph und $\mathcal{D}_\Sigma$ ein Symmetriedefinierer. Des Weiteren seien $\mathcal{M}$, $\mathcal{M}'$, $\mathcal{N}$ und $\mathcal{N}'$ Shapegraphenmengen (Mengen von abstrakten Zuständen), wobei die Shapegraphen jeder Menge zum selben Programmpunkt gehören und es gelte $\mu(\mathcal{M},\mathcal{M}') = \mu(\mathcal{N},\mathcal{N}') = 1$. Wir setzen*

$$\mu_{\mathcal{D}_\Sigma}\left([(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle\right) = \frac{|\{[S,T\rangle : S \in \mathcal{M}, T \in \mathcal{N}, [S',T'\rangle \in \mathcal{TG}\}|}{|\{[S,T\rangle : S \in \mathcal{M}, T \in \mathcal{N}\}|},$$

wobei S' einen Shapegraphen aus $\mathcal{M}'$ mit $(S,S') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$ bezeichnet und entsprechen T' einen Shapegraphen aus $\mathcal{N}'$ mit $(T,T') \in \mathrm{sym}_{\mathcal{D}_\Sigma}$.

Wir haben wieder μ als Symbol für die Messende benutzt. Verwechslungen sind aber nicht zu befürchten, da die Struktur des Argumentes eine eindeutige Zuordnung (des Zeichens zur Definition) ermöglicht. Auf die Indizierung mit dem Symmetriedefinierer verzichten wir in der Regel wieder. – Für Shapegraphenmengen $\mathcal{M}$, $\mathcal{M}'$, $\mathcal{N}$ und $\mathcal{N}'$ ist $[(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle$ genau dann symmetrieabgeschlossen, wenn $\mu\left([(\mathcal{M},\mathcal{M}'),(\mathcal{N},\mathcal{N}')\rangle\right) = \mu\left([(\mathcal{M}',\mathcal{M}),(\mathcal{N}',\mathcal{N})\rangle\right) = 1$ gilt. – Mit dieser Messenden können wir nicht nur den Grad der Symmetrieabgeschlossenheit beurteilen. Wir können damit beispielsweise auch messen, inwieweit TVLA-Aktionen zueinander symmetrisch sind, vergleiche die Ausführungen hierzu in Abschnitt 8.2.

Beleuchten wir eine Anwendungsmöglichkeit. Es sei K ein Kantenzug (im Transitionsgraphen) entlang der Shapegraphen (abstrakten Zustände) $S_1, S_2, \ldots, S_k$ durch die Programmpunkte $m_1, m_2, \ldots, m_k$, $k \geq 1$, weiter sei K' ein zu K symmetrischer Kantenzug entlang $S'_1, S'_2, \ldots, S'_k$ durch die Programmpunkte $n_1, n_2, \ldots, n_k$. Die Programmpunkte brauchen nicht verschieden zu sein. Zu jedem Programmpunkt m_i, $1 \leq i \leq k$, sei eine Teilmenge $\mathcal{M}_i$ der Shapegraphenmenge dieses Programmpunktes ausgewählt, und es gelte sinnigerweise $S_i \in \mathcal{M}_i$. Ebenso sei zu jedem Programmpunkt n_i, $1 \leq i \leq k$, eine Teilmenge $\mathcal{M}'_i$ der Shapegraphenmenge von n_i ausgewählt, und es gelte $S'_i \in \mathcal{M}'_i$. Wir wollen die beiden Stränge quantitativ untersuchen. Zunächst können wir für jedes i, $1 \leq i \leq k$, die Werte $\mu(\mathcal{M}_i, \mathcal{M}'_i)$ und $\mu(\mathcal{M}'_i, \mathcal{M}_i)$ berechnen. Sind sie von 1 verschieden, dann sind die Shapegraphenmengen für die Betrachtung vermutlich unglücklich gewählt. Es könnte auch sein, dass der Algorithmus beziehungsweise der berechnete Transitionsgraph keine hinreichende Symmetrie bezüglich des verwendeten Symmetriedefinierers aufweist. Hat man die Shapegraphenmengen so bestimmt, dass $\mu(\mathcal{M}_i, \mathcal{M}'_i) = \mu(\mathcal{M}'_i, \mathcal{M}_i) = 1$ für jedes i, $1 \leq i \leq k$ gilt, dann kann man $\mu\left([(\mathcal{M}_i, \mathcal{M}'_i), (\mathcal{M}_{i+1}, \mathcal{M}'_{i+1})\rangle\right)$ und $\mu\left([(\mathcal{M}'_i, \mathcal{M}_i), (\mathcal{M}'_{i+1}, \mathcal{M}_{i+1})\rangle\right)$ für $1 \leq i \leq k-1$ betrachten. Im Idealfall sind alle Werte gleich 1. Wenn nicht, dann sind die Shapegraphenmengen vermutlich immer noch nicht gut gewählt oder wir haben den Anhaltspunkt einer potentiellen Unstimmigkeit. Sind einzelne Werte vorhanden, die stark von den übrigen abweichen, dann können die zugehörigen Programmpunkte beispielsweise als Ausgangspunkt einer weitergehende Examination des Algorithmus beziehungsweise des Transitionsgraphen dienen.

Zusammenfassung

Unser Ansatz zur Algorithmenvisualisierung basiert auf der Visualisierung der abstrakten Programmausführung. Wir berechnen im Vorfeld mittels einer Shapeanalyse für den betrachteten Algorithmus eine Beschreibung der möglichen Heapsituationen. Die Analyse wird durch die Analysespezifikation gesteuert. Theoretische Überlegungen legen eine gewisse Reichhaltigkeit der Spezifikation nahe; zudem wünschen wir Prädikate, die im Hinblick auf die Visualisierung eine starke und deutliche Aussage haben. Bei Datenstrukturen, die komplizierter als Listen sind, resultiert aus dieser Forderung in der Regel eine (sehr) großen Analyseausgabe. Eine direkte Visualisierung ist aufgrund ihrer Größe häufig als problematisch anzusehen. Daher befassen wir uns in dieser Arbeit mit Mechanismen zur Strukturierung der Analyseausgabe. Das Symmetriekonzept, das wir in diesem Kapitel behandelt haben, ordnet sich in diesen Rahmen ein. Symmetrie ist eine visuelle Form von Ähnlichkeit. Hat man aus einer Menge zueinander symmetrischer Situationen eine verstanden, dann erschließen sich die anderen in der Regel leicht und brauchen nicht notwendigerweise mitvisualisiert zu werden, sie können beispielsweise herausgefiltert oder ausgeblendet werden.

In Abschnitt 8.1 haben wir einen Symmetriebegriff für Shapegraphen entwickelt.

Das symmetrische Aussehen der gezeichneten Shapegraphen diente uns als Inspiration. Formal stellt sich der Begriff als binäre Relation dar. Wir haben ihre Eigenschaften studiert und die durch sie induzierte Strukturierung einer Shapegraphenmenge eingeführt. Wie schon das Ähnlichkeitskonzept in Kapitel 6 ist auch das Symmetriekonzept parametrisch und kann damit wechselnden Wünschen und Anforderungen des Visualisierungsanwenders angepasst werden. Mit der Symmetrieabgeschlossenheit haben wir Mengen von Shapegraphen (beziehungsweise Paare von Shapegraphenmengen) mit einem besonders hohen Grad an Symmetrie ausgezeichnet.

