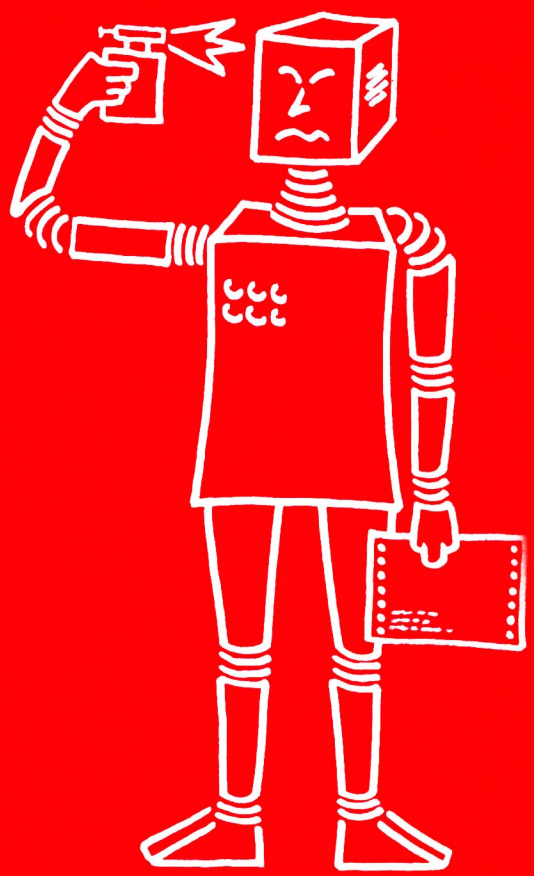
SEKI-PROJEKT

SEKI MEMO

Institut für Informatik III
Universität Bonn
Bertha-von-Suttner-Platz 6
D 5300 Bonn 1, W. Germany

Institut für Informatik I
Universität Karlsruhe
Postfach 6380
D-7500 Karlsruhe 1, W. Germany



MEMO SEKI-BN-81-04

PROGRAMMSYNTHESE AUS BEISPIELSFOLGEN

von

Christoph Beierle

PROGRAMMSYNTHESE

AUS

BEISPIELSFOLGEN

Christoph Beierle

Institut für Informatik III
Universität Bonn
Bertha-von-Suttner-Platz 6
D-5300 Bonn 1

Dezember 1980

Inhaltsverzeichnis

Abstract

Automatic program synthesis from example computations is investigated. Four kinds of instruction and store description sequences are distinguished; for each the problem of program synthesis is shown to be NP-complete. NP-completeness is shown also for various restricted subclasses of the general problem as well as for the synthesis of Turing machines and Mealy automatons from traces of their behavior. A decision procedure for solvability of program synthesis problems is presented. A universal program synthesis algorithm applicable to different kinds of example computations and yielding minimal programs is designed and verified. Due to its representation methods and extensive pruning techniques it solves a large class of program synthesis problems efficiently.

Kapitel 1 EINLEITUNG

Verschiedene Ansätze zur automatischen Programmsynthese wurden bisher untersucht, z.B. [Gill 1966], [Amarel 1971], [Waldinger 1969], [Manna,Waldinger 1971], [Lee et al. 1974], [Green, Barstow 1975], [Barstow 1977], [Biermann 1972], [Biermann, Krishnaswamy 1976]. [Biermann 1976] gibt einen Überblick über die unterschiedlichen Vorgehensweisen.

Die in der vorliegenden Arbeit angenommene Ausgangssituation der Programmsynthese aus Beispielsfolgen ist nur mit dem Ansatz von Biermann [Biermann, Krishnaswamy 1976] vergleichbar. Dort wird aus einer Menge $\{C_1,\ldots,C_n\}$ von aus Instruktionen bestehenden Berechnungsfolgen eines Programms P ein bzgl. der Anzahl der Zuweisungsstatements minimales Programm P' konstruiert, sodaß die C_i auch von P' erzeugt werden können. In den Eingabefolgen C_i müssen neben den Zuweisungen zusätzlich alle erfolgreich ausgeführten Tests mit angegeben sein.

In dieser Arbeit werden nicht nur Instruktionsfolgen, sondern auch Speicherbelegungsfolgen als Eingabe für einen Synthesealgorithmus zugelassen. Darüberhinaus wird zwischen Folgen unterschieden, die entweder auch Testinstruktionen oder aber lediglich Zuweisungsinstruktionen berücksichtigen, sodaß sich insgesamt vier verschiedene Arten von Beispielsfolgen - je zwei für Instruktions- und Speicherbelegungsfolgen - ergeben. Dementsprechend werden verschiedene Realisierungsbegriffe zur Präzisierung der an das zu synthetisierende Programm zu stellenden Anforderungen eingeführt.

Während der Ansatz von Biermann von vorneherein nur durch ein Programm erzeugte Eingabefolge zuläßt (entsprechendes gilt auch für die Synthese von Turingmaschinen in [Hwa, Wrightson 1973]), besteht in der vorliegenden Arbeit ein Programmsyntheseproblem (PSP) aus einer Menge von beliebigen Beispielsfolgen. Für jedes PSP Q ist entscheidbar, ob überhaupt ein Programm existiert, das die Beispielsfolgen von Q erzeugen kann. Ist das der Fall, so liefert der in Kap. 3 - 5 entwickelte Programmsynthesealgorithmus (PSA) PA ein Programm P, das eine minimale Realisierung von Q darstellt; andernfalls liefert PA die leere Menge.

Mit Hilfe der Methoden der Programmverifikation (z.B. [Manna 1974]) werden Korrektheit, Vollständigkeit und Lösungsminimalität von PA nachgewiesen. Da bei der Definition von PA keine spezifischen Annahmen über die den PSP zugrunde liegenden Speicherbelegungen oder die Zuweisungs- und Testinstruktionen gemacht werden, ist PA für beliebige Berechnungsmodelle anwendbar.

Komplexitätsberechnungen wurden in den bisherigen Arbeiten zur Programmsynthese aus Beispielsfolgen ([Biermann 1972], [Hwa, Wrightson 1973], [Raulefs 1973], [Biermann, Baum, Petry 1975], [Biermann, Krishnaswamy 1976]) nicht vorgenommen, und zwar weder für die dort entwickelten Algorithmen noch für die Problemstellung selbst.

In dieser Arbeit wird gezeigt, daß das allgemeine Problem der Programmsynthese aus Beispielsfolgen NP-vollständig

([Cook 1971], [Garey, Johnson 1979]) ist. Sogar bei einer Reihe von vereinfachenden Einschränkungen etwa bzgl. der Menge der Zuweisungen und Tests, der Speicherbelegungen und der Länge und Anzahl der Beispielsfolgen bleibt die NP-Vollständigkeit der Programmsynthese erhalten. Diese Ergebnisse werden für die verschiedenen Arten von PSP aus Instruktions- und aus Speicherbelegungsfolgen nachgewiesen und anschliessend auf die Synthese von Turingmaschinen (und damit auch von Mealy-Automaten) aus Beispielsfolgen übertragen. So ist z.B. das Ja/Nein-Entscheidungsproblem, ob es zu einem $k \in \mathbb{N}$ und einer einzelnen Folge C von Ein/Ausgabepaaren über einem festen Alphabet mit nur 5 verschiedenen Zeichen einen Mealy-Automaten mit k Zuständen gibt, der das durch C gegebene Ein/Ausgabe-Verhalten besitzt, NP-vollständig (vgl. Abschnitt 6.6).

Die NP-Vollständigkeit einer Problemstellung impliziert unter der Annahme P-TIME $\neq$ NP-TIME, daß der Zeitaufwand eines jeden Algorithmus für diese Problemstellung im Grenzverhalten für immer größer werdende Eingaben schneller als polynomial anwächst [Garey, Johnson 1979]. Für die Programmsynthese aus Beispielsfolgen folgt daher, daß es keinen PSA geben kann (vorausgesetzt P-TIME $\neq$ NP-TIME), der bzgl. der Menge aller PSP mit polynomialer Zeitkomplexität arbeitet. Ziel eines PSA muß es daher sein, eine möglichst große Anzahl von PSP effizient zu lösen.

In dieser Arbeit können keine experimentellen Angaben über den benötigten Rechenzeitaufwand von PA gemacht werden, da

eine Implementierung von PA noch nicht vorliegt. Die in PA integrierten Methoden und Heuristiken der statischen und dynamischen Suchraumreduktion und der Bildung, Sortierung, Auswahl und Überprüfung von Hypothesen stellen jedoch sicher, daß zumindest für eine Großzahl von PSP PA effizient arbeitet, was anhand von verschiedenen Beispielen belegt wird.

Von den Ergebnissen dieser Arbeit ausgehend stellt sich die Aufgabe, durch Implementierung von PA statistische Angaben über das Rechenzeitverhalten von PA zu gewinnen. Im Hinblick auf die Komplexitätsbetrachtungen bietet sich an, neben den in dieser Arbeit schon aufgeführten Einschränkungen des allgemeinen Problems der Programmsynthese aus Beispielsfolgen weitere Einschränkungen zu finden, für die man gerade noch die Zugehörigkeit zu P-TIME zeigen kann, um so die Grenze zwischen P-TIME und NP-TIME (bzw. NP-Vollständigkeit) bzgl. dieser Problemstellung genauer zu bestimmen.

Es folgt ein kurzer Überblick über den Aufbau dieser Arbeit:
In Kapitel 2 werden die grundlegenden Begriffe eingeführt und präzisiert.
In Kapitel 3 werden für PSP Repräsentationen entwickelt, die die Klasse der in Frage kommenden Programme von vorneherein stark einschränken und in geeigneter Weise in PA eingesetzt werden können. Mit dem sog. Realisierbarkeitssatz wird ein effizientes Verfahren zur Entscheidung der 'Lösbarkeit' eines PSP geliefert.

Kapitel 4 beschreibt die in PA verwandten Strategien und Methoden. Die Eigenschaften der hier entworfenen Verfahren zur Suchraumreduktion, Hypothesenmanipulation usw. werden nachgewiesen.

In Kapitel 5 wird der PSA PA definiert. Die von ihm geforderten Eigenschaften werden verifiziert und anhand eines Beispiels erläutert, gefolgt von einigen Hinweisen für eine Implementierung von PA.

In Kapitel 6 sind die Komplexitätsuntersuchungen zur Programmsynthese aus Beispielsfolgen sowie die Sätze zum Nachweis der NP-Vollständigkeit verschiedener Einschränkungen davon enthalten.

Im Anschluß an das Literaturverzeichnis folgt ein Index zu den Seitenzahlen der einzeln nummerierten Definitionen, Sätze, Beispiele und Abbildungen dieser Arbeit.

Kapitel 2 GRUNDLAGEN UND DEFINITIONEN

Es werden das Berechnungsmodell, das den Überlegungen dieser Arbeit zugrunde liegt, sowie die Problemstellungen, die untersucht werden, präzisiert. Da die entwickelten Algorithmen und Verfahren keine spezifischen Annahmen über den Aufbau des vorgegebenen Berechnungsmodells machen, sind die in 2.1 vorhandenen Definitionen ausreichend. Nach Formalisierung der Begriffe "Programmsyntheseproblem" (PSP) und "Programmsynthesealgorithmus" (PSA) in 2.2 und 2.3 werden die Zielsetzungen für den in dieser Arbeit entworfenen PSA PA angegeben (2.4).

2.1 Programme und Programmberechnungen

Definition 2.1

Σ sei ein endliches nicht-leeres Alphabet, STORES $\subseteq \Sigma^*$.

1. STORES ist die Menge der Speicherzustände oder Speicherbelegungen.
2. Eine Abbildung A : STORES —> STORES heißt Zuweisungsinstruktion.
3. Eine Abbildung T : STORES —> {true,false} heißt Testinstruktion.

4. ASSIGNMENTS = $\{A_1,\ldots,A_p\}$ ist eine endliche, nichtleere Menge von Zuweisungsinstruktionen.
5. TESTS = $\{T_1,\ldots,T_q\}$ ist eine endliche, nichtleere Menge von Testinstruktionen.
6. INSTRUCTIONS = ASSIGNMENTS $\cup$ TESTS ist die Menge der Instruktionen, wobei START,STOP,dummy $\notin$ INSTRUCTIONS.
7. Für eine Menge IM $\subseteq$ INSTRUCTIONS $\cup$ {START,STOP} ist:
   $IM^0 = IM \setminus \{START,STOP\}$
   $IM^1 = IM \cup \{START,STOP\}$

Beispiel 2.1

Für fast alle Beispiele dieser Arbeit gelten die folgenden Vereinbarungen:

1. Die Menge der Speicherzustände ist gegeben durch die Menge aller geordneten 5-Tupel über den nicht-negativen ganzen Zahlen: STORES = $(\mathbb{N} \cup \{0\})^5$.
2. ASSIGNMENTS besteht aus insgesamt 30 Zuweisungsinstruktionen. Für $i,j \in \{1,\ldots,5\}$ sind dies:
   (2.1) A1(i) inkrementiert die i-te Komponente der gegebenen Speicherbelegung um 1:
   $A1(i) = \lambda s.\ s = (s_1,\ldots,s_i,\ldots,s_5) \longrightarrow (s_1,\ldots,s_i+1,\ldots,s_5)$
   (2.2) S1(i) dekrementiert die i-te Komponente um 1:
   $S1(i) = \lambda s.\ s = (s_1,\ldots,s_i,\ldots,s_5) \longrightarrow (s_1,\ldots,s_i-1,\ldots,s_5)$
   (2.3) CL(i) setzt die i-te Komponente auf 0:
   $CL(i) = \lambda s.\ s = (s_1,\ldots,s_i,\ldots,s_5) \longrightarrow (s_1,\ldots,0,\ldots,s_5)$

(2.4) A(i,j) weist der i-ten Komponente den Wert der Addition aus i-ter und j-ter Komponente zu:

$A(i,j) = \lambda s.\ s = (s_1,\ldots,s_i,\ldots,s_j,\ldots,s_5) \longrightarrow (s_1,\ldots,s_i+s_j,\ldots,s_j,\ldots,s_5)$

5. TESTS enhält 5 verschiedene Testinstruktionen.
   ZR(i) liefert true, wenn die i-te Komponente 0 ist, false sonst.
   $ZR(i) = \lambda s.\ s = (s_1,\ldots,s_i,\ldots,s_5) \longrightarrow (s_i = 0 \longrightarrow \underline{true},\ s_i \neq 0 \longrightarrow \underline{false})$

Definition 2.2:

Sei K = (l,I) mit $l \in \mathbb{N}$.

1. K heißt Zuweisungsknoten <=> I ε ASSIGNMENTS.
2. K heißt Testknoten <=> I ε TESTS.
3. K heißt Startknoten <=> $l = 1 \wedge I = \underline{START}$.
4. K heißt Stopknoten <=> $l = 1 \wedge I = \underline{STOP}$.
5. K heißt Instruktionsknoten <=> I ε INSTRUCTIONS.
6. In allen Fällen heißt l Label von K (Schreibweise: LABEL(K) = l). K wird auch als I-Knoten bezeichnet (INSTR(K) = I) oder allgemeiner als Programmknoten.
7. PNODES ist die Menge aller Programmknoten.

Definition 2.3

(K_1,m,K_2) heißt Programmtransition <=>

(i) K_1 ist Instruktions- oder Startknoten

(ii) K_2 ist Instruktions- oder Stopknoten

(iii) $m \in \{true,false,nil\} \wedge (m = nil \iff INSTR(K_1) \notin TESTS)$

TFN = {true,false,nil} ist die Menge der Transitionsmarkierungen.

Mit den Definitionen 2.1 - 2.3 wird ein Flußprogramm definiert, wie es z.B. durch einen gerichteten, markierten Graphen angegeben werden kann, wobei die Knoten dieses Graphen mit START,STOP oder I ε INSTRUCTIONS markiert sind und die Kanten entsprechend mit true,false oder nil. Die hier gewählte, äquivalente Darstellungsart für Programme läßt sich leicht in ein Flußprogramm im üblichen Sinne transformieren.

Definition 2.4

P sei eine endliche Menge von Programmtransitionen.