In Abschnitt 8.2 haben wir den Symmetriebegriff in den Transitionsgraphen hineingetragen, ihn gewissermaßen auf Kanten übertragen. Die Visualisierung (ebenso das Verstehenwollen) eines Algorithmus beinhaltet das Studium des Programmflusses, also die Untersuchung von Sequenzen von an aufeinanderfolgenden Programmpunkten auftretenden Situationen. Diesen Sequenzen entsprechen (Mengen von) Kantenzügen im Transitionsgraphen. Wir haben erklärt, wann Kantenzüge zueinander symmetrisch sind. Auch hier haben wir mit der Symmetrieabgeschlossenheit einen besonders hohen Grad an Symmetrie ausgezeichnet.

Ist ein Symmetriedefinierer spezifiziert, dann braucht in der Analyseausgabe keine Symmetrie bezüglich dieser Symmetrieinstanziierung vorzuliegen oder sie kann schwach ausgeprägt sein. Es ist daher nützlich, über Mittel zu verfügen, mit denen man beurteilen kann, ob und wie stark Symmetrie ausgeprägt ist. Diesem Aspekt, der die bisherige qualitative Diskussion um eine quantitative Komponente ergänzt, haben wir uns im aktuellen Abschnitt gewidmet. Wir haben eine Symmetriemessende eingeführt, die den Grad der Ausprägung von Symmetrie innerhalb einer Shapegraphenmenge beziehungsweise zwischen zwei Menge misst. In einem nächsten Schritt haben wir eine weitere Messende für „Kantenmengen“ (des Transitionsgraphen) eingeführt, so dass auch eine quantitative Beschreibung der Symmetrie in Bezug auf Kantenzüge möglich ist. Obwohl die Messenden von ihrer Natur her einfach gehalten sind, erlauben sie doch – oder gerade deshalb – eine praktikable Anwendung. Zudem lassen sie sich dazu verwenden, um das Zusammenspiel vom Ähnlichkeits- und Symmetriekonzept zu beurteilen. Auch lassen sich mit ihnen Rückschlüsse auf den Symmetriegehalt der TVLA-Aktionen in der Analysespezifikation anstellen.

9 Anwenderrollen

> Denn das allein ist deine Aufgabe:
> die dir zugeteilte Rolle gut zu spielen; sie auszuwählen, ist Sache eines anderen.
>
> *(Epiktet)*

Gegenstand dieser Arbeit ist ein Ansatz zur Algorithmenvisualisierung, der auf Shapegraphen basiert. Wir behandeln dabei hauptsächlich konzeptionelle Gesichtspunkte. In diesem Kapitel widmen wir uns den Arten von Interaktionen, die im Zusammenspiel mit einem Anwender auftreten. Den primären Interaktionspartner auf der Seite der Algorithmenvisualisierung nennen wir *Visualisierer*. Wir können ihn ganz allgemein als gedankliches Konstrukt der Konzepte und Methoden verstehen. Wir können ihn auch konkreter als ein Softwarepaket verstehen, dass diese Konzepte und Methoden realisiert. Noch konkreter können wir an die Visualisierungssoftware denken, die in [Parduhn 2005] beschrieben ist und die zum Zeitpunkt der Abfassung dieser Arbeit weiterhin Gegenstand stetiger Weiterentwicklung ist.

Wir sehen den hier beschriebenen Ansatz zur Algorithmenvisualisierung als ein lernunterstützendes Mittel, als ein Werkzeug zum Verständnisgewinn. Eine Interaktion mit dem Visualisierer stellt eine Lernsituation dar. Je nach Gegenstand, was gelernt werden soll, je nach Stand der Vorkenntnisse des Anwenders, je nach spezifischer Absicht der Visualisierungssitzung ergeben sich unterschiedliche Lernsituationen. Wenn verschiedene Personen mit dem Visualisierer interagieren, dann ist die Art oder Zielsetzung ihrer Interaktion nicht immer die gleiche. Denken wir beispielsweise zum einen an eine Person, die den Visualisierer zum Zwecke des Algorithmenlernens verwendet, und zum anderen an einen TVLA-Entwickler, der ihn zum Überprüfen und Testen der von der Analyse berechneten Ausgabe benutzt. In der Regel werden beide ihr Augenmerk auf ganz verschiedene Aspekte der Analyseausgabe lenken. Dieser Sachverhalt ist nicht auf verschiedene Personen begrenzt. Auch dieselbe Person kann den Visualisierer (zu verschiedenen Zeitpunkten) mit verschiedenen Absichten benutzen, ein Beispiel folgt weiter unten.

Die „Art und Weise“, die „Form“ einer Interaktion bezeichnen wir mit dem Wort *Rolle*. Sie isoliert einen Typus von Interaktion. Da Personen mehrere Rollen einnehmen können, es also keine Bijektion zwischen Personen(gruppen) und Rollen gibt, sind Interaktionen mit dem Visualisierer nur unzureichend über Personengruppenzugehörigkeiten bestimmbar. Meistens lässt sich die Rolle treffend durch

den Zweck, beziehungsweise die Zielsetzung der Interaktion kennzeichnen. In diesem Kapitel werden wir die jeweiligen Charakteristiken der verschiedenen Rollen isoliert untersuchen. Dieses Verständnis erlaubt, bei der (theoretischen) Entwicklung und anschließenden Implementierung neuer Funktionalität, diese besser auf einen Anwendungszweck, auf eine Lernsituation abzustimmen. Haben verschiedene Rollen ähnliche Funktionalitätsanforderungen, kann dies bei der Realisierung einfacher bedacht werden, so dass sich deren Realisierung auf alle Rollen positiv auswirkt.

Wir werden neun Rollen unterscheiden, die wir wie folgt bezeichnen: Entwickler des Visualisierers, Shapeanalyst, TVLA-Entwickler, Spezifikationsentwerfer, Visualisierungsinszenator, Algorithmen-Datenstrukturen-Experte, Algorithmen-Datenstrukturen-Lernender, Programmtester und Programmoptimierer. Eine Erklärung der Rollen erfolgt weiter unten. Mit ihnen decken wir die üblichen Interaktionen mit dem Visualisierer ab. Wir verstehen diese Liste aber als eine offene Liste: Sollten sich in der Zukunft weitere beachtenswerte Rollen herauskristallisieren, kann sie problemlos um diese ergänzt werden. Diese Arbeit behandelt Algorithmenvisualisierung, daher ist natürlich auch der Bereich Algorithmen und Datenstrukturen in der Rolleneinteilung vertreten. Die an Lernprozessen grundsätzlich beteiligten Personen sind der Lernende und der Wissende, der als Lehrender auftritt. Diese Unterscheidung ist im Bereich Algorithmen und Datenstrukturen in der Rolleneinteilung umgesetzt, während sie aber beispielsweise im Bereich Programmoptimierung nicht durchgeführt ist.

Rollen treten nur ganz selten in Reinform auf, eine reale Person wird nur selten in genau einer Rolle mit dem Visualisierer interagieren. Betrachten wir als Beispiel einen Algorithmen-Datenstrukturen-Experten, der an Invarianten für einen Algorithmus interessiert ist. Er schreibt zunächst eine Analysespezifikation, die seinen Wünschen entspricht; er tritt als Spezifikationsentwerfer auf. Anschließend führt er eine Shapeanalyse aus und initialisiert den Visualisierer mit der gewünschten Darstellung, er ist Visualisierungsinszenator. Vielleicht hat er beim Programmieren des Kontrollflussgraphen Fehler gemacht und muss sich nun noch als Programmtester betätigen.

Wir charakterisieren jede Rolle in Bezug auf die folgenden vier Aspekte:

- Als Erstes erfolgt eine Beschreibung der Rolle. Daran schließt sich eine Darstellung der (für die Rolle) charakteristischen Interaktionen mit dem Visualisierer an. Die Rolle definiert sich hauptsächlich aus dem Zweck, aus der Zielsetzung der Interaktion. In den allermeisten Fällen lässt sich die Interaktion als ein Lernvorgang auffassen, als ein Hinzugewinnen von Verständnis. Dieser ist auf einen Gegenstand (das zu Lernende) gerichtet, den wir abschließend benennen.
- Wenn eine Person mit einer bestimmten Absicht mit dem Visualisierer interagieren möchte, dann sollte beziehungsweise muss sie über die nötigen Vorkenntnisse dazu verfügen; sonst kann sie ihrer Rolle nicht gerecht werden.

Daher befassen wir uns als Zweites mit dem Grad an Kenntnissen, über den eine Person verfügen sollte (muss), damit sie diese Rolle annehmen kann.

- Aus der Rollendefinition, aus dem Zweck der Interaktion, ergeben sich spezifische Wünsche und Anforderungen an den Visualisierer. Als Drittes beschreiben wir, welche Funktionalität eine Rolle vom Visualisierer erwartet und wünscht.
- Wir haben oben in einem Beispiel gesehen, dass eine reale Person oft mehrere Rollen einnehmen kann und dies häufig auch tut. Das zeigt sehr deutlich, dass Rollen in einer Wechselbeziehung zueinander stehen. Diesem Aspekt widmen wir uns als vierten Punkt.

Für jede Rolle charakterisieren wir den gewünschten beziehungsweise erforderlichen Stand an Vorkenntnissen, den eine Person aufweisen sollte beziehungsweise muss, damit sie der Rolle gerecht werden kann. Diese Vorkenntnisse können ganz verschiedenen Themenbereiche angehören, weswegen wir diesen Punkt differenzieren. Im Einzelnen unterscheiden wir:

Shapeanalyse: Hierunter verstehen wir die (logische) Theorie, die der Programmanalyse zugrunde liegt. Die Skala reicht von guten Kenntnissen der Methode über ein für Anwendungen ausreichendes Verständnis, hierzu gehört das Verstehen wie TVLA-Aktionen auf Shapegraphen operieren, bis hinunter zur Fähigkeit, Shapegraphen lesen und verstehen, mit ihnen etwas anfangen zu können.

TVLA: Hierzu zählen wir alle Kenntnisse über die Analysesoftware TVLA. Es können zwei Dimensionen, zwei Arten, von Vorkenntnissen unterschieden werden. Zum Ersten sind dies Kenntnisse, wie sie zur Entwicklung beziehungsweise Weiterentwicklung der Software notwendig sind. Da wir Programmierkenntnisse als Bereich extra aufführen, wollen wir darunter in erster Linie Kenntnisse über Interna der Software verstehen. Zum Zweiten handelt es sich um Kenntnisse über die Benutzung der Software. Hierzu zählt das Wissen um die angebotenen Features und das Verständnis, die daraus erwachsenen Möglichkeiten zu nutzen. Hierzu zählt auch die Feinabstimmung der Analyse (mittels properties).

Programmierung: Hierunter wollen wir die Gesamtheit der Kenntnisse verstehen, die zur Erstellung von Software notwendig sind. Sie umfassen sowohl eher theoretische Aspekte wie Softwaretechnik (Software engineering) als auch praktische wie die Beherrschung der verwendeten Programmiersprache und Bibliotheken.

Didaktik: Hierunter verstehen wir ganz allgemein Kenntnisse über das Lehren. Wenn man Algorithmenvisualisierung als lernunterstützendes Mittel verwendet, tritt sie (die Software beziehungsweise derjenige, der eine konkrete Visualisierung vorbereitet) als Lehrender auf. Verständnis über die Zusammenhänge, wie Menschen lernen, ist eine Grundvoraussetzung für effizientes Lernen.

Visualisierung: Algorithmenvisualisierung gehört, wie der Name schon sagt, zur Visualisierung; ihre Objekte sind Algorithmen. Unter Visualisierung verstehen wir allgemein eine bildliche Formulierung und Kommunikation. Der Begriff bezeichnet die Überführung und Präsentation von Daten oder Zusammenhängen in eine graphische beziehungsweise visuell erfassbare Form. Konkreter ausformuliert verstehen wir darunter das Wissen um die Möglichkeiten und Methoden der visuellen Aufbereitung, insbesondere der computergestützten, sowie ihre wirkungsvolle Nutzung.

Visualisierer: Hierunter verstehen wir die Software, mit der der Anwender interagiert. Es können wie bei TVLA zwei Dimensionen, zwei Arten, von Vorkenntnissen unterschieden werden. Zum Ersten sind dies Kenntnisse, die zur (Weiter-)Entwicklung der Software notwendig sind. Das Spektrum ist wieder weit gefächert und umfasst sowohl Kenntnissen in Didaktik und Visualisierung als auch Kenntnisse in Programmierung. Da wir die meisten Bereiche gesondert aufführen, wollen wir hierunter in erster Linie Interna der Software verstehen. Zum Zweiten handelt es sich um Kenntnisse über die Benutzung der Software. Hierzu zählt die Kenntnis der angebotenen Funktionalität und das Verständnis, die daraus erwachsenen Möglichkeiten zu nutzen.

Algorithmen und Datenstrukturen: Die Gegenstände der Algorithmenvisualisierung sind Algorithmen, und diese verwenden bestimmte Datenstrukturen. Wenn Algorithmenvisualisierung als lernunterstützendes Mittel benutzt wird, sind normalerweise Algorithmen und Datenstrukturen der Gegenstand des Lernens. Die Skala reicht vom Experten über jemand, der den Inhalt einer gleichlautenden Vorlesung theoretisch verstanden hat und ihn beim Programmieren umsetzen kann, bis zu jemand, der diese Inhalte gerade lernt.

Wir werden nicht für jede Rolle alle genannten Vorkenntnisbereiche ansprechen. Einige sind für bestimmte Rollen typisch, während sie für die restlichen unerheblich sind. Zum Beispiel sind Didaktikfähigkeiten primär nur für den Entwickler des Visualisierers, für den Spezifikationsentwerfer und für den Visualisierungsinszenator wichtig, umfassende Softwareentwicklungsfähigkeiten sind nur für die Rollen bedeutsam, die das Wort Entwickler in sich tragen. Wir werden jeweils nur die im Hinblick auf den Zweck der Interaktion mit dem Visualisierer relevanten Themenbereiche ansprechen.

Rollen stehen häufig in einer Wechselbeziehung, in einer Interaktion, miteinander. Eine Person kann beim Umgang mit dem Visualisierer zum Beispiel mehrere Rollen bekleiden. Einige dieser Interaktionen tauchen bei (fast) allen Rollen auf, weswegen wir sie hier vorziehen und im Folgenden gesondert beschreiben.

- Der Entwickler des Visualisierers interagiert mit jeder anderen Rolle, beziehungsweise diese mit ihm, in der Art, dass der Entwickler Rückmeldungen von den Anwendern anfordert oder erhält. Sie liefern Vorschläge für Erweiterungen, machen auf Fehler aufmerksam und dergleichen.