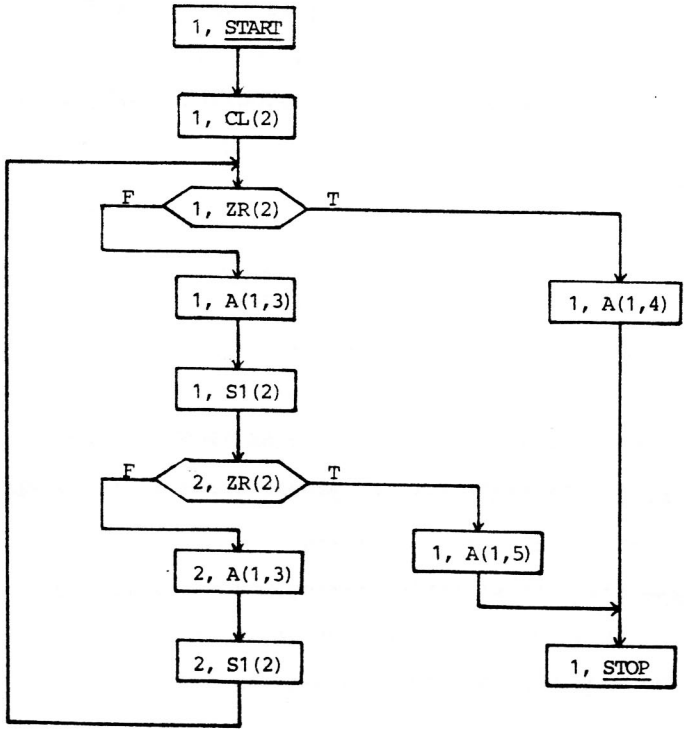
1. P heißt (Fluß-) Programm <=>
   (i) P enthält genau einen START- und einen STOP-Knoten.
   (ii) Für jeden Knoten K von P gibt es in P einen Weg vom START- zum STOP-Knoten, der durch K geht.
   (iii) $\forall (K,m,K'),(K,m',K'') \in P.\ m = m' \Rightarrow K' = K''$
2. PROGRAMS bezeichnet die Menge aller Programme.
3. M sei eine Menge von Tripeln.
   $NODES(M) = \{K \mid \exists K',m.\ (K,m,K') \in M \vee (K',m,K) \in M\}$

Beispiel 2.2

Das hier angegebene Programm P wird in späteren Beispielen benützt.

P := {((1,START),nil,(1,CL(1))),((1,CL(1)),nil,(1,ZR(2))),
((1,ZR(2)),true,(1,A(1,4))),((1,ZR(2)),false,(1,A(1,3))),
((1,A(1,3)),nil,(1,S1(2))),((1,S1(2))),nil,(2,ZR(2))),
((2,ZR(2)),true,(1,A(1,5))),((2,ZR(2)),false,(2,A(1,3))),
((2,A(1,3)),nil,(2,S1(2))),((2,S1(2))),nil,(1,ZR(2))),
((1,A(1,4)),nil,(1,STOP)),((1,A(1,5)),nil,(1,STOP))}

In üblicher Flußprogrammnotation ist P gegeben durch:



Die verlangte Eigenschaft für Programme impliziert, daß die Programmtransitionen eines Programms P im folgenden Sinne eindeutig sind:

Sei (K,m,K') ε P .

(i) Ist K Start- oder Zuweisungsknoten, so ist nach Def. 2.3 m = nil. Nach Def. 2.4 folgt, daß es nur ein K' geben kann, sodaß (K,m,K') ε P.

(ii) Ist K Testknoten, so ist nach Def. 2.3 m = true oder m = false. Ist m = true, so ist nach Def. 2.4 K' eindeutig bestimmt; ebenso ist dies im Fall m = false.

Aus dieser Eigenschaft folgt, daß die in Def. 2.5 eingeführte Übergangsfunktion Φ rechtseindeutig ist. Φ bildet zusammen mit der Berechnungsfunktion Γ, die die jeweilige Instruktion auf die gegebene Speicherbelegung anwendet, das eingangs erwähnte Berechnungsmodell.
Definition 2.6 legt fest, wie ein Programm ausgeführt wird.

Definition 2.5:

BM = (Γ, Φ) heißt Berechnungsmodell, wobei gilt:

1. Γ : INSTRUCTIONS × STORES —> TFN × STORES
heißt Berechnungsfunktion und ist definiert durch:
Γ = λI. λs. IεASSIGNMENTS —> (nil,I(s))
IεTESTS —> (I(s),s)

2. Φ : PROGRAMS × PNODES × TFN —> PNODES
heißt Übergangsfunktion und ist definiert durch:
Φ = λP. λK. λm. (K,m,K')εP —> K'

Definition 2.6

Sei P ε PROGRAMS, s ε STORES

$\underline{c(P,s)} = ((l_0,I_0),(t_1,s_1),(l_1,I_1),\ldots,(t_n,s_n),(l_n,I_n))$
heißt vollständige, terminierende Berechnungsfolge von P bei s <=>

(i) $(l_0,I_0) = (1,\underline{START})$

(ii) $(t_1,s_1) = (\underline{nil},s)$

(iii) $\forall\, i\varepsilon\{2,\ldots,n\}.\ (t_i,s_i) = \Gamma(I_{i-1},s_{i-1})$

(iv) $\forall\, i\varepsilon\{1,\ldots,n\}.\ (l_i,I_i) = \Phi(P,(l_{i-1},I_{i-1}),t_i)$

(v) $(l_n,I_n) = (1,\underline{STOP})$

Wenn im folgenden von Berechnungsfolge gesprochen wird, so ist damit immer eine vollständige, terminierende Berechnungsfolge gemeint. Eine Berechnungsfolge stellt ein ausführliches Protokoll einer Programmausführung dar: neben den berechneten Zwischenwerten (s_i) werden alle jeweils ausgeführten Zuweisungen und Tests (I_i) angegeben sowie die Ergebnisse der ausgeführten Tests (t_i). Darüberhinaus wird durch die l_i protokolliert, welchem Programmknoten (von eventuell mehreren I_i-Knoten) die ausgeführte I_i-Instruktion entspricht. Die folgende Definition ist sinnvoll, wenn nur ein Teil dieser Informationen über die Programmausführung von P bei s vorliegen.

Definition 2.7

Sei c(P,s) Berechnungsfolge von P bei Eingabe s mit den Bezeichnungen von Def. 2.6.

1. $\underline{stmts(P,s)} = (I_0,\ldots,I_n)$ heißt Statementfolge von P bei s.
2. $\underline{assign(P,s)} = proj_{\downarrow A}\ (stmts(P,s))$ heißt Zuweisungsfolge von P bei s, wobei A = ASSIGNMENTS $\cup$ {START,STOP}
3. $\underline{ss\text{-}stores(P,s)} = (s_1,\ldots,s_n)$ heißt statement-getreue Speicherbelegungsfolge von P bei s.
4. $\underline{as\text{-}stores(P,s)} = \langle s_{i_r} \mid r = 1,2,\ldots; i_r < i_{r+1}; t_{i_r} = \underline{nil}\rangle$ heißt zugeweisungsgetreue Speicherbelegungsfolge von P bei s.
5. $\underline{val(P,s)} = s_n$ heißt Ausgabewert von P bei s.

Die Statementfolge von P bei s ist also eine mit START beginnende und STOP endende Folge, die alle bei der Berechnung von P(s) ausgeführten Zuweisungen und Tests in der entsprechenden Reihenfolge wiedergibt. Die zugehörige Zuweisungsfolge erhält man aus der Statementfolge, wenn man aus letzterer genau alle Testinstruktionen entfernt. Die statementgetreue Speicherbelegungsfolge erhält man, indem man die Folge der Speicherbelegungen betrachtet, die sich nach Ausführung einer Zuweisung oder eines Tests ergeben. Aufgrund der Definition des Berechnungsmodell BM ergibt sich, daß eine Testinstruktion eine Speicherbelegung nicht unmittelbar verändern kann. Streicht man in der statementgetreuen Speicherbelegungsfolge alle die Speicherbelegungen, die sich unmittelbar nach Aus-

führung einer Testinstruktion ergeben haben, so erhält man die zugehörige zuweisungsgetreue Speicherbelegungsfolge. Der Ausgabewert von P bei s ist die Speicherbelegung, die bei Erreichen des STOP-Knoten erzeugt worden ist.

Mithilfe der in Def. 2.7 eingeführten Begriffe werden auf der Menge der Programme verschiedene Äquivalenzrelationen definiert. Diese Relationen bilden die Grundlage für die Kriterien, die die Korrektheit eines Programmsynthesealgorithmus bestimmen. Da ein Programmsynthesealgorithmus normalerweise nur eine endliche Menge von Beispielsrechnungen als Eingabe erhält, werden die Relationen nicht nur bezüglich aller möglichen Eingabewerte, sondern allgemeiner bzgl. beliebiger Teilmengen der Eingabewerte definiert.

Definition 2.8

Seien $P_1, P_2 \in$ PROGRAMS, $S \subseteq$ STORES.

(1) P_1 und P_2 heißen statement-äquivalent bzgl. S <=>
$\forall s \in S.\ stmts(P_1,s) = stmts(P_2,s)$.

(2) P_1 und P_2 heißen zuweisungs-äquivalent bzgl. S <=>
$\forall s \in S.\ assign(P_1,s) = assign(P_2,s)$.

(3) P_1 und P_2 heißen statement-speicher-äquivalent bzgl. S
<=> $\forall s \in S.$ ss-stores(P_1,s) = ss-stores(P_2,s).

(4) P_1 und P_2 heißen zuweisungs-speicher-äquivalent bzgl. S
<=> $\forall s \in S.$ as-stores(P_1,s) = as-stores(P_2,s).

(5) P_1 und P_2 heißen ausgabe-äquivalent bzgl. S
<=> $\forall s \in S.\ val(P_1,s) = val(P_2,s)$.

Der folgende Satz gibt die bestehenden Relationen zwischen diesen Äquivalenzbegriffen an.

Satz 2.1

Sei S wie in Def. 2.8 gegeben.

(1) Jede der fünf in Def. 2.8 eingeführten Relationen ist eine Äquivalenzrelation auf der Menge PROGRAMS.

(2) Sind (i) - (v) die Aussagen:

(i) P_1 und P_2 sind statement-äquivalent bzgl. S

(ii) P_1 und P_2 sind zuweisungs-äquivalent bzgl. S

(iii) P_1 und P_2 sind statement-speicher-äquivalent bzgl. S

(iv) P_1 und P_2 sind zuweisungs-speicher-äquivalent bzgl. S

(v) P_1 und P_2 sind ausgabe-äquivalent bzgl. S_1

so gelten die folgenden Implikationen:

(2.1) (i) => (ii) $\wedge$ (iii) $\wedge$ (iv) $\wedge$ (v)

(2.2) (ii) => (iv) $\wedge$ (v)

(2.3) (iii) => (iv) $\wedge$ (v)

(2.4) (iv) => (v).

Beweis: folgt unmittelbar aus Def. 2.6 - 2.8.

2.2 Programmsyntheseprobleme

Eingaben zu den in 2.3 definierten Programmsynthesealgorithmen sind Beispielsberechnungen. Während Beispielsberechnungen z.B. von einem Programm P erzeugt werden können, werden sie in Def. 2.9 zunächst allgemeiner unabhängig von der Menge der Programme definiert.

Definition 2.9

(1) S heißt Speicherbelegungsfolge <=> $S \in STORES^+$.

(2) (s,IS) heißt (vollständige) Statementbeispielsfolge <=>
$s \in STORES \wedge IS \in$ START INSTRUCTIONS* STOP .

(3) (s,AS) heißt (vollständige) Zuweisungsbeispielsfolge <=>
$s \in STORES \wedge AS \in$ START ASSIGNMENTS* STOP .

In Def. 2.10 wird die Eigenschaft von Programmen formalisiert, die man vom Ausgabeprogramm eines Synthesealgorithmus immer verlangen muß, daß nämlich das synthetisierte Programm das Verhalten besitzt, wie es durch die eingegebenen Beispiele repräsentiert wird. Die eingegebenen Beispielsberechnungen müssen also auch Berechnungen des erzeugten Programms sein. Da hier vier verschiedene Arten von Eingaben betrachtet werden, nämlich Statementbeispielsfolgen, Zuweisungsbeispielsfolgen, Speicherbelegungsfolgen, die als statement-speicher-getreu interpretiert werden, und Speicherbelegungsfolgen, die als zuweisungs-speicher-getreu interpretiert werden, werden dementsprechend auch vier verschiedene Realisierungsbegriffe für Eingabefolgen unterschieden.

Definition 2.10

B_{IS} sei Menge von Statementbeispielsfolgen,
B_{AS} Menge von Zuweisungsbeispielsfolgen und
B_S Menge von Speicherbelegungsfolgen, $P \in PROGRAMS$.

(1) P realisiert B_{IS} statement-getreu <=>
$\forall (s,IS) \in B_{IS}.\ IS = stmts(P,s)$.

(2) P realisiert B_{AS} zuweisungs-getreu <=>
$\forall (s,AS) \in B_{AS}.\ AS = assign(P,s)$.

(3) P realisiert B_S statement-speicher-getreu <=>
$\forall (s_1,\ldots,s_n) \in B_S.\ (s_1,\ldots,s_n) =$ ss-stores(P,s_1) .

(4) P realisiert B_S zuweisungs-speicher-getreu <=>
$\forall (s_1,\ldots,s_n) \in B_S.\ (s_1,\ldots,s_n) =$ as-stores(P,s_1) .

Entsprechend heißt B_{IS} (bzw. B_{AS},B_S,B_S) statement-getreu (bzw. zuweisungs-getreu, statement-speicher-getreu, zuweisungs-speicher-getreu) realisierbar genau dann, wenn es ein $P \in PROGRAMS$ gibt, das die Bedingung unter (1) (bzw. (2),(3),(4)) erfüllt.

Ein weiteres Kriterium für die Leistungsfähigkeit eines Programmsynthesealgorithmus ist die Komplexität der erzielten Lösungen. Als Komplexitätsmaß für Programme dienen i.f. zwei verschiedene Längenmaße.

Definition 2.11

Sei P ε PROGRAMS .

(1) $L_S(P) = |\{K | K \varepsilon NODES(P) \wedge INSTR(K) \varepsilon INSTRUCTIONS\}|$
ist die Anzahl der Instruktionsknoten von P.

(2) $L_A(P) = |\{K | K \varepsilon NODES(P) \wedge INSTR(K) \varepsilon ASSIGNMENTS\}|$
ist die Anzahl der Zuweisungsknoten von P.

Es ist zu beachten, daß bei beiden Längenmaßen START- und STOP-Knoten nicht mitgezählt werden, da START, STOP ∉ INSTRUCTIONS.

Damit ergibt sich folgende Definition:

Definition 2.12

Sei B_{IS}, B_{AS}, B_S und P wie in Def. 2.10,
$A_{IS} = \{s | \exists\ IS.\ (s,IS) \varepsilon B_{IS}\}$,
$A_{AS} = \{s | \exists\ AS.\ (s,AS) \varepsilon B_{AS}\}$,
$A_S = \{s | \exists\ s_2,\ldots,s_n.\ (s,s_2,\ldots,s_n) \varepsilon B_S\}$.

(1) P ist statement-minimal bzgl. B_{IS} <=>
P realisiert B_{IS} statement-getreu $\wedge$ ($\forall$ P'εPROGRAMS.
P und P' statement-äquivalent bzgl. A_{IS} => $L_S(P) \leq L_S(P')$).

(2) P ist zuweisungs-minimal bzgl. B_{AS} <=>
P realisiert B_{AS} zuweisungs-getreu $\wedge$ ($\forall$ P'εPROGRAMS.
P und P' zuweisungs-äquivalent bzgl. A_{AS} => $L_A(P) \leq L_A(P')$).

(3) P ist statement-minimal bzgl. B_S <=>
P realisiert B_S statement-speicher-getreu $\wedge$ ($\forall$ P'εPROGRAMS.
P und P' statement-speicher-äquivalent bzgl.
A_S => $L_S(P) \leq L_S(P')$).

(4) P ist zuweisungs-minimal bzgl. B_S <=>
P realisiert B_S zuweisungs-speicher-getreu $\wedge$ ($\forall$ P'εPROGRAMS.
P und P' zuweisungs-speicher-äquivalent bzgl.
A_S => $L_A(P) \leq L_A(P')$).

Für einen Programmsynthesealgorithmus sind gemäß Def. 2.12 vier verschiedene Problemstellungen zu unterscheiden:

1. Gegeben ist eine Menge M von Statementbeispielsfolgen. Gesucht ist ein Programm P mit minimaler Anzahl von Instruktionsknoten, sodaß die durch M dargestellten Berechnungen auch Berechnungen von P sind. Dieser Fall wird durch die sogenannte Problemspezifizierungsmarke s gekennzeichnet.

2. Gegeben ist eine Menge M von Zuweisungsbeispielsfolgen. Gesucht ist ein Programm P mit minimaler Anzahl von Zuweisungsknoten, sodaß sich die durch M gegebenen Sequenzen von Zuweisungen auch bei den entsprechenden Berechnungen von P ergeben. Dieser Fall wird durch die Problemspezifizierungsmarke a gekennzeichnet.