- Ein Anwender des Visualisierers benötigt eine Inszenierung. Sobald eine Rolle mit dem Visualisierer interagiert, ist sie auch ein Anwender. Damit interagiert jede Rolle mit (der Rolle des) Visualisierungsinszenators.

  Da jede Benutzung des Visualisierers einer bestimmten Absicht unterliegt, muss auch die Spezifikation auf diesen Zweck abgestimmt sein. Sofern keine darauf zugeschnittene Spezifikation vorliegt, ist auch der Spezifikationsentwerfer an der Wechselwirkung beteiligt. (Man könnte dies auch lediglich als eine Interaktion zwischen dem Inszenator und dem Spezifikationsentwerfer ansehen.)

- Wir können zwei Arten von Wechselbeziehungen unterscheiden: Zum Ersten gibt es solche, die den Visualisierer mit einbeziehen, die auf ihn bezogen sind. Wir können sie primäre Interaktionen nennen. Die eben beschriebenen Wechselwirkungen sind von dieser Art. Zum Zweiten existieren Interaktionen, die den Visualisierer primär nicht mit einbeziehen. Wir können sie als sekundäre Interaktionen bezeichnen. Hierzu gehört, wenn ein Lernender oder eine andere Rolle sich an einen Algorithmen-Datenstrukturen-Experten wendet, um Verständnisproblem hinsichtlich des Algorithmus zu besprechen, oder wenn ein TVLA-Entwickler sich mit einem Shapeanalysten bespricht. Bei Interaktion dieser Art werden wir nur die sehr charakteristischen erwähnen.

In diesem Abschnitt müssen wir klarer zwischen Algorithmus und Programm unterscheiden. Beide Wörter beschreiben eine Formulierung einer Vorgehensweise, in unserem Fall eine Berechnung im Stile imperativer Programmiersprachen. Allerdings ist der Grad der Detailliertheit unterschiedlich. Algorithmen sind allgemeiner und betonen mehr das Prinzip der Berechnung und die strukturellen Zusammenhänge. Programme sind spezieller, bei ihnen ist jedes Detail umgesetzt. Algorithmen werden durch Programme realisiert, umgekehrt kann man Programme als Instantiierungen von Algorithmen auffassen. Die Shapeanalyse wird naturgemäß, der TVLA-Kontrollflussgraph ist ein Programm, für ein Programm berechnet. Die Ergebnisse lassen sich häufig auf den Algorithmus übertragen. (In Abschnitt 4.3 ist die Umsetzung von Algorithmen(beschreibungen) in TVLA-Kontrollflussgraphen angesprochen.) Bei einigen der vorgestellten Rollen liegt das Augenmerk auf dem Algorithmus, auf dem Prinzip der Berechnung, während es bei anderen auf dem Programm, auf der konkreten Realisation, liegt.

Entwickler des Visualisierers

Beschreibung der Rolle: Es geht um die Arbeit an der Software des Visualisierers. Dies umfasst seine Realisierung als Programm und die Erweiterung seiner Funktionalität. Hierzu gehören auch die Implementierung von Änderungen und die Wartung der Software. Jemand, der diese Rolle einnimmt, interagiert mit dem Visualisierer als Anwendung auf die Weise, dass er die Korrektheit der programmierten Modifikationen verifizieren möchte. Der Gegenstand des Verstehen-Wollen ist die Software des Visualisierers.

Vorkenntnisse: Der Name der Rolle suggeriert einen hohen Grad an Kenntnissen im Bereich Programmierung und Visualisierung. Natürlich kennt er sich sehr gut mit den Interna des Visualisierers aus. Da es um Algorithmenvisualisierung mittels Shapegraphen geht, darf ein wenigstens durchschnittliches Verständnis der Shapeanalyse erwartet werden. Da wir die Benutzung des Visualisierers als Lernvorgang interpretieren, sind auch didaktische Kenntnisse vorteilhaft, um das Lernprozess effektiv zu unterstützen. Kenntnisse im Bereich Algorithmen und Datenstrukturen sind dann nützlich, wenn er als Tester seines Programms auftritt.

Anforderungen an den Visualisierer: Wenn ein Entwickler des Visualisierers mit dem Programm interagiert, dann primär zum Austesten von Änderungen. Seine Anforderungen richten sich dann nicht auf die Qualität der Visualisierung der Analyseausgabe und der einzelnen Shapegraphen. Im Vergleich zu den anderen Rollen nimmt er damit eine Sonderrolle ein. Deswegen werden wir seine Anforderungen auch nicht weiter betrachten.

Interaktion mit anderen Rollen: Die Hauptinteraktionen besteht wohl üblicherweise auf dem Einholen von Feedback, die anderen zuvor angesprochenen Wechselwirkungen bestehen natürlich auch. Zuweilen interagiert er mit dem TVLA-Entwickler: Für die Visualisierung des Programmablaufes ist es förderlich, wenn die Analyse nicht (nur) für jeden Programmpunkt eine Menge von Shapegraphen, sondern stattdessen einen Transitionsgraphen berechnet. In einem der Visualisierungsprojekte, vergleiche [Bieber 2001], wurde beispielsweise die Analysemaschine dahingehend erweitert, dass in jedem Shapegraphen bei jeder Ecke vermerkt wurde, auf welche Art sie entstanden war, ob sie also beispielsweise schon im Ausgangsshapegraphen vorhanden war, ob sie durch Materialisation aus einer Summary-Ecke hervorgegangen war oder ob sie durch Verschmelzung bei der Blur-Operation entstanden war.

Shapeanalyst

Beschreibung der Rolle: Es geht um die Shapeanalyse als solche. Die Interessen liegen im Bereich Logik und Programmanalyse. (Wir haben es also nicht mit einem professionellen Börsenspezialisten zu tun). Die Interaktion mit dem Visualisierer ist eher beschränkt, er dient in erster Linie zum Testen und experimentellen Verifizieren von Aspekten der Shapeanalyse. Der Lernvorgang, der Abstraktionsprozess im Gehirn richtet sich auf die Shapeanalyse.

Vorkenntnisse: Ein Shapeanalyst ist in erster Linie ein Experte in Shapeanalyse. Damit einher geht sicherlich eine überdurchschnittliche Kenntnis der Interna von TVLA. Seine Kenntnisse im Bereich Algorithmen und Datenstrukturen sind zumindest so hoch, dass die Beispiele, die er (zum Testen) verwendet, für ihn klar sind.

Anforderungen an den Visualisierer: Da die Interaktion dieser Rolle mit dem Visualisierer eher beschränkt ist, können auch nur bedingt charakteristische Anforde-

rungen formuliert werden. Wenn der Shapeanalyst einen Aspekt der Shapeanalyse geändert hat, der sich nur auf bestimmte Shapegraphen auswirkt, wird er diese auf einfache Art finden wollen. Die Visualisierung sollte das Filtern der Analyseausgabe nach spezifizierbaren Eigenschaften unterstützen.

Interaktion mit anderen Rollen: Es bestehen nur die zuvor genannten (primären und sekundären) Wechselwirkungen. Er kann gegenüber anderen Rollen auch als Lehrender im Bereich Shapeanalyse auftreten.

TVLA-Entwickler

Beschreibung der Rolle: Es geht um die Umsetzung der (Theorie der) Shapeanalyse in Software, also um das Arbeiten an der TVLA-Software. Hierzu gehört die Realisierung der Shapeanalyse als Programm, die Implementierung von Erweiterungen und Änderungen sowie die Pflege der TVLA-Software. Die Interaktion dieser Rolle mit dem Visualisierer ist beschränkt. Sie verwendet ihn in erster Linie als Debugging-Hilfe und zum Testen. Es geht dabei darum, sich von der Korrektheit der Änderungen an der Software zu überzeugen. Der Lerngegenstand ist damit die TVLA-Software.

Vorkenntnisse: Er verfügt über ein überdurchschnittliches Wissen im Bereich Shapeanalyse. Das theoretische Niveau eines Shapeanalysten ist nicht erforderlich, jedoch sind praktische Aspekte nützlich und wichtig. Er verfügt sowohl über gutes theoretisches Wissen als auch über praktische Fähigkeiten im Bereich Softwareentwicklung, in den Interna von TVLA kennt er sich gut aus. Was seine Kenntnisse im Bereich Algorithmen und Datenstrukturen anbetrifft, gilt das Gleiche wie beim Shapeanalysten: Sie sind zumindest so hoch, dass die Beispiele, die er (zum Testen und Debuggen) verwendet, für ihn keine weiteren Problem aufwerfen.

Anforderungen an den Visualisierer: Die Interaktion, die im Umfeld der Entwicklung, Erweiterung und Wartung von TVLA erfolgt, kann nach ihrer Auswirkung auf die Analyse in zwei Arten eingeteilt werden: Die Softwareänderungen können sich auf (fast) alle Shapegraphen während der Berechnung auswirken, dies ist zum Beispiel der Fall, wenn eine andere Fokusoperation implementiert wird, oder die Änderung kann sich nur auf bestimmte Shapegraphen auswirken. Im zweiten Falle will man diese Art von Shapegraphen bei der Visualisierung besonders überprüfen. Wie auch beim Shapeanalysten sollen Such- und Filterfunktionen unterstützt werden. Im ersten Fall will man wie der Algorithmen-Datenstrukturen-Lernende gut durch die Analyseausgabe navigieren können. Generell ist für ihn die Mitvisualisierung von Debug-Informationen ausgesprochen nützlich.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Außerdem kann eine Interaktion mit dem Entwickler des Visualisierers stattfinden (siehe die Beschreibung dort).

Spezifikationsentwerfer

Beschreibung der Rolle: Diese Rolle umfasst die Spezifikation von Shapeanalysen. Zu einer gegebenen Datenstruktur wird eine Abstraktion festgelegt, elementare Operationen auf der Datenstruktur werden als TVLA-Aktionen formuliert. Ein Algorithmus, der auf dieser Datenstruktur operiert, wird als TVLA-Kontrollflussgraph kodiert und es werden Startshapegraphen spezifiziert, die den Heap zu Beginn des Programmstarts beschreiben. Jemand, der diese Rolle einnimmt, verwendet den Visualisierer als Test- und Debugging-Hilfe. Er überzeugt sich, dass die Abstraktion im Hinblick auf den beabsichtigten Zweck ausreichend und angemessen ist, dass die Shapegraphen in ihrer Struktur so aussehen wie erwartet und die erwünschten Aussagen erlauben. Fehler in der Spezifikation wirken sich in „fehlerhaften“ Shapegraphen aus. Er sucht und untersucht sie, um dadurch Probleme und Unstimmigkeiten auf der Seite der Spezifikation zu verstehen. Der Gegenstand des Lernens, des Verstehen-Wollen, ist die Spezifikation.

Vorkenntnisse: Im Bereich Shapeanalyse muss verstanden werden, wie eine TVLA-Aktion auf einen Shapegraphen wirkt, also unter anderem was die Fokus- und Coerce-Operationen bewirken. TVLA sollte gut bedient werden können, Kenntnisse der Interna sind aber nicht erforderlich. Programmierkenntnisse sind nur insoweit erforderlich, als die Kodierung des Algorithmus als Kontrollflussgraph erfolgen und die Spezifikation syntaktisch einwandfrei korrekt sein muss. Die Datenstruktur und der Algorithmus müssen verstanden sein, es müssen starke, charakterisierende Eigenschaften, die dann gegebenenfalls als Prädikate formuliert werden, benannt werden können. Dies bestimmt die erforderlichen Kenntnisse im Bereich Algorithmen und Datenstrukturen. Didaktisches Wissen kann nicht schaden, besonders wenn die Spezifikation von Lernenden im Bereich Algorithmen und Datenstrukturen verwendet werden soll, ist es angeraten. Für das Kontrollieren und Testen der Spezifikation mit dem Visualisierer sollte seine Bedienung wenigstens auf durchschnittlichem Niveau beherrscht werden.

Anforderungen an den Visualisierer: Für einen allgemeinen Überblick über die Analyseausgabe sind angemessene Navigationsmöglichkeiten durch die Analyseausgabe notwendig. Probleme und Fehler in der Spezifikation sind häufig lokal, das heißt sie wirken sich nur auf Shapegraphen mit bestimmten Eigenschaften aus. Daher erwartet er ausreichende Such- und Filterfunktionen. In der Debug-Ausgabe von TVLA werden bei jeder Anwendung einer Aktion die Zwischenshapegraphen, also die Shapegraphen nach Anwendung der Fokus-, (Coerce-), Precondition-Update-, Coerce- und Blur-Operation, mit ausgegeben. Besonders die Ausgabe nach der Coerce-Operation ist bei der Formulierung von Konsistenzregeln zur Verschärfung von Shapegraphen essenziell, nur mit dieser lassen sich Constraint-Breaches angemessen beurteilen. Daher profitiert diese Rolle sehr stark, wenn TVLA-Debug-Ausgaben mitvisualisiert werden können.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Gegebenenfalls bespricht er sich beim Entwickeln

einer Spezifikation mit einem Algorithmen-Datenstrukturen-Experten (sekundäre Interaktion).

Visualisierungsinszenator

Beschreibung der Rolle: Diese Rolle beschreibt die Vorbereitung des Visualisierers für einen Benutzer. Er wählt im Hinblick auf die Vorkenntnisse und den Zweck, mit dem der Anwender den Visualisierer benutzen will, eine Spezifikation aus. Gegebenenfalls führt er eine Shapeanalyse (mit dem gewünschten Grad an Genauigkeit) durch. Des Weiteren initialisiert er den Visualisierer mit einer geeigneten, auf den Anwender abgestimmten Darstellung. Diese Rolle hat vieles mit der Rolle eines künstlerischen Leiters oder Regisseurs einer Aufführung gemein. Die Interaktion eines Visualisierungsinszenators mit dem Visualisierer stellt keine Lernsituation dar, deshalb lässt sich auch kein Lerngegenstand benennen.

Vorkenntnisse: Der Visualisierungsinszenator wählt sich eine vom Entwerfer entwickelte Spezifikation aus. Kenntnisse von Shapeanalyse sind also nur beschränkt erforderlich. Da er eine Shapeanalyse durchführen können muss, sollte die Bedienung von TVLA beherrscht werden. Sehr vorteilhaft ist, wenn er sich in die Rolle des Anwenders hineinversetzen kann. Der Algorithmus, der letztendlich visualisiert werden soll, wird auf wenigstens einem ähnlichen Niveau verstanden, wie die Rolle, für die die Inszenierung gedacht ist. Wenn sich die Inszenierung an Lernende im Bereich Algorithmen und Datenstrukturen richtet, sind didaktische Kenntnisse nützlich. Er kennt sich gut mit Visualisierung und mindestens gut mit dem Visualisierer aus.