3. Gegeben ist eine Menge M von Speicherbelegungsfolgen. Gesucht ist ein Programm P mit minimaler Anzahl von Instruktionsknoten, sodaß alle Elemente von M statementgetreue Speicherbelegungsfolgen von P sind. Dieser Fall wird durch ss gekennzeichnet.

4. Gegeben ist eine Menge M von Speicherbelegungsfolgen. Gesucht ist ein Programm P mit minimaler Anzahl von Zuweisungsknoten, sodaß alle Elemente von M zuweisungsgetreue Speicherbelegungsfolgen von P sind. Fall 4 wird durch as gekennzeichnet.

Die vier angegebenen Fälle werden in der folgenden Definition formalisiert:

Definition 2.13

(1) (b,M) heißt Programmsyntheseproblem (PSP) <=>
b = s $\wedge$ M ist Menge von Statementbeispielsfolgen
oder:
b = a $\wedge$ M ist Menge von Zuweisungsbeispielsfolgen
oder:
b = ss $\wedge$ M ist Menge von Speicherbelegungsfolgen
oder:
b = as $\wedge$ M ist Menge von Speicherbelegungsfolgen.
$PSP_{I,S}$ sei die Menge aller PSP.

(2) Ein PSP(b,M) heißt PSP aus Instruktionsfolgen <=>
b = s $\vee$ b = a .
PSP_I sei die Menge aller PSP aus Instruktionsfolgen.

(3) Ein PSP(b,M) heißt PSP aus Speicherbelegungsfolgen <=>
b = ss $\vee$ b = as .
PSP_S sei die Menge aller PSP aus Speicherbelegungsfolgen.

(4) (b,M) sei ein PSP. b heißt Problemspezifizierungsmarke, M heißt Beispielsmenge des PSP.

(5) Ein PSP (b,M) heißt endliches PSP genau dann, wenn $|M| < \infty$. Ist X eine Menge von PSP, so bezeichnet X^{fin} die Teilmenge aller endlichen PSP von X.

Um auch bei den in Def. 2.8, 2.10 und 2.12 eingeführten Äquivalenz-, Realisierungs- und Minimalitätsbegriffen Gebrauch von der Problemspezifizierungsmarkierung $b \in \{s,a,ss,as\}$ machen zu können, wird folgendes vereinbart:

In den Def. 2.8 und 2.10 sowie im folgenden entspreche das Präfix

- "statement" dem Präfix "s"
- "zuweisungs" dem Präfix "a"
- "statement-speicher" dem Präfix "ss"
- "zuweisungs-speicher" dem Präfix "as" .

Darüberhinaus sei für $b \in \{s,a,ss,as\}$ $L_b(P)$ mit $P \in PROGRAMS$ definiert durch:

$$L_s(P) := L_{ss}(P) :=_{def} L_S(P)$$

$$L_a(P) := L_{as}(P) :=_{def} L_A(P)$$

Diese Vereinbarungen ermöglichen eine einheitliche Sprechweise bzgl. der gesamten Menge der Programmsyntheseprobleme und haben Gültigkeit für alle folgenden Kapitel.

Definition 2.14

Sei $Q \varepsilon PSP_{I,S}$, $Q = (b,M)$, $P \varepsilon$ PROGRAMS, $n \varepsilon \mathbb{N}$.

(1) P realisiert Q <=> P realisiert M b-getreu.

(2) P n-realisiert Q <=> P realisiert $Q \wedge L_b(P) = n$.

(3) P ist minimal bzgl. Q <=> P realisiert $Q \wedge (\forall P' \varepsilon$PROGRAMS. P' realisiert $Q \Rightarrow L_b(P) \leq L_b(P'))$.

(4) Q ist realisierbar <=> $\exists$ $P \varepsilon$PROGRAMS. P realisiert Q.

(5) Q ist n-realisierbar <=> $\exists$ $P \varepsilon$PROGRAMS. P n-realisiert Q.

(6) Q ist minimal n-realisierbar <=> Q ist n-realisierbar $\wedge$ $\neg$(Q ist (n-1)-realisierbar).

Beispiel 2.3

Mit den Vereinbarungen von Beispiel 2.1 werden vier PSP Q_1, Q_2, Q_3, Q_4 angegeben, die auch in den weiteren Beispielen dieser Arbeit benützt werden.

1. $Q_1 = (\underline{s}, \{M_1, M_2, M_3, M_4\})$, wobei:

M_1 = ((3,1,3,4,4),(START,CL(1),ZR(2),A(1,3),S1(2),ZR(2), A(1,5),STOP))

M_2 = ((0,2,4,3,4),(START,CL(1),ZR(2),A(1,3),S1(2),ZR(2), A(1,3),S1(2),ZR(2),A(1,4),STOP))

M_3 = ((2,0,2,3,0),(START,CL(1),ZR(2),A(1,4),STOP))

M_4 = ((1,1,2,0,1),(START,CL(1),ZR(2),A(1,3),S1(2),ZR(2), A(1,5),STOP))

2. $Q_2 = (\underline{a}, \{M_1, M_2, M_3, M_4\})$, wobei:

M_1 = ((0,1,4,2,3),(START,CL(1),A(1,3),S1(2),A(1,5),STOP))

M_2 = ((2,2,5,2,2),(START,CL(1),A(1,3),S1(2),A(1,3),S1(2), A(1,4),STOP))

M_3 = ((3,0,4,3,1),(START,CL(1),A(1,4),STOP))

M_4 = ((3,2,2,5,4),(START,CL(1),A(1,3),S1(2),A(1,3),S1(2), A(1,4),STOP))

3. $Q_3 = (\underline{ss}, \{M_1, M_2, M_3, M_4\})$ mit:

M_1 = ((1,2,4,1,1),(0,2,4,1,1),(0,2,4,1,1),(4,2,4,1,1), (4,1,4,1,1),(4,1,4,1,1),(8,1,4,1,1),(8,0,4,1,1), (8,0,4,1,1),(9,0,4,1,1))

M_2 = ((0,1,3,1,3),(0,1,3,1,3),(0,1,3,1,3),(3,1,3,1,3), (3,0,3,1,3),(3,0,3,1,3),(6,0,3,1,3))

M_3 = ((1,1,0,2,1),(0,1,0,2,1),(0,1,0,2,1),(0,1,0,2,1), (0,0,0,2,1),(0,0,0,2,1),(1,0,0,2,1))

M_4 = ((3,2,1,2,3),(0,2,1,2,3),(0,2,1,2,3),(1,2,1,2,3), (1,1,1,2,3),(1,1,1,2,3),(2,1,1,2,3),(2,0,1,2,3), (2,0,1,2,3),(4,0,1,2,3))

4. $Q_4 = (\underline{as}, \{M_1, M_2, M_3, M_4, M_5\})$ mit:

M_1 = ((1,2,4,1,1),(0,2,4,1,1),(4,2,4,1,1),(4,1,4,1,1), (8,1,4,1,1),(8,0,4,1,1),(9,0,4,1,1))

M_2 = ((0,1,3,1,2),(0,1,3,1,2),(3,1,3,1,2),(3,0,3,1,2), (5,0,3,1,2))

M_3 = ((2,0,3,2,4),(0,0,3,2,4),(2,0,3,2,4))

M_4 = ((0,3,1,2,3),(0,3,1,2,3),(1,3,1,2,3),(1,2,1,2,3),
(2,2,1,2,3),(2,1,1,2,3),(3,1,1,2,3),(3,0,1,2,3),
(6,0,1,2,3))

M_5 = ((2,2,1,4,3),(0,2,1,4,3),(1,2,1,4,3),(1,1,1,4,3),
(2,1,1,4,3),(2,0,1,4,3),(6,0,1,4,3))

2.3 Programmsynthesealgorithmen

Aufgabe eines Programmsynthesealgorithmus (PSA) ist es, aus einer vorgegebenen endlichen Menge von Beispielen ein Programm zu konstruieren, das alle diese Beispiele erzeugen kann, wobei die vier angegebenen Arten von Beispielen zu unterscheiden sind.

Definition 2.15

(1) Ein Programmsynthesealgorithmus (PSA) ist eine Abbildung PA

$$PA : PSP_{I,S}^{fin} \longrightarrow PROGRAMS.$$

(2) Ein PSA aus Instruktionsfolgen ist eine Abbildung

$$PA : PSP_{I}^{fin} \longrightarrow PROGRAMS.$$

(3) Ein PSA aus Speicherbelegungsfolgen ist eine Abbildung

$$PA : PSP_{S}^{fin} \longrightarrow PROGRAMS.$$

Ob ein PSA PA 'richtig' arbeitet und wie seine Lösungen charakterisiert sind, kann von verschiedenen Gesichtspunkten aus beurteilt werden. Da die als Eingabe vorliegenden PSP unabhängig von der Menge der Programme definiert wurden, wird hiervon ausgehend in Def. 2.16 festgelegt, wann ein PSA fehlerfreie bzw. minimale Beispielsrealisierungen erzeugt.

Definition 2.16

Sei PA ein PSA, $B \subseteq PSP_{I,S}^{fin}$.

(1) PA erzeugt aus B fehlerfreie Beispielsrealisierungen <=>
($\forall Q \in B$. (PA(Q) realisiert Q <=> Q ist realisierbar)
$\wedge$ (PA(Q) = $\emptyset$ <=> $\neg$(Q ist realisierbar)))

(2) PA erzeugt aus B minimale Beispielsrealisierungen <=>
($\forall Q \in B$. Q ist realisierbar => PA(Q) ist minimal bzgl.Q)

Betrachtet man nur die realisierbaren PSP, so kann man sagen, daß jedes (realisierbare) PSP von einem Programm P erzeugt wird. Nimmt man z.B. eine Menge CP von Berechnungsfolgen von P und bildet daraus die entsprechenden zuweisungsgetreuen Speicherbelegungsfolgen M, so erhält man ein von P erzeugtes PSP (as,M). I.a. besitzt ein Programm P eine unendliche Menge verschiedener Berechnungsfolgen; da STORES und damit auch die Menge der möglichen Eingabewerte für P jedoch abzählbar und P nach Def. 2.4 deterministisch ist, kann P höchstens ab-

zählbar viele unterschiedliche Berechnungsfolgen haben, jedoch mehr als abzählbar viele PSP erzeugen.

Insbesondere werden dies endliche und i.a. auch unendliche PSP sein. Alle durch ein Programm P erzeugten endlichen und unendlichen PSP sind natürlich realisierbar, ein PSA kann sinnvollerweise aber nur endliche PSP bearbeiten.

Ein PSA wird als korrekt (bzgl. der Menge aller Programme) bezeichnet, wenn er aus jedem endlichen, von einem beliebigen Programm P erzeugten PSP Q ein Programm P' konstruiert, das Q realisiert.

Weiterhin ergibt sich die Frage, ob ein PSA PA zu jedem Programm P aus einem von P erzeugten endlichen PSP Q ein Programm P' synthetisieren kann, sodaß alle b-getreuen Berechnungsfolgen von P auch b-getreue Berechnungsfolgen von P' sind. Wenn für jede Aufzählung E der Berechnungsfolgen von P ein endliches Präfix von E genügt, sodaß PA aus dem diesem Präfix entsprechenden PSP ein Programm P' mit der gewünschten Eigenschaft konstruiert, so ist PA vollständig bzgl. der Menge aller Programme. (In diesem Fall kann man davon sprechen, daß PA jedes Programm P aus einer endlichen Menge von Beispielen synthetisieren kann). Ist das so erhaltene Programm P' darüberhinaus das 'kleinste' in der Klasse aller Programme P", die die von P' geforderte Eigenschaft erfüllen, so ist PA lösungsminimal.

Die drei folgenden Definitionen präzisieren diese Überlegungen.

Definition 2.17

Sei $P \in$ PROGRAMS.

(1) $COMP(P) = \{c(P,s) \mid \exists s \in STORES.\ c(P,s)$ ist vollständige, terminale Berechnungsfolge$\}$
ist die Menge der Berechnungsfolgen von P.

(2) $DEF(P) = \{s \mid s \in STORES \wedge c(P,s) \in COMP(P)\}$ ist der Definitionsbereich von P (das ist gerade der Bereich, über dem P terminiert).

Jede Teilmenge der Berechnungsfolgen eines Programms P kann vier verschiedene Programmsyntheseprobleme erzeugen.

Definition 2.18

Für eine Menge M bezeichne POT(M) die Potenzmenge von M.

(1) GPSP ist eine Abbildung
$GPSP : \{s,a,ss,as\} \times POT(COMP(P)) \longrightarrow PSP_{I,S}$
und ist definiert durch:
$GPSP = \lambda b.\ \lambda CP.\ b = s \longrightarrow (s, \{stmts(P,s) \mid c(P,s) \in CP\})$
$b = a \longrightarrow (a, \{assign(P,s) \mid c(P,s) \in CP\})$
$b = ss \longrightarrow (ss, \{ss\text{-}stores(P,s) \mid c(P,s) \in CP\})$
$b = as \longrightarrow (as, \{as\text{-}stores(P,s) \mid c(P,s) \in CP\})$

(2) Sei $P \in$ PROGRAMS, $CP \subseteq COMP(P)$, $b \in \{s,a,ss,as\}$.
GPSP(b,CP) ist das von CP b-getreu erzeugte Programmsyntheseproblem.

Beispiel 2.4

Die im vorigen Beispiel angegebenen PSP Q_1, Q_2, Q_3, Q_4 werden vom Programm P (Beispiel 2.2) erzeugt.

Hiermit sind die Voraussetzungen dafür gegeben, Korrektheit, Vollständigkeit und Lösungsminimalität von PSA zu formalisieren.

Definition 2.19

PA sei ein PSA. Für eine aufzählbare Menge X bezeichne ENUMERATE(X) die Menge aller Aufzählungen von X.

(1) PA ist korrekt (bzgl. PROGRAMS) <=>

$\forall$ P$\in$PROGRAMS. $\forall$ CP $\subseteq$ COMP(P). $\forall$ b$\in\{$s,a,ss,as$\}$

(|CP| < ∞ => PA(GPSP(b,CP)) realisiert GPSP(b,CP))

(2) PA ist vollständig (bzgl. PROGRAMS) <=>

$\forall$ P$\in$PROGRAMS. $\forall$ $(C_1, C_2, \ldots) \in$ ENUMERATE(COMP(P)).

$\forall$ b$\in\{$s,a,ss,as$\}$. $\exists$ k$\in\mathbb{N}$.

(PA(GPSP(b,$\{C_1, \ldots, C_k\}$)) realisiert GPSP(b,COMP(P)))

(3) PA ist lösungsminimal (bzgl. PROGRAMS) <=>

$\forall$ P,P'$\in$PROGRAMS. $\forall$ $(C_1, C_2, \ldots) \in$ ENUMERATE(COMP(P)).

$\forall$ b$\in\{$s,a,ss,as$\}$. $\forall$ k$\in\mathbb{N}$.

(PA(GPSP(b,$\{C_1, \ldots, C_k\}$)) realisiert GPSP(b,COMP(P))

$\wedge$ P' realisiert GPSP(b,COMP(P))

=> L_b(PA(GPSP(b,$\{C_1, \ldots, C_k\}$))) $\subseteq$ L_b(P'))

Betrachtet man nur eine der vier Arten von PSP, so ist für b$\in\{$s,a,ss,as$\}$ b-korrekt wie "korrekt" zu definieren, wobei die Quantifizierung über b entfällt und für b der entsprechende Wert eingesetzt wird (analog für b-vollständig und b-lösungsminimal).

Bei der Untersuchung von Vollständigkeit und Lösungsminimalität von PSA und bei dem Vergleich von Programmen miteinander erweist sich die folgende Definition als nützlich.

Definition 2.20:

Seien P,P'$\in$PROGRAMS, b$\in\{$s,a,ss,as$\}$.

(1) P' ist b-getreue Extension von P <=>

P' realisiert GPSP(b,COMP(P)).

(2) P' und P sind total b-äquivalent <=>

P' ist b-getreue Extension von P

$\wedge$ P ist b-getreue Extension von P'.

Mit den in Def. 2.8 eingeführten Äquivalenzbegriffen ergeben sich folgende Beziehungen:

(i) P' ist b-getreue Extension von P <=>

P' und P sind b-äquivalent bzgl. DEF(P).

(ii) P' und P sind total b-äquivalent <=>

P' und P sind b-äquivalent bzgl. DEF(P) $\wedge$ DEF(P)=DEF(P').