Anforderungen an den Visualisierer: Diese Rolle bereitet eine Visualisierungssitzung für einen Anwender vor, der über bestimmte Vorkenntnisse verfügt und der einen bestimmten Zweck mit der Visualisierung verfolgt. Dabei versetzt er sich in die Rolle des Anwenders. Seine Wünsche an den Visualisierer sind dann die des Anwenders.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Die am stärksten ausgeprägte Interaktion findet mit dem Spezifikationsentwerfer statt: Ist keine zum beabsichtigten Zweck passende Spezifikation verfügbar, muss eine solche entwickelt werden.

Algorithmen-Datenstrukturen-Experte

Beschreibung der Rolle: Es geht darum, ein tieferes theoretisches Verständnis des Algorithmus zu erlangen. Eine Person in dieser Rolle könnte beispielsweise aussagekräftigere Schleifeninvarianten finden wollen. In diesem Beispiel bestünde die Interaktion mit dem Visualisierer im Wunsch, logisch formulierbare Eigenschaften zu finden, die alle Shapegraphen eines Programmpunktes erfüllen. Der Gegenstand des

Lernens ist der Algorithmus. (Ein Algorithmen-Datenstrukturen-Experte als Person kann auch als Lehrender auftreten.[1] Dann besteht seine Interaktion mit dem Visualisierer in der Auswahl einer Präsentation für den Lernenden. Er tritt dann aber eigentlich als Visualisierungsinszenator auf.)

Vorkenntnisse: Dass er sich bei Algorithmen und Datenstrukturen bestens auskennt, suggeriert schon der Name der Rolle. Kenntnisse im Bereich Shapeanalyse sind nur soweit erforderlich, dass er die Analyseausgabe versteht und Shapegraphen „lesen" kann. Mit dem Visualisierers kennt er sich gut aus und kann die ihm gebotenen Möglichkeiten gewinnbringend nutzen.

Anforderungen an den Visualisierer: Sie ähneln denen des Shapeanalysten und sind mit dem Begriffsfeld Such-, Filter- und Gruppierungsmöglichkeiten geeignet charakterisiert. Wenn er beispielsweise an Invarianten interessiert ist, dann sucht er Formeln, die für alle oder zumindest für eine große beziehungsweise bestimmte Teilmenge der Shapegraphenmenge eines Programmpunktes gelten; Gruppierungsfunktionen können ihn hierbei unterstützen.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Besonders ausgeprägt ist die sekundäre Interaktionen als Wissensvermittler, als Lehrender.

Algorithmen-Datenstrukturen-Lernender

Beschreibung der Rolle: Diese Rolle verkörpert die klassische Lernsituation im Fachgebiet Algorithmen und Datenstrukturen. Eine Person in dieser Rolle will verstehen, wie ein vorgegebener Algorithmus funktioniert und warum er das Gewünschte leistet. Darauf ist die Interaktion mit dem Visualisierer ausgerichtet. Der Lerngegenstand ist das Programm beziehungsweise der Algorithmus.

Vorkenntnisse: Im Bereich Shapeanalyse kennt er sich soweit aus, dass er die Ausgabe einer Analyse versteht und mit Shapegraphen vertraut ist. Die einzelnen Anweisungen des betrachteten Programmes muss er verstehen. Grundkenntnisse in der Bedienung des Visualisierers sind nötig, weitergehende Kenntnisse nützlich.

Anforderungen an den Visualisierer: Für diese Rolle ist charakteristisch, dass für den Lerngegenstand noch keine ausgereifte Abstraktion im Gehirn entwickelt wurde. Daher profitiert diese Rolle wohl am meisten von einer guten didaktischen Führung und einer effektiven Nutzung der Möglichkeiten, die Visualisierung als Methode bietet. Zu den Themenbereichen, die dabei eine besondere Beachtung verdienen, zählen: Darstellung der Shapegraphen, Navigationsmöglichkeiten durch die Analyseausgabe (durch den Transitionsgraphen), Möglichkeiten zur Klassifikation von

1. In diesem Zusammenhang können wir auch ein Buch als Experten auffassen. Eine Rolle wird zwar meistens von einer Person eingenommen, jedoch zeigt dieses Beispiel, dass dies nicht notwendigerweise der Fall sein braucht.

Shapegraphen, beispielsweise in Bezug auf den Grad ihrer Allgemeinheit, und Strukturierungsmöglichkeiten, beispielsweise Methoden zum Zusammenfassen ähnlicher (Heap-)Situationen.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Seine weiteren Interaktionen richten sich auf den Erwerb weiteren Wissens: Er interagiert mit dem Inszenator oder dem Entwickler des Visualisierers um mehr über die Bedienung des Visualisierers, er interagiert mit dem Algorithmen-Datenstrukturen-Experten, um mehr über den Algorithmus und gegebenenfalls interagiert er mit dem Inszenator oder dem Spezifikationsentwerfer um mehr über die gewählte Spezifikation zu erfahren.

Programmtester

Beschreibung der Rolle: Diese Rolle befasst sich mit der Korrektheit der Implementierung eines Algorithmus. (Die Korrektheit des Algorithmus an sich gehört zur Rolle des Algorithmen-Datenstrukturen-Experten.) Eine Person in dieser Rolle hat ein konkretes Programm vorliegen, also einen Algorithmus, der in einer konkreten Programmiersprache formuliert wurde und von einem Spezifikationsentwerfer getreu in einen TVLA-Kontrollflussgraphen umgesetzt wurde. Er will sich davon überzeugen, dass dieses Programm und der Algorithmus dasselbe berechnen, dass das Programm das Gewünschte leistet und korrekt ist. Dies bestimmt seine Interaktion mit dem Visualisierer. Der Gegenstand des Lernens ist das Programm.

Vorkenntnisse: Er kennt sich in Shapeanalyse auf einem Niveau aus, so dass er deren Ausgabe versteht und Shapegraphen „lesen“ kann. Er verfügt über ausreichende Kenntnisse im Bereich Algorithmen und Datenstrukturen um zu wissen, dass und warum der Algorithmus das Gewünschte leistet. Den Visualisierer kann er gut bedienen und seine Möglichkeiten nutzen, um die konkrete Implementierung zu untersuchen.

Anforderungen an den Visualisierer: Sie gehen in zwei Richtungen: Zum einen ist ähnlich wie beim Algorithmen-Datenstrukturen-Lernenden eine gute Qualität der Navigation durch die Analyseausgaben und deren Darstellung wünschenswert. Man vergleiche die Darstellung dort. Zum anderen profitiert er stark von Such-, Filter- und Gruppierungsfunktionen. Damit kann er bestimmte Situationen gesondert betrachten und den Rest ausblenden.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen.

Programmoptimierer

Beschreibung der Rolle: Diese Rolle befasst sich mit der Verbesserung von Teilaspekten eines Programms, die hinsichtlich vorgegebener Kriterien bewertet werden. (Die

Verbesserung des Algorithmus an sich gehört zur Rolle des Algorithmen-Datenstrukturen-Experten.) Die Rolle baut auf der Rolle des Programmtesters auf, über Fragen der Korrektheit des Programms herrscht Gewissheit. Bei der Interaktion eines Benutzers in dieser Rolle mit dem Visualisierer erhofft er sich neben Inspirationen allgemeiner Natur, den Grad der Erfülltheit der Kriterien beurteilen zu können, um so verbesserungsbedürftige Programmstellen zu finden. Der Gegenstand des Lernen-Wollens ist wie beim Programmtester das Programm.