P' ist also b-getreue Extension von P genau dann, wenn jede terminierende Berechnungsfolge von P auch terminierende Berechnungsfolge von P' ist.

2.4 Zielsetzungen

Es soll ein PSA PA entwickelt werden, der

- den erforderlichen Suchraum von vorneherein möglichst stark einschränkt
- diesen eingeschränkten Suchraum nicht "ziellos" durchsucht, sonden die durch M gegebenen Informationen konstruktiv benutzt.
- durch fortgesetzte Hypothesenbildung sich immer mehr dem Zielprogramm annähert
- sinnvolle Heuristiken für die Reihenfolge der Auswahl der Hypothesen benützt.
- falsche Hypothesen möglichst frühzeitig als solche erkennt
- alle vier Arten von PSP in uniformer Weise bearbeitet

und für den durch formalen Korrektheitsbeweis gezeigt wird, daß er

- aus der Menge aller endlichen PSP fehlerfreie und minimale Beispielsrealisierungen erzeugt
- bzgl. der Menge aller Programme korrekt, vollständig und lösungsminimal ist.

Neben Abschätzungen der Größe des Suchraumes von PA in Abhängigkeit vom Eingabe-PSP Q und dessen Parametern sollen für $PSP^{fin}_{I,S}$ und verschiedene Unterklassen davon Komplexitätsberechnungen vorgenommen werden.

Kap. 3. IN PA VERWANDTE PROBLEMDARSTELLUNGSARTEN

3.1. Standarddarstellungen

Standarddarstellungen werden für PSP (3.1.1) und deren Realisierungen (3.1.2) definiert. In 3.1.3 werden die bestehenden Beziehungen zwischen beiden aufgezeigt.

3.1.1 Standarddarstellung eines PSP

Q = (b,M) sei ein PSP. In Abhängigkeit von b stehen mit M zunächst völlig verschiedenartige Informationen zur Verfügung, aus denen ein Programm synthetisiert werden soll. Betrachtet man Q als ein von einem Programm P erzeugtes PSP, so geben die Folgen aus M Informationen über die Berechnungsfolgen von P bei gewissen Eingabewerten. Nach Definition von Q sind diese Eingabewerte, aus denen die Elemente von M gewonnen wurden, gegeben.

Bei PSP aus Instruktionsfolgen sind keine weiteren Angaben zu den jeweiligen Zwischen- und Endspeicherbelegungen gemacht; bei PSP aus Speicherbelegungsfolgen sind gerade diese vorhanden, aber keinerlei Angaben darüber, welche Instruktionen ausgeführt werden. In beiden Fällen fehlen die Transitionsmarkierungen aus {true,false,nil}, die in der vollständigen Berechnungsfolgen erscheinen müßten. Ist $b \varepsilon \{a,as\}$, so können zwischen zwei in M hintereinanderfolgenden Instruktionen bzw. Speicherbelegungen in der

zugehörigen vollständigen Berechnungsfolge keine, eine oder beliebig viele Testinstruktionen stehen. In allen Fällen fehlen im Vergleich zu den vollständigen Berechnungsfolgen mit den l_i die Angaben über die LABEL der durchlaufenen Programmknoten.

Wären alle diese Informationen gegeben, d.h. bestünde M aus einer Menge von Berechnungsfolgen von P, so wäre es eine triviale Aufgabe, ein Programm P' anzugeben, das alle diese Berechnungsfolgen ebenfalls erzeugen kann, indem man als Programmknoten von P' gerade die (l_i,I_i) nimmt, die in M enthalten sind, und die Transitionen zwischen den Knoten gemäß den in M gegebenen wählt. Allerdings ist dann P' nicht notwendig minimale Beispielsrealisierung von M.

Die erste Phase des PSA besteht daher darin, Q in ein sog. standardisiertes PSP zu überführen, in dem bereits mehr von Q ableitbare Informationen über die entsprechenden vollständigen Berechnungsfolgen enthalten sind. Die zu Q gehörige Standarddarstellung ist ein 4-Tupel (b,S, MI,MT), wobei S die Folgen der berechneten Zwischenwerte, MI die Menge der jeweils möglicherweise ausgeführten Instruktionen und MT die zugehörigen Transitionen wiedergibt. Im Falle der Transitionen macht man sich dabei folgende Überlegungen zunutze:
Ist bε{s,ss}, so kann zwischen zwei Instruktionen bzw. zwei Speicherbelegungen nur genau ein Übergang t von einem Instruktionsknoten zum anderen erfolgt sein, wobei

tε{true,false,nil}. Ist b=s, so ist t eindeutig festgelegt und in MT wird an der entsprechenden Stelle t gespeichert. Für b=ss ist t abhängig von der zuvor ausgeführten Instruktion, für die es i.a. mehrere Alternativer gibt. In MT werden in diesem Fall Paare (I,t) aus möglich Instruktion I und entsprechendem Übergang t abgelegt.

Ist bε{a,as}, so kann der oben beschriebene Fall von keiner, einer oder mehreren Testinstruktionen vorliegen. Da für solche b zunächst nur eine Minimisierung der Zuweisungsknoten gefordert wurde, genügt vorerst die Annahme, daß jede Testinstruktion genau einmal angewendet wurde, bevor eine neue Zuweisungsinstruktion ausgeführt wurde. Eine solche Folge von unterschiedlichen Tests läßt sich (bis auf deren Reihenfolge) eindeutig charakterisieren z.B. durch die Angabe der Tests, die true liefern. Dies wird in MT an der entsprechenden Stelle gespeichert.

Die folgende Definition erlaubt es, in allen Fällen von b-Transitionen zu sprechen.

Definition 3.1

(1) Für bε{s,a,ss,as} heißt $TRANS_b$ Menge der b-Transitionen <=>

(i) b = s => $TRANS_b$ = {true,false,nil} = TFN

(ii) bε{a,as} => $TRANS_b$ = POT(TESTS)

(iii) b = ss => $TRANS_b$ = POT(INSTRUCTIONS×TFN)

2) Für $b \in \{\underline{s},\underline{a},\underline{ss},\underline{as}\}$ heißt $TRANS'_b$ Menge der b'-Transitionen <=>

(i) $b \in \{\underline{s},\underline{ss}\}$ => $TRANS'_b$ = TFN

(ii) $b \in \{\underline{a},\underline{as}\}$ => $TRANS'_b$ = POT(TESTS)

Bei dem in Kapitel 2 eingeführten Berechnungsmodell wurden die Mengen der Zuweisungsoperationen bzw. die der Testoperationen nicht näher spezifiziert. Die Berechnungsfunktion Γ besteht i.w. aus einer Anwendung der jeweils vorliegenden Instruktion auf die gegebene Speicherbelegung. In Def. 3.2 werden zwei Projektionen von Γ und drei sich daraus ergebende Umkehrfunktionen definiert, mit deren Hilfe das eingangs erwähnte standardisierte PSP angegeben werden kann.

Definition 3.2.

Γ sei die Berechnungsfunktion des in Def. 2.5 beschriebenen Berechnungsmodells.

(1) γ: INSTRUCTIONS × STORES → TFN

$\gamma = proj_{\downarrow TFN}(\Gamma)$

(2) δ: INSTRUCTIONS × STORES → STORES

$\delta = proj_{\downarrow STORES}(\Gamma)$

(3) $\bar{\gamma}$: STORES × {true,false} → POT(TESTS)

$\bar{\gamma} = \lambda s.\lambda t.\ \{I \mid I \in TESTS \wedge \gamma(I,s) = t\}$

(4) $\bar{\delta}_A$: $STORES^2$ ⟶ POT(ASSIGNMENTS)

$\bar{\delta}_A = \lambda s.\lambda s'.\ \{I \mid I \in ASSIGNMENTS \wedge \delta(I,s) = s'\}$

(5) $\bar{\delta}_I$: $STORES^2$ ⟶ POT(INSTRUCTIONS)

$\bar{\delta}_I = \lambda s.\lambda s'.\ \{I \mid I \in INSTRUCTIONS \wedge \delta(I,s) = s'\}$

Definition 3.3.

Q = (b,M) sei ein PSP. Für M gelte in Abhängigkeit von b:

$M = (M_i)_{1 \le i \le m}$ mit:

$b \in \{\underline{s},\underline{a}\} \Rightarrow \forall i \in \{1,\ldots,m\}.\ M_i = (st_i,(M_{i,0},\ldots,M_{i,n_i})$

$b \in \{\underline{ss},\underline{as}\} \Rightarrow \forall i \in \{1,\ldots,m\}.\ M_i = (M_{i,1},\ldots,M_{i,n_i})$

(1) $dim(Q) = (m,(n_1,\ldots,n_m))$ heißt Dimension von Q.

(2) (b,S,MI,MT) heißt standardisiertes PSP von Q <=>

(i) $S = (s_{i,j})_{\substack{1 \le i \le m \\ 1 \le j \le n_i}}$ mit:

$b \in \{\underline{s},\underline{a}\}$ => $\forall i \in \{1,\ldots,m\}.\ (s_{i,1} = st_i) \wedge \forall j \in \{2,\ldots,n_i\}.\ s_{i,j} = \delta(M_{i,j},s_{i,j-1})$

$b \in \{\underline{ss},\underline{as}\}$ => S = M

S heißt Speicherbelegungsmatrix von Q.

(ii) $MI = (MI_{i,j})_{\substack{1 \le i \le m \\ 0 \le j \le n_i}}$ mit:

$\forall i \in \{1,\ldots,m\}.\ MI_{i,0} = \{\underline{START}\} \wedge \forall j \in \{1,\ldots,n_i-1\}.$

$$MI_{i,j} = \begin{cases} \{M_{i,j}\} & <=> b \varepsilon \{\underline{s},\underline{a}\} \\ \overline{\delta}_I\,(s_{i,j},s_{i,j+1}) & <=> b = \underline{ss} \\ \overline{\delta}_A\,(s_{i,j},s_{i,j+1}) & <=> b = \underline{as} \end{cases}$$

$\wedge\ MI_{i,n_i} = \{\underline{STOP}\}$

MI heißt Matrix der Instruktionen von Q.

(iii) $MT = (MT_{i,j})_{\substack{1 \le i \le m \\ 0 \le j < n_i}}$ mit:

$\forall_{i \varepsilon \{1,\ldots,m\}}.\ \forall_{j \varepsilon \{0,\ldots,n_i-1\}}.$

$$MT_{i,j} = \begin{cases} \underline{nil} & <=> b=\underline{s} \wedge j=0 \\ \gamma(M_{i,j},s_{i,j}) & <=> b=\underline{s} \wedge j \neq 0 \\ \{(START,\underline{nil})\} & <=> b=\underline{ss} \wedge j=0 \\ \{(I,\gamma(I,s_{i,j}))\,|\,I \varepsilon MI_{i,j}\} & <=> b=\underline{ss} \wedge j \neq 0 \\ \overline{\gamma}(s_{i,j+1},\underline{true}) & <=> b \varepsilon \{\underline{a},\underline{as}\} \end{cases}$$

MT heißt Matrix der Transitionen von Q.

Notation: Für $I \varepsilon MI_{i,j}$ sei:

$$MT^I_{i,j} = \begin{cases} MT_{i,j} & <=> b \varepsilon \{\underline{s},\underline{a},\underline{as}\} \\ m & <=> b=\underline{ss} \wedge (I,m) \varepsilon MT_{i,j} \end{cases}$$

(Bemerkung: Für alle $I \varepsilon MI_{i,j}$ ist $M^I_{i,j}$ eindeutig bestimmt!)

(3) Für ein PSP Q bezeichnet Q_{ST} das standardisierte PSP von Q.

Beispiel 3.1

Für die Beispiels-PSP Q_1, Q_2, Q_3, Q_4 werden die Standarddarstellungen angegeben:

1. Für Q_1 : (s,S,MI,MT) mit:

$$\begin{bmatrix}
3 & 0 & 0 & 3 & 3 & 3 & 7 & & & \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & & & \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & & & \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & & & \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & & & \\
0 & 0 & 0 & 4 & 4 & 4 & 8 & 8 & 8 & 11 \\
2 & 2 & 2 & 2 & 1 & 1 & 1 & 0 & 0 & 0 \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\
3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\
4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\
2 & 0 & 0 & 3 & & & & & & \\
0 & 0 & 0 & 0 & & & & & & \\
2 & 2 & 2 & 2 & & & & & & \\
3 & 3 & 3 & 3 & & & & & & \\
0 & 0 & 0 & 0 & & & & & & \\
1 & 0 & 0 & 2 & 2 & 2 & 3 & & & \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & & & \\
2 & 2 & 2 & 2 & 2 & 2 & 2 & & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & & & \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & & &
\end{bmatrix} = S$$

$$MI = \begin{bmatrix}
\{\underline{START}\}\ \{CL(1)\}\ \{ZR(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{ZR(2)\}\ \{A(1,5)\}\ \{\underline{STOP}\} \\
\{\underline{START}\}\ \{CL(1)\}\ \{ZR(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{ZR(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{ZR(2)\}\ \{A(1,4)\}\ \{\underline{STOP}\} \\
\{\underline{START}\}\ \{CL(1)\}\ \{ZR(2)\}\ \{A(1,4)\}\ \{\underline{STOP}\} \\
\{\underline{START}\}\ \{CL(1)\}\ \{ZR(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{ZR(2)\}\ \{A(1,5)\}\ \{\underline{STOP}\}
\end{bmatrix}$$

$$\begin{bmatrix}
\underline{nil} & \underline{nil} & \underline{false} & \underline{nil} & \underline{nil} & \underline{true} & \underline{nil} & & & \\
\underline{nil} & \underline{nil} & \underline{false} & \underline{nil} & \underline{nil} & \underline{false} & \underline{nil} & \underline{nil} & \underline{true} & \underline{nil} \\
\underline{nil} & \underline{nil} & \underline{true} & \underline{nil} & & & & & & \\
\underline{nil} & \underline{nil} & \underline{false} & \underline{nil} & \underline{nil} & \underline{true} & \underline{nil} & & &
\end{bmatrix} = MT$$

2. Für Q_2 : (a,S,MI,MT) mit:

$$S = \begin{bmatrix}
0 & 0 & 4 & 4 & 7 & & \\
1 & 1 & 1 & 0 & 0 & & \\
4 & 4 & 4 & 4 & 4 & & \\
2 & 2 & 2 & 2 & 2 & & \\
3 & 3 & 3 & 3 & 3 & & \\
 & & & & & & \\
2 & 0 & 5 & 5 & 10 & 10 & 12 \\
2 & 2 & 2 & 1 & 1 & 0 & 0 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 \\
 & & & & & & \\
3 & 0 & 3 & & & & \\
0 & 0 & 0 & & & & \\
4 & 4 & 4 & & & & \\
3 & 3 & 3 & & & & \\
1 & 1 & 1 & & & & \\
 & & & & & & \\
3 & 0 & 2 & 2 & 4 & 4 & 9 \\
2 & 2 & 2 & 1 & 1 & 0 & 0 \\
2 & 2 & 2 & 2 & 2 & 2 & 2 \\
5 & 5 & 5 & 5 & 5 & 5 & 5 \\
4 & 4 & 4 & 4 & 4 & 4 & 4
\end{bmatrix}$$

$$MI = \begin{bmatrix}
\{\underline{START}\}\ \{CL(1)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{A(1,5)\}\ \{\underline{STOP}\} & & & & & & \\
\{\underline{START}\}\ \{CL(1)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{A(1,4)\}\ \{\underline{STOP}\} & & & & & & \\
\{\underline{START}\}\ \{CL(1)\}\ \{A(1,4)\}\ \{\underline{STOP}\} & & & & & & \\
\{\underline{START}\}\ \{CL(1)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{A(1,3)\}\ \{S1(2)\}\ \{A(1,4)\}\ \{\underline{STOP}\} & & & & & &
\end{bmatrix}$$

$$MT = \begin{bmatrix}
\{ZR(1)\} & \{ZR(1)\} & \emptyset & \{ZR(2)\} & \{ZR(2)\} & & \\
\emptyset & \{ZR(1)\} & \emptyset & \emptyset & \emptyset & \{ZR(2)\} & \{ZR(2)\} \\
\{ZR(2)\} & \{ZR(1),ZR(2)\} & \{ZR(2)\} & & & & \\
\emptyset & \{ZR(1)\} & \emptyset & \emptyset & \emptyset & \{ZR(2)\} & \{ZR(2)\}
\end{bmatrix}$$

3. Für Q_3 : (ss,S,MI,MT) mit:

Abkürzend stehe (A(i,1),nil) für
(A(1,1),nil),...,(A(1,5),nil);
entsprechendes gelte für (ZR(i),false). Weiterhin sei:

A1: = {(CL(1),nil),(A(i,1),nil),(ZR(1),true),(ZR(2),false),...
...,(ZR(5),false)}

A2: = {(CL(2),nil),(A(i,2),nil),(ZR(1),false),(ZR(2),true),
(ZR(3),false),(ZR(4),false),(ZR(5),false)}

A3: = {(CL(1),nil),(CL(3),nil),(A(i,1),nil),(A(i,3),nil),
(ZR(1),true),(ZR(3),true),(ZR(4),false),(ZR(5),false)}

MT ist gemäß diesen Vereinbarungen in Abb. 3.1 wiedergegeben.