Vorkenntnisse: Sie sind ähnlich wie beim Programmtester, jedoch in der Regel auf einem höheren Niveau. Im Bereich Shapeanalyse liegt der Kenntnisstand auf einem Niveau, so dass er deren Ausgabe versteht und Shapegraphen „lesen“ kann. Er verfügt im Gebiet Algorithmen und Datenstrukturen wenigstens über die Kenntnisse, um beurteilen zu können, dass und warum der Algorithmus das Gewünschte leistet. Er verfügt über ausreichende Kenntnisse, um für einzelne Programmteile alternative Implementierungen in Erwägung zu ziehen. Den Visualisierer kann er auf einem überdurchschnittlichen Niveau bedienen.

Anforderungen an den Visualisierer: Hier gilt das Gleiche wie beim Programmtester. Wünschenswert wäre, dass sich die Optimierungskriterien in die Visualisierung integrieren lassen. Der Grad ihrer Erfülltheit könnte für jeden Shapegraphen mitvisualisiert werden. Es sollte sich beispielsweise nach Shapegraphen suchen lassen, für welche die Kriterien nur schwach ausgeprägt sind, die also kritisch sind.

Interaktion mit anderen Rollen: Es bestehen die zuvor genannten (primären und sekundären) Wechselwirkungen. Spielt beim Zweck der Visualisierung neben der Optimierung des Programms auch die Verifikation eine Rolle, so interagiert er mit dem Programmtester.

Literaturverzeichnis

Tief ist der Brunnen der Vergangenheit.

(Thomas Mann)

Bieber 2001: Bieber, Ronald: *Alexsa. Algorithm Explanation by Shape Analysis Extensions to the TVLA System.* Saarbrücken, Universität des Saarlandes, Diplomarbeit, 2001

Brockhaus 2006: *Brockhaus-Enzyklopädie.* 21., völlig neu bearb. Aufl. Leipzig, Mannheim : Brockhaus, 2006. – ISBN 978–3–7653–4140–3

Cormen u. a. 2001: Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald L. ; Stein, Clifford: *Introduction to Algorithms.* 2. Aufl. Cambridge : The MIT Press, 2001. – ISBN 0–262–03293–7

Cox u. Roman 1992: Cox, Kenneth C. ; Roman, Gruia-Catalin: Abstraction in Algorithm Animation. In: *Proceedings of the 1992 IEEE Workshop on Visual Languages*, IEEE Computer Society, 1992. – ISBN 0–8186–3090–6, S. 18–24

Diestel 2006: Diestel, Reinhard: *Graphentheorie.* 3., neu bearb. u. erw. Aufl. Berlin, Heidelberg, New York : Springer-Verlag, 2006. – ISBN 3–540–21391–0

Dillon 1996: Dillon, Andrew: Myths, misconceptions and an alternative perspective on information usage and the electronic medium. In: Rouet, Jean-François (Hrsg.) ; Levonen, Jarmo J. (Hrsg.) ; Dillon, Andrew (Hrsg.) ; Spiro, Rand J. (Hrsg.): *Hypertext and Cognition.* Mahwah NJ : Erlbaum, 1996, S. 25–42

Duden 2007: *Duden, Deutsches Universalwörterbuch.* 6., überarb. und erw. Aufl. Mannheim : Dudenverlag, 2007. – ISBN 978–3–411–05506–7

Ebbinghaus u. a. 1996: Ebbinghaus, Heinz-Dieter ; Flum, Jörg ; Thomas, Wolfgang: *Einführung in die mathematische Logik.* 4. Aufl. Heidelberg, Berlin, Oxford : Spektrum, Akad. Verl., 1996. – ISBN 3–8274–0130–5

Gottwald 2004: Gottwald, Siegfried: *Many-Valued Logic.* Version: 2004. `http://plato.stanford.edu/entries/logic-manyvalued/`, Abruf: 2. Juli 2009 (Stanford Encyclopedia of Philosophy). – Internet-Dokument

Grimm 2004: *Der digitale Grimm : deutsches Wörterbuch von Jacob und Wilhelm Grimm.* 1. Aufl., PC-Version Juli 2004. Frankfurt am Main : Zweitausendeins,

2004. – ISBN 3–86150–628–9. – Medienpaket ; elektronische Ausgabe der Erstbearbeitung

Johannes u. a. 2005: Johannes, Dierk ; Seidel, Raimund ; Wilhelm, Reinhard: Algorithm animation using shape analysis: visualising abstract executions. In: Naps, Thomas L. (Hrsg.) ; Pauw, Wim D. (Hrsg.): *SoftVis '05: Proceedings of the 2005 ACM symposium on Software visualization*. New York, NY, USA : ACM Press, 2005. – ISBN 1–59593–073–6, S. 17–26

Kerren u. Stasko 2002: Kerren, Andreas ; Stasko, John T.: Algorithm Animation – Introduction. In: Diehl, Stephan (Hrsg.): *Software Visualization, International Seminar Dagstuhl Castle, Germany, May 20-25, 2001, Revised Lectures* Bd. 2269, Springer, 2002 (Lecture Notes in Computer Science). – ISBN 3–540–43323–6, S. 1–15

Klavins 2003: Klavins, Gints: *Algorithm Visualization Using Shape Analysis*. Saarbrücken, Universität des Saarlandes, Diplomarbeit, 2003

Lee u. a. 2005: Lee, Oukseh ; Yang, Hongseok ; Yi, Kwangkeun: Automatic Verification of Pointer Programs Using Grammar-Based Shape Analysis. In: Sagiv, Shmuel (Hrsg.): *14th European Symposium on Programming, ESOP 2005* Bd. 3444, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 3–540–25435–8, S. 124–140

Lev-Ami 2000: Lev-Ami, Tal: *TVLA: A Framework for Kleene Logic Based Static Analysis*. Tel-Aviv, Israel, Tel-Aviv University, Department of Computer Science, Diplomarbeit, 2000

Lev-Ami u. a. 2007: Lev-Ami, Tal ; Manevich, Roman ; Sagiv, Mooly: *TVLA: User's Manual*. 2007. – Bestandteil von TVLA, siehe TVLA [2007]

Lev-Ami u. a. 2004: Lev-Ami, Tal ; Manevich, Roman ; Sagiv, Shmuel: TVLA: A system for generating abstract interpreters. In: Jacquart, René (Hrsg.): *IFIP Congress Topical Sessions*, Kluwer, 2004, S. 367–376

Lev-Ami u. a. 2000: Lev-Ami, Tal ; Reps, Thomas ; Sagiv, Mooly ; Wilhelm, Reinhard: Putting static analysis to work for verification: A case study. In: *ISSTA '00: Proceedings of the 2000 ACM SIGSOFT international symposium on Software testing and analysis*. New York : ACM Press, 2000. – ISBN 1–58113–266–2, S. 26–38

Lev-Ami u. Sagiv 2000: Lev-Ami, Tal ; Sagiv, Shmuel: TVLA: A System for Implementing Static Analyses. In: *SAS* Bd. 1824, Springer-Verlag, 2000 (Lecture Notes in Computer Science). – ISBN 3–540–67668–6, S. 280–301

Loginov u. a. 2006: Loginov, Alexey ; Reps, Thomas W. ; Sagiv, Mooly: Automated Verification of the Deutsch-Schorr-Waite Tree-Traversal Algorithm. In: Yi, Kwangkeun (Hrsg.): *Static Analysis, 13th International Symposium, SAS 2006* Bd. 4134, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 3–540–37756–5, S. 261–279