MI ergibt sich aus den ersten Elementen der Paare in MT (z.B. ist $MI_{2,3}$ = {A(1,3),A(1,5)}), wobei jede Zeile in MI noch zusätzlich als letztes Element {STOP} besitzt.

S gibt die vorgegebenen Speicherbelegungsfolgen wieder (Beispiel 2.3-3).

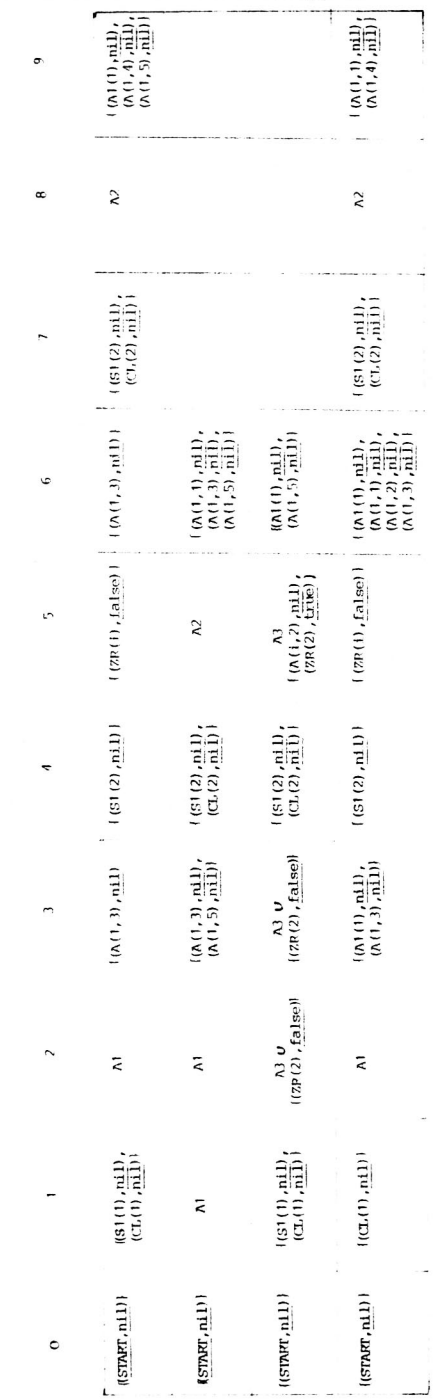


Abb. 3.1 Matrix MT zu Beispiel 3.1-3

4. Für Q_4 : (as,S,MI,MT) mit:

S gibt die vorgegebenen Speicherbelegungsfolgen wieder (Beispiel 2.3-4).

$$MI = \begin{bmatrix}
\{START\} & \{S1(1), CL(1)\} & \{A(1,3)\} & \{S1(2)\} & \{A(1,3)\} & \{S1(2), CL(2)\} & \{A1(1), A(1,4) A(1,5)\} & \{STOP\} \\
\{START\} & \{CL(1), A(i,1)\} & \{A(1,3)\} & \{S1(2), CL(2)\} & \{A(1,5)\} & \{STOP\} & & \\
\{START\} & \{CL(1)\} & \{A(1,4)\} & \{STOP\} & & & & \\
\{START\} & \{CL(1), A(i,1)\} & \{A1(1), A(1,3)\} & \{S1(2)\} & \{A1(1), A(1,1) A(1,3)\} & \{S1(2)\} & \{A1(1), A(1,3)\} & \{S1(2), CL(2)\} \quad \{A(1,1), A(1,5)\} \quad \{STOP\} \\
\{START\} & \{CL(1)\} & \{A1(1), A(1,3)\} & \{S1(2)\} & \{A1(1), A(1,1) A(1,2) A(1,3)\} & \{S1(2), CL(2)\} & \{A(1,4)\} & \{STOP\}
\end{bmatrix}$$

$$MT = \begin{bmatrix}
\emptyset & \{ZR(1)\} & \emptyset & \emptyset & \emptyset & \{ZR(2)\} & \{ZR(2)\} & \\
\{ZR(1)\} & \{ZR(1)\} & \emptyset & \{ZR(2)\} & \{ZR(2)\} & & & \\
\{ZR(2)\} & \{ZR(1),ZR(2)\} & \{ZR(2)\} & & & & & \\
\{ZR(1)\} & \{ZR(1)\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{ZR(2)\} \quad \{ZR(2)\} \\
\emptyset & \{ZR(1)\} & \emptyset & \emptyset & \emptyset & \{ZR(2)\} & \{ZR(2)\} &
\end{bmatrix}$$

Mit der Konstruktion der Standarddarstellung von Q aus einem gegebenen PSP Q liegt bereits eine größere Menge an verwertbarer Information über das zu konstruierende Programm vor; außerdem stehen diese Informationen nun in relativ uniformer Weise für alle vier Fälle $b \in \{\underline{s},\underline{a},\underline{ss},\underline{as}\}$ zur Verfügung. Wie die Standarddarstellung von Q in Relation zu einem Q realisierenden Programm P steht, wird in 3.3.3 gezeigt.

3.1.2 Standarddarstellung einer Realisierung

Analog zur Standarddarstellung eines PSP Q ist die Standardarstellung eines Programms P, das Q realisiert, ein 4-Tupel, das als Elemente 4 Matrizen enthält: Die Matrix der ausgeführten Instruktionen (PI), die Matrix der Markierungen dieser Instruktionen (PL) und die Matrizen der erzeugten Speicherbelegungen (PS) und der durchlaufenen Transitionsmarkierungen (PT). Auch hier sind diese Informationen in Abhängigkeit von der Problemspezifizierungsmarke b so angegeben, daß sich eine uniforme Darstellungsweise ergibt.

Definition 3.4.

$Q = (b,M)$ sei realisierbares PSP, $\dim(Q) = (m,(n_1,\ldots,n_m))$, $M = \{M_1,\ldots,M_m\},\{st_1,\ldots,st_m\}$ die M entsprechenden Anfangsspeicherbelegungen.
P sei ein Programm, das Q realisiert.

(1) Für $i \in \{1,\ldots,m\}$ ist die Standardbezeichnung der Berechnungsfolgen von P bei Realisierung von Q wie folgt:

(1.1) falls $b \in \{\underline{s},\underline{ss}\}$:

$$c(P,st_i) = ((l_{i,0},I_{i,0}),(t_{i,0},s_{i,1}),(l_{i,1},I_{i,1}),\ldots,(t_{i,n_i-1},s_{i,n_i}),(l_{i,n_i},I_{i,n_i}))$$

(1.2) falls $b \in \{\underline{a},\underline{as}\}$:

$$c(P,st_i) = ((l_{i,0},I_{i,0}),(t_{i,0},s_{i,1}))\bullet TS_{i,0}\bullet((l_{i,1},I_{i,1}),(t_{i,1},s_{i,2}))\bullet TS_{i,1}\bullet\ldots\bullet TS_{i,n_i-1}\bullet((l_{i,n_i},I_{i,n_i}))$$

wobei für $j \in \{1,\ldots,n_i-1\}$:

$$I_{i,j} \in \text{ASSIGNMENTS}$$

und für $j \in \{0,\ldots,n_i-1\}$:

$$TS_{i,j} = ((l^1_{i,j},I^1_{i,j}),(t^1_{i,j},s^1_{i,j}),\ldots,(l^{k(i,j)}_{i,j},I^{k(i,j)}_{i,j}),(t^{k(i,j)}_{i,j},s^{k(i,j)}_{i,j}))$$

sodaß gilt: $k(i,j) \geq 0 \wedge \forall k \in \{1,\ldots,k(i,j)\}.\ I^k_{i,j} \in \text{TESTS}$

(2) Mit den unter (1) angegebenen Bezeichnungen

ist RPQ(P,Q) = (PL,PI,PS,PT) definiert durch:

(i) $PL = (PL_{i,j})_{\substack{1\le i\le m\\ 0\le j\le n_i}} := (l_{i,j})_{\substack{1\le i\le m\\ 0\le j\le n_i}}$

(ii) $PI = (PI_{i,j})_{\substack{1\le i\le m\\ 0\le j\le n_i}} := (I_{i,j})_{\substack{1\le i\le m\\ 0\le j\le n_i}}$

(iii) $PS = (PS_{i,j})_{\substack{1\le i\le m\\ 1\le j\le n_i}} := (s_{i,j})_{\substack{1\le i\le m\\ 1\le j\le n_i}}$

(iv)
$$PT = (PT_{i,j})_{\substack{1\le i\le m\\ 0\le j<n_i}} := \begin{cases} (t_{i,j})_{\substack{1\le i\le m\\ 0\le j<n_i}} & <=> b\varepsilon\{\underline{s},\underline{ss}\} \\ (\bar{\gamma}(s_{i,j+1},\underline{true}))_{\substack{1\le i\le m\\ 0\le j<n_i}} & <=> b\varepsilon\{\underline{a},\underline{as}\} \end{cases}$$

(3) RPQ(P,Q) heißt Standardarstellung der Realisierung von Q durch P.

Beispiel 3.2

Programm P (Beispiel 2.2) realisiert PSP Q_1, Q_2, Q_3 und Q_4. Für Q_2 und Q_3 werden die Standarddarstellungen ihrer Realisierung durch P angegeben.

1. Für Q_2 : (PL,PI,PS,PT) mit:

$$PL = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & & \\ 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\ 1 & 1 & 1 & 1 & & & & \\ 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \end{bmatrix}$$

$$PI = \begin{bmatrix} \underline{START} & CL(1) & A(1,3) & S1(2) & A(1,5) & \underline{STOP} & & \\ \underline{START} & CL(1) & A(1,3) & S1(2) & A(1,3) & S1(2) & A(1,4) & \underline{STOP} \\ \underline{START} & CL(1) & A(1,4) & \underline{STOP} & & & & \\ \underline{START} & CL(1) & A(1,3) & S1(2) & A(1,3) & S1(2) & A(1,4) & \underline{STOP} \end{bmatrix}$$

PS ist identisch mit S, PT identisch mit MT aus Beispiel 3.1-2.

2. Für Q_3 : (PL,PI,PS,PT) mit:

$$PL = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & & & \\ 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 1 & 1 \end{bmatrix}$$

$$PI = \begin{bmatrix} \underline{\text{START}} & \text{CL(1)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,4)} & \underline{\text{STOP}} \\ \underline{\text{START}} & \text{CL(1)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,5)} & \underline{\text{STOP}} & & & \\ \underline{\text{START}} & \text{CL(1)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,5)} & \underline{\text{STOP}} & & & \\ \underline{\text{START}} & \text{CL(1)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,3)} & \text{S1(2)} & \text{ZR(2)} & \text{A(1,4)} & \underline{\text{STOP}} \end{bmatrix}$$

PS ist identisch mit S aus Beispiel 3.1-3.

$$PT = \begin{bmatrix} \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{true}} & \underline{\text{nil}} \\ \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{true}} & \underline{\text{nil}} & & & \\ \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{true}} & \underline{\text{nil}} & & & \\ \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{false}} & \underline{\text{nil}} & \underline{\text{nil}} & \underline{\text{true}} & \underline{\text{nil}} \end{bmatrix}$$

Betrachtet man die Markierungsmatrix PL und die Instruktionsmatrix PI einer solchen Standarddarstellung, so sind i.a. viele Elemente der Matrizen gleich. Ein Paar $(PL_{i,j},PI_{i,j})$ entspricht genau einem Programmknoten des Programms P. Die Paare $(PL_{i,0},PI_{i,0})$ entsprechen z.B. alle dem START-Knoten von P, sind also für alle in Frage kommenden i identisch. In Abhängigkeit von den Transitionen, die durch PT dargestellt sind, können nun hinreichende Bedingungen dafür aufgestellt werden, welche der Paare $(PL_{i,j},PI_{i,j})$ notwendigerweise denselben Programmknoten aus P bezeichnen; dies geschieht in den beiden folgenden Sätzen.

Satz 3.1

Sei $P \in PROGRAMS$; $st_x, st_y \in STORES$; $c(P,st_x)$ und $c(P,st_y)$ seien die Berechnungsfolgen von P mit den Standardbezeichnungen aus Def. 3.4 (1.1) (Fall $b \in \{\underline{s},\underline{ss}\}$).

Behauptung: $\forall n \in \{1,\ldots,\min(n_x,n_y)\}.$
$((\forall j \in \{0,\ldots,n-1\}.\ t_{x,j} = t_{y,j})$
$\Rightarrow (\forall j \in \{0,\ldots,n\}.\ l_{x,j} = l_{y,j} \wedge I_{x,j} = I_{y,j}))$

Beweis:

Sei $n \in \{1,\ldots,\min(n_x,n_y)\}$ und es gelte:
$\forall j \in \{0,\ldots,n-1\}.\ t_{x,j} = t_{y,j}$.

Induktion über j:

j=0 : $(l_{x,0},I_{x,0}) = (l_{y,0},I_{y,0}) = (I,\underline{\text{START}})$

j+1 : Es gilt: $((l_{x,j},I_{x,j}),t_{x,j},(l_{x,j+1},I_{x,j+1})) \in P$

und $((l_{y,j},I_{y,j}),t_{y,j},(l_{y,j+1},I_{y,j+1})) \in P$

Da $(l_{x,j},I_{x,j}) = (l_{y,j},I_{y,j})$ nach Induktionsannahme und $t_{x,j} = t_{y,j}$ nach Voraussetzung, folgt wegen der Eindeutigkeit von P:

$(l_{x,j+1},I_{x,j+1}) = (l_{y,j+1},I_{y,j+1})$.

Satz 3.2

Sei $P \in PROGRAMS$; $st_x, st_y \in STORES$; $c(P,st_x)$ und $c(P,st_y)$ seien die Berechnungsfolgen von P mit den Standardbezeichnungen aus Def. 3.4 (1.2) (Fall $b \in \{\underline{a},\underline{as}\}$).

Behauptung:

(1) $\forall n \in \{1,\ldots,\min(n_x,n_y)\}.$
$\forall j \in \{0,\ldots,n-1\}.$
$[(l_{x,j} = l_{y,j} \wedge I_{x,j} = I_{y,j} \wedge \bar{\gamma}(s_{y,j+1},\underline{true}) = \bar{\gamma}(s_{y,j+1},\underline{true})$
$\Rightarrow (k(x,j) = k(y,j)$
$\wedge \forall k \in \{1,\ldots,k(x,j)\}.$
$(l^k_{x,j} = l^k_{y,j} \wedge I^k_{x,j} = I^i_{y,j} \wedge t^k_{x,j} = t^k_{y,j})]$

(2) $\forall n \in \{1,\ldots,\min(n_x,n_y)\}.$
$[(\forall j \in \{0,\ldots,n-1\}.\ \bar{\gamma}(s_{x,j+1}) = \bar{\gamma}(s_{y,j+1}))$
$\Rightarrow (\forall j \in \{0,\ldots,n\}.\ l_{x,j} = l_{x,j}\ \ I_{x,j} = I_{y,j})]$

Beweis:

(1) Analog zum Beweis von Satz 3.1 durch Induktion über k, wobei beim Induktionsschritt jeweils zusätzlich $t^{k+1}_{x,j} = t^{k+1}_{y,j}$ aus $\bar{\gamma}(s_{x,j+1}) = \bar{\gamma}(s_{y,j+1})$ gefolgert wird.

(2) Ebenfalls analog zu Satz 3.1, wobei (1) jeweils den Induktionsschritt ermöglicht.

Die in diesen beiden Sätzen gezeigten Eigenschaften bilden die Grundlage der in Abschnitt 3.2 definierten Stufengraphdarstellungen.

3.1.3 Relationen zwischen Standarddarstellungen

Zwischen der Standdarddarstellung eines PSP Q und der einer Realisierung von Q durch ein Programm P besteht die durch den folgenden Satz ausgedrückte Beziehung.

Satz 3.3

Q sei PSP, $dim(Q) = (m,(n_1,\ldots,n_m))$, $Q_{st} = (b,S,MI,MT)$.

Behauptung: $\forall P \in PROGRAMS.$ P realisiert Q $\Rightarrow$
sei $RPQ(P,Q) = (PL,PI,PS,PT)$;
dann gilt:

(1) $\forall i \in \{1,\ldots,m\}.\ \forall j \in \{1,\ldots,n_i\}.$
$PI_{i,j} \in MI_{i,j}$

(2) $\forall i \in \{1,\ldots,m\}.\ \forall j \in \{0,\ldots,n_i-1\}.$
$b \in \{\underline{s},\underline{a},\underline{as}\} \Rightarrow PT_{i,j} = MT_{i,j}$
$\wedge\ b = \underline{ss} \Rightarrow (PI_{i,j},PT_{i,j}) \in MT_{i,j}$

(3) $PS = S$

Beweis:

folgt direkt aus Satz 2.1 und Def. 3.3 und 3.4 .

3.2. Stufengraphdarstellungen

Die in Form von Matrizen vorliegende Information in Standarddarstellungen wird in Stufengraphdarstellungen überführt, die in geeigneter Weise die in den Sätzen 3.1 und 3.2 gezeigte Eigenschaft zum Ausdruck bringen, unter der Elemente der Matrizen identisch sind.

3.2.1 Stufengraphdarstellung für eine Instruktionsmatrix

Gegeben sei die Standarddarstellung eines PSP Q mit der Speicherbelegungsmatrix S. In einem Programm, das Q realisiert, entspricht jeder Speicherbelegungsübergang von $s_{i,j}$ nach $s_{i,j+1}$ genau einem Programmknoten. Entsprechen in jedem Q realisierenden Programm P die Übergänge $(s_{i,j},s_{i,j+1})$ und $(s_{i',j'},s_{i',j'+1})$ jeweils demselben Programmknoten, so wäre es wünschenswert, diese Tatsache bereits bei der Programmsynthese ausnützen zu können und bei der Konstruktion des Zielprogramms zu berücksichtigen.

Um dies zu ermöglichen, werden die den beiden Übergängen zugeordneten Elemente der Instruktionsmatrix $MI_{i,j}$ und $MI_{i',j'}$ "zusammengefaßt". Für diese Zusammenfassung sind zunächst 3 Punkte von Bedeutung: zum einen sollen in $MI_{i,j}$ und $MI_{i',j'}$ nur noch beiden Mengen gemeinsame Elemente berücksichtigt werden, zum anderen sollen bei der Programmsynthese immer nur jeweils identische Elemente daraus als mögliche Instruktionen ausgewählt werden und drittens

sollen auch nur jeweils gleiche Markierungen (LABEL) dafür berücksichtigt werden.

Da alle Berechnungsfolgen mit Durchlaufen des START-Knotens beginnen, können entsprechend alle $MI_{i,0}$ auf diese Weise zusammengefaßt werden. In Abhängigkeit von den vorliegenden Transitionen MT und mit Hilfe der Sätze 3.1 und 3.2 können die $MI_{i,1}$ in Äquivalenzklassen (bei b = ss ist eine Modifizierung notwendig, s.u.) aufgeteilt werden, wobei diejenigen zusammengefaßt werden, die in jedem Q realisierenden Programm P einem jeweils identischen Programmknoten entsprechen müssen. Eine Fortsetzung dieses Verfahrens zeigt, daß dies immer nur für Elemente von MI zutrifft, die in der gleichen Spalte liegen. Für den Fall b∈{s,a,as} bilden diese Zusammenfassungen bzgl. der entsprechenden Zeilenindizes fortlaufende Verfeinerungen der trivialen einelementigen Zerlegung; in diesem Fall kann die Matrix MI in einfacher Weise in einen Baum überführt werden, der gerade die gewünschte Information über die "Zusammenfassungen" von Elementen von MI enthält.

Für den Fall b = ss ergibt sich jedoch eine etwas andere Situation. Zwar gilt auch hier, daß Zusammenfassungen von Elementen von MI nur möglich sind, wenn sie gleichen Spaltenindex haben (d.h. auf der gleichen Stufe liegen); sind jedoch z.B. die den Indizes $(i_1,j),(i_2,j)$ und (i_3,j) entsprechenden Elemente zusammengefaßt, so gibt es im allgemeinen mehrere Instruktionen, mit denen der Programmknoten markiert sein kann, der den drei Übergängen entspricht.

Trifft dies z.B. für eine Testinstruktion T zu, die für zwei der beteiligten Speicherbelegungen (Indizes i_1 und i_2) true, für die dritte (i_3) aber false liefert, so müssen, falls T an dieser Stelle ausgewählt wird, aufgrund der Eindeutigkeitseigenschaft für Programme auch die den Indizes $(i_1,j+1)$ und $(i_2,j+1)$ entsprechenden Übergänge demselben Programmknoten in P zugeordnet werden, während dies für $(i_3,j+1)$ nicht notwendigerweise gilt. Auf der Stufe j+1 müßten also $MI_{i_1,j+1}$ und $MI_{i_2,j+1}$ zusammengefaßt werden; würde dagegen an Stelle von T z.B. eine Testinstruktion T' ausgewählt, die für alle drei Speicherbelegungen true liefert, so müßte dies für alle drei MI-Elemente auf der Stufe j+1 geschehen.

Allgemein gesprochen gibt es also für die verschiedenen Spalten mehrere Zerlegungen der Zeilenindizes, und zwar in Abhängigkeit von den jeweils zuvor ausgewählten Instruktionen.

Dies skizziert den Aufbau eines (nicht reduzierten) Stufengraphs für MI. Ein Stufengraph ist ein markierter gerichteter Graph, dessen Knoten Paare aus Spaltenindex und Menge von Zeilenindizes sind, die gerade den "zusammengefaßten" Elementen von MI entsprechen. Die Knoten sind markiert mit der Menge der Instruktionen, die alle die betreffenden Speicherbelegungsübergänge erklären, nach den obigen Überlegungen also mit der Durchschnittsmenge der beteiligten $MI_{i,j}$. Eine Kantenmarkierungsfunktion gibt an, unter welchen angenommenen Instruktionen

auf der Stufe j der so markierte Übergang zu der Zerlegung auf der Stufe j+1 führt. Letzteres ist für $b\varepsilon\{\underline{s},\underline{a},\underline{as}\}$ trivial, da nur im Fall b=ss die Auswahl der Instruktionen Einfluß auf die Zerlegungen hat.

Führt in dem oben angegebenen Beispiel die Auswahl von T' dazu, daß es keine Instruktionen gibt, die alle drei folgenden Speicherbelegungsübergänge auf der Stufe j+1 erzeugen kann, so führt diese Auswahl von T' zwangsläufig zum Widerspruch: es gibt kein Programm P, das Q realisiert und die drei vorliegenden Übergänge auf der Stufe j durch die Instruktion T' erzeugt.

Um derartige Widersprüche bei der späteren dynamischen Programmsynthese von vorneherein zu vermeiden, entfernt man systematisch alle dermaßen als nicht realisierbar erkannten Instruktionen T' und erhält so die reduzierte Stufengraphdarstellung von MI. Auch dies ist für den Fall $b\varepsilon\{\underline{s},\underline{a},\underline{as}\}$ sehr viel einfacher als für b=ss; in der folgenden Definition werden deshalb die beiden Fälle getrennt behandelt, um das Vorgehen zu verdeutlichen, obwohl beide durch das allgemeinere Verfahren (b = ss) abgedeckt werden.

Definition 3.5

Q sei PSP, $Q_{ST} = (b,S,MI,MT)$, $dim(Q) = (m,(n_1,\ldots,n_m))$,
$n = max\{n_1,\ldots,n_m\}$.

(1) (i) Für $j\varepsilon\{0,\ldots,n\}$ sei $IND_j = \{i \mid i\varepsilon\{1,\ldots,m\}, j \leq n_i\}$.

(ii) N sei die Menge

$$\{[j,IND] \mid j \in \{0,\ldots,n\},\ IND \subseteq IND_j\},$$

deren Elemente Knoten heißen.

(iii) Für $K \in N, K = [j,IND]$ heißt

$LEVEL(K) = j$ Stufe des Knoten K,

$INDEX(K) = IND$ Indexmenge des Knoten K.

(iv) D sei die Abbildung

$$D : N \longrightarrow POT(INSTRUCTIONS^1)$$

$$D = \lambda K. \bigcap_{i \in INDEX(K)} MI_{i,LEVEL(K)}$$

Damit gilt:

(2) (V,E,f,g) heißt nicht reduzierte Stufengraphdarstellung von MI

<=>

(2.1) falls $b \in \{s,a,as\}$:

Für $j \in \{0,\ldots,n\}$ sei

(a) $INDZ_{MT}(j)$ die durch $\sim_j$ induzierte Zerlegung von IND_j, wobei:

$$i_1 \sim_j i_2 \iff \forall k \in \{0,\ldots,j-1\}.\ MT_{i_1,k} = MT_{i_2,k}$$

(b) $V_j = \{[j,IK] \mid IK \in INDZ_{MT}(j)\}$

Dann gilt:

(i) $V = V_0 \cup \ldots \cup V_n$

(ii) $E = \{(K,K') \mid LEVEL(K') = LEVEL(K)+1,\ INDEX(K') \subseteq INDEX(K)\}$

(iii) $f : V \longrightarrow POT(INSTRUCTIONS^1)$

$f = \lambda K.\ D(K)$

(iv) $g : E \longrightarrow POT(INSTRUCTIONS^1)$

$g = \lambda(K,K').\ f(K)$

(2.2) falls b = ss :

(a) $V_0 = \{[0,\{1,\ldots,m\}]\}$

Für $K \in N, I \in D(K), I \neq STOP$ sei:

(b) $INDZ_{MT}(K,I)$ die durch $\sim_{K,I}$ induzierte Zerlegung von INDEX(K), wobei:

$$i_1 \sim_{K,I} i_2 \iff MT^I_{i_1,LEVEL(K)} = MT^I_{i_2,LEVEL(K)}$$

Für $j \in \{1,\ldots,n\}$ sei:

(c) $V_j = \{[j,IK] \mid \exists K \in V_{j-1}, I \in D(K), I \neq STOP,\ IK \in INDZ_{MT}(K,I)\}$

Dann gilt:

(i) $V = V_0 \cup \ldots \cup V_n$

(ii) $E = \{(K,K') \mid LEVEL(K') = LEVEL(K)+1, \exists I \in D(K), I \neq STOP,\ INDEX(K') \in INDZ_{MT}(K,I)\}$

(iii) $f : V \longrightarrow POT(INSTRUCTIONS^1)$

$f = \lambda K.\ D(K)$

(iv) $g : E \longrightarrow POT(INSTRUCTIONS^1)$

$g = \lambda(K,K').\ \{I \mid I \in f(K), I \neq STOP,\ INDEX(K') \in INDZ_{MT}(K,I)\}$

(3) (V,E,f,g) sei nicht-reduzierte Stufengraphdarstellung von MI.

(V',E',f',g') heißt (reduzierte) Stufengraphdarstellung von MI

<=>

(3.1) falls $b \in \{s,a,as\}$:

$$(V',E',f',g') = \begin{cases} (V,E,f,g) & \iff \forall K \in V.\ f(K) \neq \emptyset \\ (\emptyset,\emptyset,\emptyset,\emptyset) & \iff \exists K \in V.\ f(K) \neq \emptyset \end{cases}$$

(3.2) falls b = ss :

Für $j \in \{0,\ldots,n\}$ sei:

(a) $F_j : V \longrightarrow POT(INSTRUCTIONS^{-1})$

$(j=0) \quad F_0 = \lambda K.\ \emptyset$

$(j>0) \quad F_j = \lambda K. \bigcup_{\substack{K' \\ (K,K') \in E, f(K') \setminus F_{j-1}(K') = \emptyset}} g(K,K') \cup F_{j-1}(K)$

Weiterhin sei:

(b) $V'' = \{K \mid K \in V,\ f(K) \setminus F_n(K) \neq \emptyset\}$

Dann gilt:

(i) $V' = \{K \mid K \in V'', \exists$ Weg in $E \cap (V'' \times V'')$ von $[0,1,\ldots,m]$ nach $K\}$

(ii) $E' = (E \cap (V' \times V')) \setminus \{(K,K') \mid (K,K') \in E \wedge g(K,K') \subseteq F_n(K)\}$

(iii) $f' : V' \longrightarrow POT(INSTRUCTIONS^{-1})$

$f' = \lambda K.\ f(K) \setminus F_n(K)$

(iv) $g' : E' \longrightarrow POT(INSTRUCTIONS^{-1})$

$g' = \lambda (K,K').\ g(K,K')$

Ein Berechnungsverfahren für die reduzierte Stufengraphdarstellung von MI bietet sich aufgrund dieser Definition direkt an. Stufenweise werden die Elemente von V gebildet, beginnend mit dem eindeutig bestimmten Knoten der Stufe 0. Für Stufe j+1 werden die Knoten der Stufe j "expandiert", indem die Instruktionsmenge f(K) eines Knoten K in Äquivalenzklassen zerlegt wird, wobei zwei Instruktionen äquivalent (bzgl. der oben erwähnten Realisierbarkeit) sind, wenn sie auf INDEX(K) dieselbe Zerlegung definieren. Die Elemente der Instruktionsmengenzerlegung bilden die

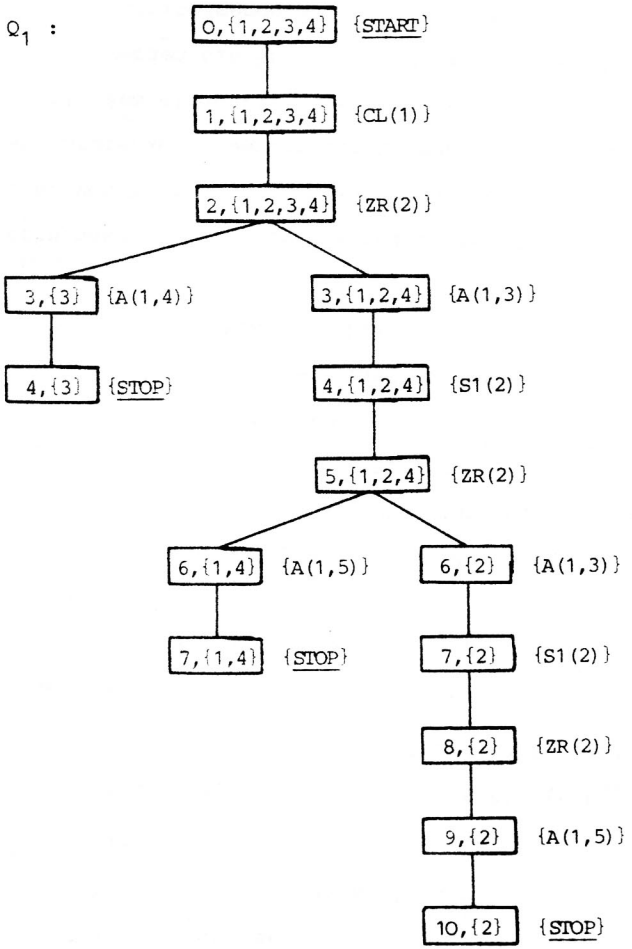
Kantenmarkierungen g(K,K'), und die Indexklassen bestimmen die Knoten der Stufe j+1.

Wird ein Knoten mit leerer Instruktionsmenge erzeugt, so wird er nicht in V aufgenommen, sondern die dadurch als nicht realisierbar erkannten Instruktionen - also gerade die Kantenmarkierungen g(K,K'), die zu diesem Knoten hinführen - werden aus den Markierungen der Knoten der vorhergehenden Stufe eliminiert. Wird dadurch die Markierung eines Knoten leer, so setzt sich dieser Eliminierungsprozeß eventuell über mehrere Stufen fort. Knoten, die nur noch über Kanten mit bereits eliminierten Instruktionen erreichbar sind, werden ebenfalls aus V entfernt. Gibt es in V keine zu expandierenden Knoten mehr, so bricht das Verfahren ab.

Beispiel 3.3 :

Für Q_1,Q_2,Q_3 und Q_4 werden die Stufengraphdarstellungen der zugehörigen MI (Beispiel 3.1) angegeben.