Manevich 2003: Manevich, Roman: *Data Structures and Algorithms for Efficient Shape Analysis.* Tel-Aviv, Israel, Tel-Aviv University, School of Computer Science, Diplomarbeit, 2003

Manevich u. a. 2005: Manevich, Roman ; Yahav, Eran ; Ramalingam, G. ; Sagiv, Mooly: Predicate Abstraction and Canonical Abstraction for Singly-Linked Lists. In: Cousot, Radhia (Hrsg.): *Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation, VMCAI 2005* Bd. 3148, Springer, 2005 (Lecture Notes in Computer Science), S. 181–198

Markowitsch 2002: Markowitsch, Hans J.: *Dem Gedächtnis auf der Spur : vom Erinnern und Vergessen.* Darmstadt : Primus-Verlag, 2002. – ISBN 3–89678–447–1

Michail 1996: Michail, Amir: Teaching Binary Tree Algorithms through Visual Programming. In: *Proceedings of the 1996 IEEE Symposium on Visual Languages*, IEEE Computer Society, 1996. – ISBN 0–8186–7508–X, S. 38–45

OPSIS 1996: *Teaching Binary Search Tree Algorithms through Programming, Proof, and Animation.* `http://opsis.sourceforge.net/`. Version: 1996, Abruf: 6. Juli 2009

Ottmann u. Widmayer 2002: Ottmann, Thomas ; Widmayer, Peter: *Algorithmen und Datenstrukturen.* 4. Aufl. Heidelberg, Berlin : Spektrum Akademischer Verlag, 2002. – ISBN 3–8274–1029–0

Parduhn 2005: Parduhn, Sascha: *Algorithm Animation Using Shape Analysis with Special Regard to Binary Trees.* Saarbrücken, Universität des Saarlandes, Diplomarbeit, 2005

Reineke 2005: Reineke, Jan: *Shape Analysis of Sets.* Saarbrücken, Universität des Saarlandes, Diplomarbeit, 2005

Reineke 2006: Reineke, Jan: Shape Analysis of Sets. In: Autexier, Serge (Hrsg.) ; Merz, Stephan (Hrsg.) ; Torre, Leon van d. (Hrsg.) ; Wilhelm, Reinhard (Hrsg.) ; Wolper, Pierre (Hrsg.): *Workshop „Trustworthy Software“ 2006*, Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, 2006. – ISBN 978–3–939897–02–6

Reps u. a. 2002: Reps, Thomas ; Sagiv, Mooly ; Wilhelm, Reinhard: Shape Analysis and Applications. In: Srikant, Y. N. (Hrsg.) ; Shankar, Priti (Hrsg.): *The Compiler Design Handbook: Optimizations & Machine Code Generation.* CRC Press, 2002, S. 175–217

Reps u. a. 2004: Reps, Thomas W. ; Sagiv, Shmuel ; Wilhelm, Reinhard: Static Program Analysis via 3-Valued Logic. In: *CAV 2004* Bd. 3114, 2004 (LNCS), S. 15–30

Sagiv u. a. 1996: Sagiv, Mooly ; Reps, Thomas ; Wilhelm, Reinhard: Solving shape-analysis problems in languages with destructive updating. In: *POPL '96:*

Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages. New York : ACM Press, 1996. – ISBN 0–89791–769–3, S. 16–31

Sagiv u. a. 1998: Sagiv, Mooly ; Reps, Thomas ; Wilhelm, Reinhard: Solving Shape-Analysis Problems in Languages with Destructive Updating. In: *ACM Transactions on Programming Languages and Systems* 20 (1998), Nr. 1, S. 1–50

Sagiv u. a. 1999: Sagiv, Mooly ; Reps, Thomas ; Wilhelm, Reinhard: Parametric shape analysis via 3-valued logic. In: *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. New York : ACM Press, 1999. – ISBN 1–58113–095–3, S. 105–118

Sagiv u. a. 2002: Sagiv, Mooly ; Reps, Thomas ; Wilhelm, Reinhard: Parametric shape analysis via 3-valued logic. In: *ACM Trans. Program. Lang. Syst.* 24 (2002), Nr. 3, S. 217–298. – ISSN 0164–0925

Schmidt 1982: Schmidt, Heinrich: *Philosophisches Wörterbuch*. 21. Aufl., neu bearb. von Georgi Schischkoff. Stuttgart : Alfred Kröner Verlag, 1982. – ISBN 3–520–01321–5

Schulmeister 2002: Schulmeister, Rolf: *Grundlagen hypermedialer Lernsysteme : Theorie – Didaktik – Design*. 3., korr. Aufl. München, Wien : Oldenbourg Verlag, 2002. – ISBN 3–486–25864–8

Schumann u. Müller 2000: Schumann, Heidrun ; Müller, Wolfgang: *Visualisierung : Grundlagen und allgemeine Methoden*. Berlin, Heidelberg : Springer-Verlag, 2000. – ISBN 3–540–64944–1

Smith 2007: Smith, Robin: *Aristotle's Logic*. Version: 2007. `http://plato.stanford.edu/entries/aristotle-logic/`, Abruf: 2. Juli 2009 (Stanford Encyclopedia of Philosophy). – Internet-Dokument

Spitzer 2003: Spitzer, Manfred: *Lernen : Gehirnforschung und die Schule des Lebens*. Korrigierter Nachdruck. Heidelberg, Berlin : Spektrum, Akademischer Verlag, 2003. – ISBN 3–8274–1396–6

Squire u. Kandel 1999: Squire, Larry R. ; Kandel, Eric R.: *Gedächtnis : die Natur des Erinnerns*. Heidelberg, Berlin : Spektrum, Akademischer Verlag, 1999. – ISBN 3–8274–0522–X

Stangl 2009: Stangl, Werner: *Werner Stangls Arbeitsblätter*. Version: 2009. `http://arbeitsblaetter.stangl-taller.at`, Abruf: 2. Juli 2009. – Internet-Dokument

TANGO 1990: *TANGO: A Framework and System for Algorithm Animation. (Homepage des Projekts und seiner Nachfolger)*. `http://www.cc.gatech.edu/gvu/softviz/algoanim/algoanim.html`. Version: 1990, Abruf: 4. Juli 2009

TVLA 2007: *TVLA: 3-Valued Logic Analysis Engine*. `http://www.cs.tau.ac.il/~tvla/`. Version: 2007, Abruf: 2. Juli 2009

Wilhelm u. a. 2002a: Wilhelm, Reinhard ; Müldner, T. ; Seidel, Raimund: Algorithm Explanation: Visualizing Abstract States and Invariants. In: Diehl, S. (Hrsg.): *Software Visualization*. Springer-Verlag, 2002 (LNCS), S. 381–394

Wilhelm u. a. 2002b: Wilhelm, Reinhard ; Reps, Thomas W. ; Sagiv, Shmuel: Shape Analysis and Applications. In: Srikant, Y. N. (Hrsg.) ; Shankar, Priti (Hrsg.): *The Compiler Design Handbook*. CRC Press, 2002. – ISBN 0–8493–1240–X, S. 175–218

Wilhelm u. a. 2000: Wilhelm, Reinhard ; Sagiv, Mooly ; Reps, Thomas: Shape Analysis. In: *International Conference on Compiler Construction*, Springer-Verlag, 2000 (LNCS 1781), S. 1–16