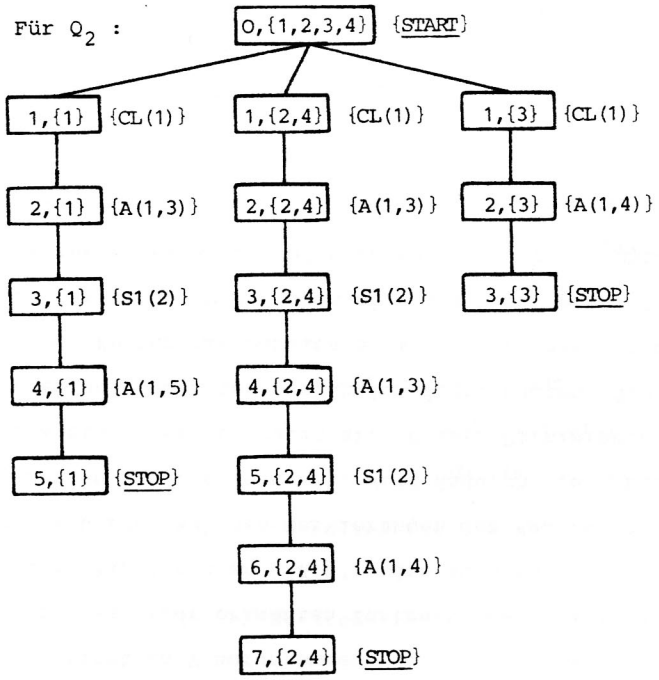
1. Für Q_1 :



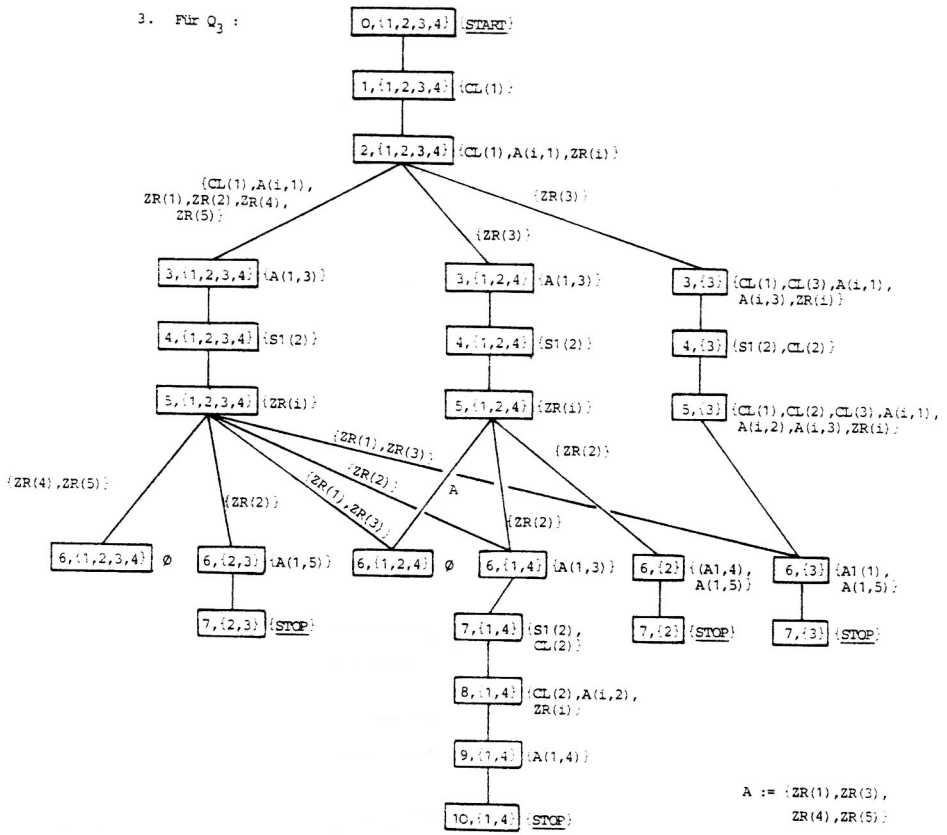
Die Markierung f(K) eines Knoten K ist durch die jeweils nebenstehende Menge gegeben; die Kantenmarkierung g(K,K') ist nicht explizit mit angegeben, da sie sich direkt aus f(K) ergibt. Es gibt keinen Knoten, der mit der leeren Menge markiert ist; die reduzierte ist also gleich der nicht reduzierten Darstellung.

Diese Punkte treffen auch für Q_2 und Q_4 (s.u.) zu.

2. Für Q_2 :



3. Für Q_3 :



Bei dieser nicht reduzierten Darstellung von MI steht A(i,1) abkürzend für A(1,1), A(2,1),...,A(5,1); entsprechend A(i,2), ZR(i) usw. Bei den Kanten (K,K'), für die keine Markierung angegeben ist, gilt g(K,K') = f(K).

Die Knoten [6,{1,2,3,4}] und [6,{1,2,4}] sind mit der leeren Menge markiert. Es gibt also keine Instruktion, die z.B. die drei Speicherbelegungsübergänge, die [6,{1,2,4}] entsprechen, erzeugt. Nun führt aber etwa die Annahme von ZR(1) bei Knoten [5,{1,2,3,4}] zwangsläufig zu [6,{1,2,4}], da ZR(1) in der entsprechenden Kantenmarkierung enthalten ist. Daher wird ZR(1) (und ZR(3)) aus der Markierung von [5,{1,2,3,4}] entfernt. Dies geschieht mit Hilfe der in Def. 3.5, Abschnitt 3.2, benutzten Funktion F_i. Mit den dort verwandten Bezeichnungen gilt:

$n = 10$

$F_{10}([5,\{1,2,3,4\}]) = \{ZR(1),ZR(3),ZR(4),ZR(5)\}$

$F_{10}([5,\{1,2,4\}]) = \{ZR(1),ZR(3),ZR(4),ZR(5)\}$

$F_{10}(K) = \emptyset$ sonst.

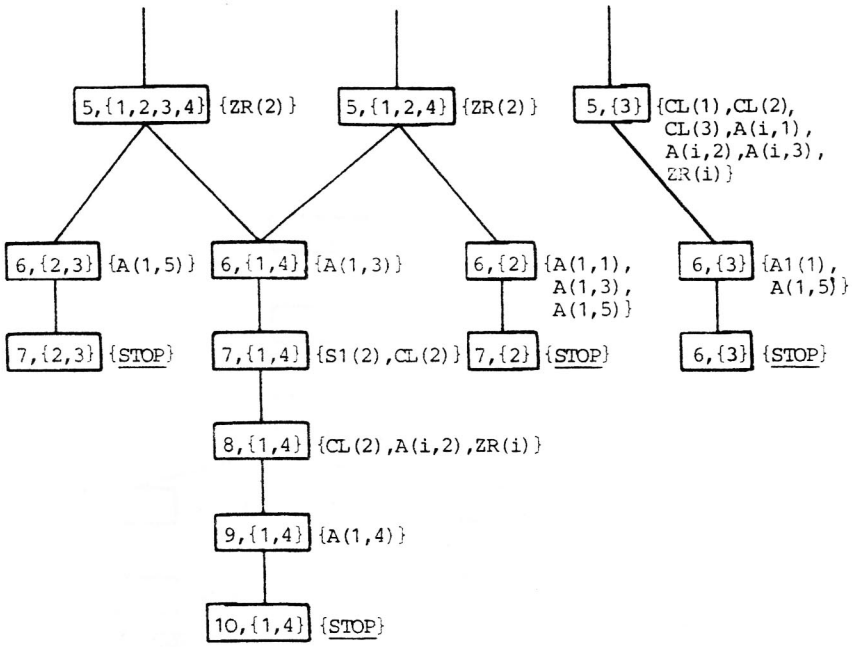
Weiterhin gilt:

$V'' = V \setminus \{[6,\{1,2,3,4\}],[6,\{1,2,4\}]\}$

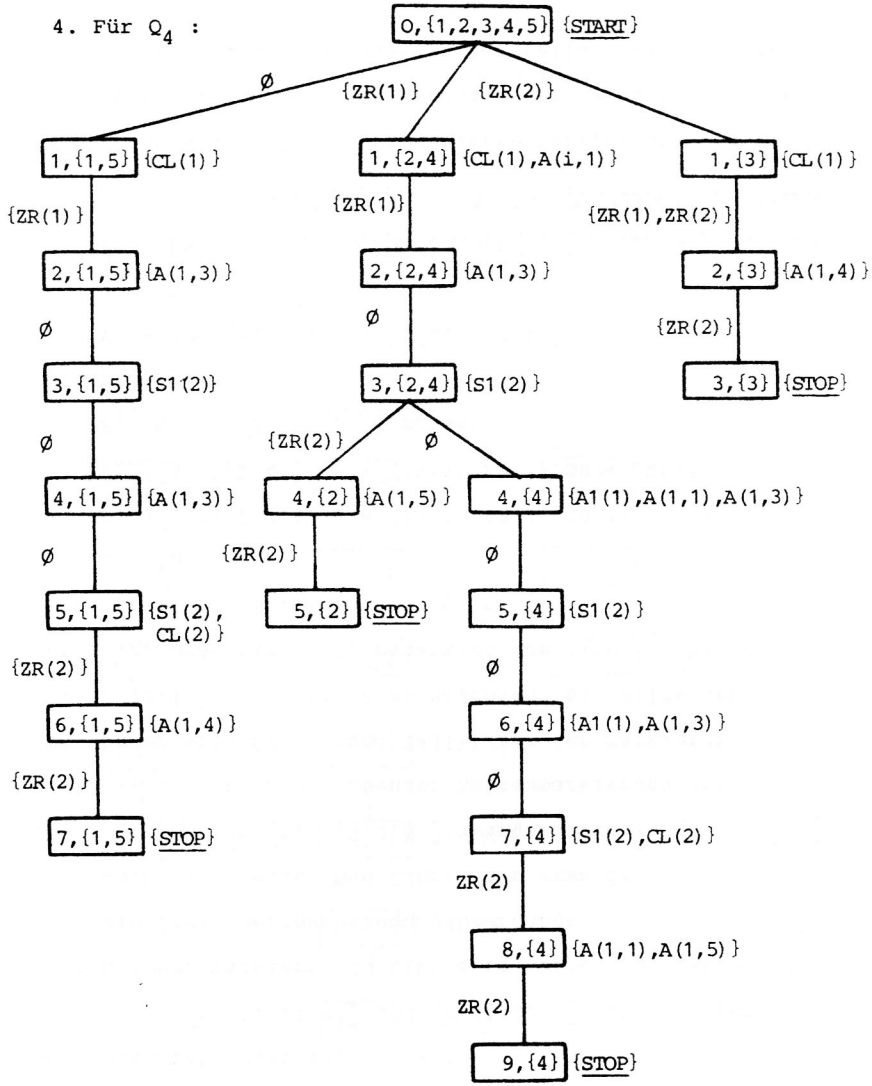
$V' = V''$

$E' = E \setminus \{([5,\{1,2,3,4\}],[6,\{1,2,3,4\}]),([5,\{1,2,3,4\}],[6,\{1,2,4\}]),$
$([5,\{1,2,3,4\}],[6,\{3\}]),([5,\{1,2,4\}],[6,\{1,2,4\}])\}$

Da bei den Stufen 0-4 die reduzierte Darstellung von MI identisch ist mit der nicht-reduzierten, werden i.f. nur die Stufen 5-10 der reduzierten Darstellung wiedergegeben.



4. Für Q_4 :



Bemerkung: Die bei Q_4 links von den Kanten stehenden Markierungen werden erst später in Def. 3.7 eingeführt.

3.2.2. Stufengraphdarstellung einer Realisierung

Die in 3.2.1 gemachten Überlegungen gelten auch für die Stufengraphdarstellung der Realisierung eines PSP Q durch ein Programm P. Zusätzlich sind hier eine Knotenmarkierungsfunktion u_p und eine Kantenmarkierungsfunktion c_p vorhanden, die die durch PL über die LABEL der jeweils durchlaufenen Programmknoten bzw. durch PT über die erzeugten b'-Transitionen gegebenen Informationen wiedergeben.

Definition 3.6

Q sei PSP, $\dim(Q) = (m,(n_1,\ldots,n_m)), n = \max\{n_1,\ldots,n_m\}$

$Q = (b,M)$. P sei Programm, das Q realisiert,

$RPQ(P,Q) = (PL,PI,PS,PT)$.

(1) Für $j \in \{0,\ldots,n\}$ sei:

(a) $IND_j = \{i \mid i \in \{1,\ldots,m\},\ j \leq n_i\}$

(b) $INDZ_{PT}(j)$ die durch $\sim_j$ induzierte Zerlegung von IND_j, wobei:

$$i_1 \sim_j i_2 \iff k \in \{0,\ldots,j-1\}.\ PT_{i_1,k} = PT_{i_2,k}$$

(c) $V_j = \{[j,IK] \mid IK \in INDZ_{PT}(j)\}$

Damit gilt:

(2) $RPQ - SG(P,Q) = (V_p,E_p,a_p,c_p,u_p)$ heißt Stufengraphdarstellung der Realisierung von Q durch P $\iff$

(i) $V_p = V_0 \cup \ldots \cup V_n$

(ii) $E_p = \{(K,K') \mid LEVEL(K') = LEVEL(K)+1, INDEX(K') \subseteq INDEX(K)\}$

(iii) $a_p : V_p \longrightarrow INSTRUCTIONS^1$

$a_p = \lambda K.\ i \in INDEX(K) \longrightarrow PI_{i,LEVEL(K)}$

(iv) $c_p : E_p \longrightarrow TRANS'_b$

$c_p = \lambda(K,K').\ i \in INDEX(K') \longrightarrow PT_{i,LEVEL(K)}$

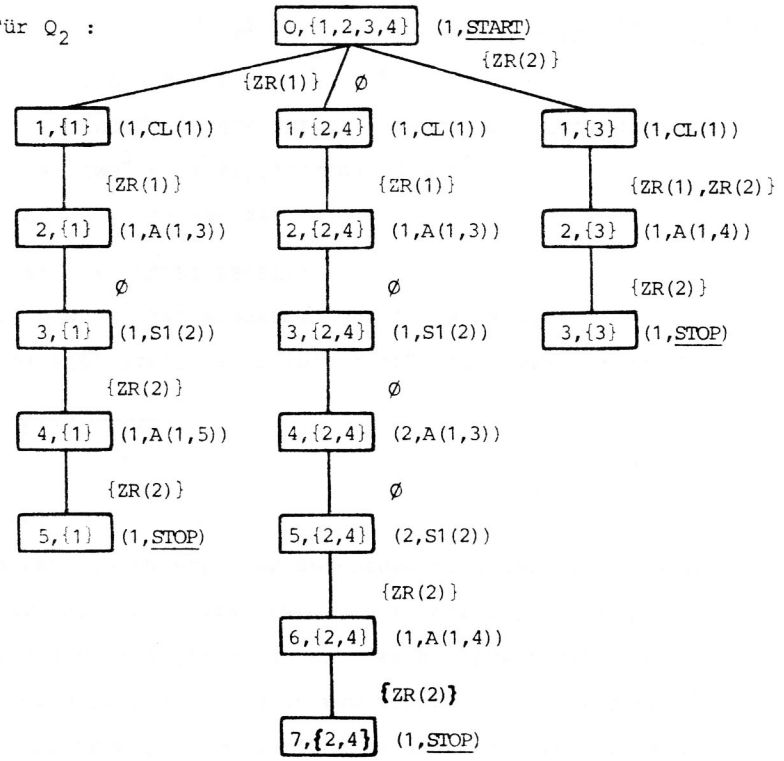
(v) $u_p : V_p \longrightarrow \mathbb{N}$

$u_p = \lambda K.\ i \in INDEX(K) \longrightarrow PL_{i,LEVEL(K)}$

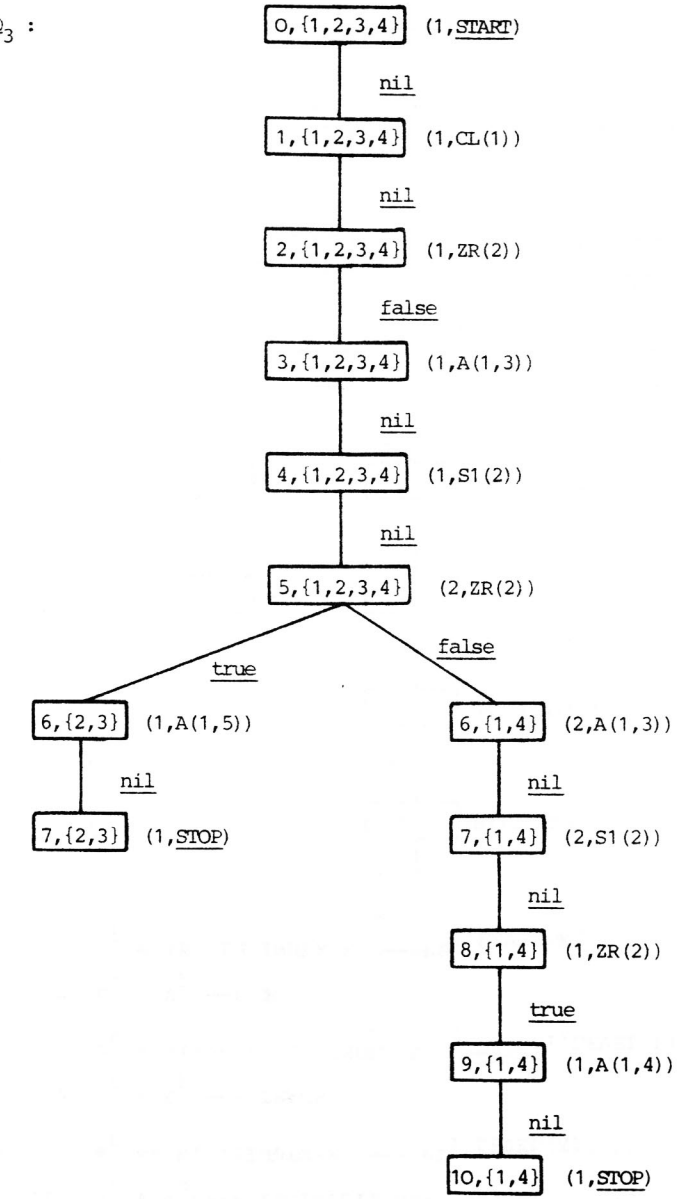
Beispiel 3.4

Für Q_2 und Q_3 werden die Stufengraphendarstellungen ihrer Realisierungen durch das Beispielprogramm P angegeben. Das rechts von einem Knoten K stehende Tupel gibt dessen Markierungen $(u_p(K), a_p(K))$ wieder; eine Kante (K,K') ist mit $c_p(K,K')$ gekennzeichnet.

1. Für Q_2 :



2. Für Q_3 :



3.2.3 Realisierbarkeitssatz für PSP

Die in Satz 3.3 ausgedrückte Beziehung zwischen den Standarddarstellungen überträgt sich auch auf die Stufengraphdarstellungen. Satz 3.4 sagt aus, daß die Stufengraphdarstellung eines PSP Q gewissermaßen die "Vereinigung" aller derartiger Darstellungen von Realisierungen von Q ist.

Satz 3.4

Q sei PSP, $Q_{ST} = (b,s,MI,MT)$, $dim(Q) = (m,(n_1,\ldots,n_m))$.
(V,E,f,g) sei nicht-reduzierte Stufengraphdarstellung von MI.
P sei ein Programm, das Q realisiert; $RPQ(P,Q)=(PL,PI,PS,PT)$,
$RPQ - SG(P,Q) = (V_p,E_p,a_p,c_p,u_p)$.

Behauptung:

(1) RPQ-SG(P,Q) ist eindeutig bestimmt.

(2) (V_p,E_p) ist ein Baum mit der Wurzel $[0,\{1,\ldots,m\}]$.

(3) falls $b\in\{\underline{s},\underline{a},\underline{as}\}$, gilt:

(3.1) $V_p = V$

(3,2) $E_p = E$

(3.3) $K \in V.\ a_p(K) \in f(K)$

(4) falls $b=\underline{ss}$, gilt:

(4.1) $V_p \subseteq V$

(4.2) $E_p \subseteq E$

(4.3) $\forall K \in V_p.\ a_p(K) \in f(K)$

(4.4) $\forall (K,K')\in E_p.\ a_p(K)\in g(K,K')$

(4.5) $\forall (K,K')\in E.\ (K\in V_p \wedge a_p(K)\in g(K,K') \Rightarrow K'\in V_p \wedge (K,K')\in E_p)$

Bemerkung: (4.1)-(4.6) gelten natürlich auch, falls $b\in\{\underline{s},\underline{a},\underline{as}\}$.

Beweis:

(1) Zu zeigen ist die Eindeutigkeit der Abbildungen a_p,c_p und u_p. Eindeutigkeit von c_p folgt direkt aus der Definition von $INDZ_{PT}(j)$, V_p und E_p. Eindeutigkeit von a_p und u_p folgt aus den Sätzen 3.1 und 3.2.

(2) $INDZ_{PT}(0)$ liefert die einelementige Zerlegung $\{1,\ldots,m\}$, sodaß $[0,\{1,\ldots,m\}]$ einziger Knoten der Stufe 0 in V_p ist. Kanten gibt es nur zwischen Knoten benachbarter Stufen. Damit folgt, daß für $j\in\{1,\ldots,n\}$ $INDZ_{PT}(j)$ eine Verfeinerung der durch $INDZ_{PT}(j-1)$ auf IND_j definierten Zerlegung ist.

(3) Für $b\in\{\underline{s},\underline{a},\underline{as}\}$ ist nach Satz 3.3 PT = MT, also auch $INDZ_{PT}(j) = INDZ_{MT}(j)$ und damit $V_p = V$ und $E_p = E$.
Sei $K \in V$, $K = [j,IK]$.

$\forall i\in IND.\ a_p(K) = PI_{i,j}$ [nach (1)]

$\forall i\in IND.\ PI_{i,j} \in MI_{i,j}$ [nach Satz 3.3]

$\Rightarrow\ a_p(K) \in \bigcap_{i\in IK} MI_{i,j} = f(K)$

(4) Für $j \geq 0$ sei A[j] die Aussage:

A[j]: (a) $\forall K \in V_p$. LEVEL(K)=j => $K \in V \wedge a_p(K) \in f(K)$

$\wedge$ (b) $\forall K \in V_p$. LEVEL(K)=j =>

$\forall (K',K) \in E_p$. $(K',K) \in E \wedge a_p(K') \in g(K,K')$

$\wedge$ (c) $\forall K \in V$. LEVEL(K)= j =>

$\forall (K',K) \in E$ $(K' \in V_p \wedge a_p(K') \in g(K',K)$ =>

$K \in V_p \wedge (K',K) \in E_p)$

Für alle j wird A[j] durch Induktion bewiesen.

Induktionsanfang:

A[0] : Sowohl in V als auch in V_p ist $K_0 = [0,\{1,\ldots,m\}]$ einziger Knoten der Stufe 0 und es gilt:

$a_p(K_0)$ = START $\in$ {START} = $f(K_0)$

A[j] : Für j>0 sei A[j] bereits bewiesen.

A[j+1] :

zu (a): Sei $K \in V_p$, K = [j+1,IK]. Da (V_p,E_p) nach (1) ein Baum ist, gibt es einen eindeutig bestimmten Knoten $K' \in V_p$ mit $(K',K) \in E_p$. Es gilt LEVEL(K') = j; sei IND = INDEX(K'). Dann gilt:

(i) $a_p(K') \neq$ STOP [nach Def. 3.6]

(ii) $K' \in V \wedge a_p(K') \in f(K')$ [Ind.-Annahme]

(iii) $IND \in INDZ_{PT}(j)$ [nach Def. 3.6]

$IK \in INDZ_{PT}(j+1)$

$IK \subseteq IND$

(iv) $\forall i \in IND.\ a_p(K') = PI_{i,j}$ [nach (1)]

(v) $\forall i \in IND.\ PI_{i,j} \in MI_{i,j}$ [Satz 3.3]

$\wedge (PI_{i,j},PT_{i,j}) \in MT_{i,j}$

(vi) $\forall i \in IND.\ MI_{i,j}^{PI_{i,j}} = PT_{i,j}$ [Def.3.3]

(vii) $INDZ_{PT}(j+1)$ induziert auf IND genau dieselbe Zerlegung wie $INDZ_{MT}(K',a_p(K'))$ [(vi),Def. 3.5 u. 3.6]

(viii) $IK \in INDZ_{MT}(K',a_p(K'))$ [(iii),(vii)]

=> $K=[j+1,IK] \in V$ [(i),(ii),

$\wedge (K',K) \in E$ Def.3.5]

$\wedge a_p(K') \in g(K',K)$

(ix) $\forall i \in IK.\ a_p(K) = PI_{i,j+1}$ [nach (1)]

$\forall i \in IK.\ PI_{i,j+1} \in MI_{i,j+1}$ [Satz 3.3]

$=> a_p(K) \in \bigcap_{i \in IK} MI_{i,j+1} = f(K)$

zu (b): Da es für jeden Knoten in V_p nur einen Vorgängerknoten in V_p geben kann, wurde Teil (b) der Aussage bereits in (viii) gezeigt.

zu (c): Sei $K \in V$, K = [j+1,IK], $(K',K) \in E$, K' = [j,IND], $K' \in E_p$, $a_p(K') \in g(K',K)$. Zu zeigen ist, daß $K \in V_p$ und $(K',K) \in E_p$.

(x) $a_p(K') \neq$ STOP [Def.3.5]

(xi) $IK \in INDZ_{MT}(K',a_p(K'))$ [Def.3.5 u.3.6]

$IND \in INDZ_{PT}(j)$

$IK \subseteq IND$

(xii) $\forall i \in IND.\ MI_{i,j}^{PI_{i,j}} = PT_{i,j}$ [analog (iv)-(vi)]

(xiii) $INDZ_{MT}(K',a_p(K'))$ induziert auf IND genau dieselbe Zerlegung wie $INDZ_{PT}(j+1)$ [(xi),(xii), Def.3.5,3.6]

=> $K = [j+1,IK] \in V_p$

$\wedge (K',K) \in E_p$

Aus der erfolgreichen Ausführung des Induktionsschrittes folgt die Gültigkeit von A[j] für alle $j \geq 0$ und damit auch die Aussagen (4.1)-(4.5), was den Beweis von Satz 3.4 abschließt.

Beispiel 3.5

Ein Vergleich der Realisierungsdarstellungen von Q_2 und Q_3 aus Beispiel 3.4 mit den Stufengraphdarstellungen der entsprechenden MI in Beispiel 3.3 zeigt, daß die Bedingungen des Satzes 3.4 erfüllt sind.

In der reduzierten Stufengraphdarstellung von MI sind nur noch solche Instruktionen I in den Markierungsmengen der Knoten enthalten, für die es ein Programm P gibt, das Q realisiert und das gerade durch I die entsprechenden Speicherbelegungsübergänge erzeugt. Dies führt zu dem Realisierbarkeitskriterium für PSP, das im folgenden Satz bewiesen wird.

Satz 3.5

Q sei PSP, Q_{ST} = (b,S,MI,MT).
(V',E',f',g') sei (reduzierte) Stufengraphdarstellung von MI.

Behauptung:

V' = ∅ <=> Q ist nicht realisierbar.

Beweis:

Es wird hier zunächst nur die Richtung " => " bewiesen; der Beweis der Gegenrichtung erfolgt durch Synthese eines Programms P, das Q realisiert, falls V' ≠ ∅.
(V,E,f,g) sei die nicht-reduzierte Stufengraphdarstellung von MI.

Widerspruchsannahme: $V' = \emptyset \wedge P \in PROGRAMS$.

P realisiert Q ; PRQ-SG(P,Q) = $(V_p, E_p, a_p, c_p, u_p)$

(1) falls $b \in \{\underline{s}, \underline{a}, \underline{as}\}$:

[nach Def. 3.5] $V' = \emptyset \Rightarrow \exists\, K \in V.\ f(K) = \emptyset$

[nach Satz 3.4] $\forall\, K \in V.\ a_p(K) \in f(K)$

$\Rightarrow \forall\, K \in V.\ f(K) \neq \emptyset$ Widerspruch!

(2) falls b = ss :

Seien m, n, F_n und V" wie in Def. 3.5 (3.2) ; $K_o = [0, \{1, \ldots, m\}]$

(i) $\forall\, K \in V.\ f(K) \setminus F_n(K) = \emptyset \Leftrightarrow f(K) = F_n(K)$ [nach Def. 3.5]

(ii) $\forall\, K \in V_p.\ K \in V \wedge a_p(K) \in f(K)$ [Satz 3.4]

(iii) $\forall\, K \in V_p.$ [(i) und (ii)]

$f(K) \setminus F_n(K) = \emptyset \Rightarrow a_p(K) \in F_n(K)$

(iv) $\forall\, K \in V_p.$

$a_p(K) \in F_n(K)$

$\Rightarrow \exists\, K' \in V.\ (K,K') \in E \wedge a_p(K) \in g(K,K') \wedge F_n(K') = f(K')$ [Def. 3.5]

$\Rightarrow \exists\, K' \in V_p.\ (K,K') \in E_p \wedge a_p(K') \in F_n(K')$ [Satz 3.4 u.(iii)]

$\Rightarrow \exists\, (K_o,K_1), (K_1,K_2), \ldots, (K_n,K_{n+1}) \in E_p$ [nach n-maliger Wiederholung der beiden letzten Ableitungen]

$\Rightarrow \exists\, K \in V_p.$ LEVEL(K) > n im Widerspruch zur Def. von V_p

(v) $\forall K \in V_p .\ a_p(K) \notin F_n(K)$ [nach (iv)]

(vi) $V' = \emptyset \Rightarrow K_o \notin V''$ [Def. 3.5]

$\Rightarrow f(K_o) \setminus F_n(K_o) = \emptyset$ ["]

(vii) $K_o \in V_p \Rightarrow a_p(K_o) \notin F_n(K)$ [nach (v)]

$\Rightarrow f(K_o) \setminus F_n(K_o) \neq \emptyset$ [" (iii)]

Widerspruch!

3.2.4 Stufengraphdarstellung eines PSP

Übernimmt man auch die durch MT gegebene Information eines standardisierten PSP Q mit in dessen Stufengraphdarstellung von MI, so werden die Matrizen S,MI und MT für die Programmsynthese nicht mehr benötigt.
Die zwischen Knoten benachbarter Stufen vorliegenden Transitionen werden durch eine zusätzliche Kantenmarkierungsfunktion wiedergegeben.

Satz 3.6.

Q sei PSP, Q_{ST} = (b,S,MI,MT).
(V,E,f,g) sei Stufengraphdarstellung von MI, $j \in \mathbb{N}$.

Behauptung:

Für jede Kante $(K,K') \in E$ mit LEVEL(K) = j gilt:

$\forall i_1, i_2 \in$ INDEX(K').

$(b \in \{\underline{s},\underline{a},\underline{as}\} \Rightarrow MT_{i_1,j} = MT_{i_2,j})$

$(b = \underline{ss} \Rightarrow MT_{i_1,j} \cap (g(K,K') \times TFN) = MT_{i_2,j} \cap (g(K,K') \times TFN))$

Beweis: analog Satz 3.4.

Der obige Satz garantiert die Eindeutigkeit der i.f. definierten Stufengraphdarstellung eines PSP Q .

Definition 3.7.

Q sei PSP, Q_{ST} = (b,S,MI,MT) .
(V,E,f,g) sei Stufengraphdarstellung von MI.
Q_{SG} = (b,V,E,f,g,t) heißt Stufengraphdarstellung von Q <=>
t : E ⟶ $TRANS_b$ ist eine Kantenmarkierungsfunktion, die definiert ist durch:
sei $(K,K') \in E$, j=LEVEL(K), $i \in$ INDEX(K') :

$$t(K,K') = \begin{cases} MT_{i,j} & <=> \ b \in \{\underline{s},\underline{a},\underline{as}\} \\ MT_{i,j} \cap (g(K,K') \times TFN) & <=> \ b = \underline{ss} \end{cases}$$

Notation: Analog zur Definition von MT (Def. 3.3) sei für $I \in g(K,K')$:

$$t^I(K,K') = \begin{cases} t(K,K') & <=> \ b \in \{\underline{s},\underline{a},\underline{as}\} \\ m & <=> \ b = \underline{ss} \wedge (I,m) \in t(K,K') \end{cases}$$

Beispiel 3.6

Die in Beispiel 3.3-4 links von einer Kante (K,K') stehende Menge gibt den Wert t(K,K') wieder, wobei t die für die Stufengraphdarstellung von Q_4 notwendige Kantenmarkierungsfunktion ist.

3.2.5 Reduzierte Stufengraphdarstellungen

3.2.5.1 Instruktionshypothesengraph

Die Stufengraphdarstellung Q_{SG} einer PSP Q kann i.a. noch weiter reduziert werden. Jeder Beispielsfolge i entsprechen in Q_{SG} Wege vom Knoten der Stufe 0 bis zu einem Endknoten. Sind nun für zwei Folgen i und i' alle diese Wege jeweils identisch, so genügt es, nur eine dieser beiden Folgen bei der weiteren Programmsynthese zu berücksichtigen.

Definition 3.8

Q sei PSP, $dim(Q) = (m,(n_1,\ldots,n_m))$,
$Q_{SG} = (b,V,E,f,g,t)$, $V \neq \emptyset$.

(1) Zwei Indizes $i_1,i_2 \varepsilon \{1,\ldots,m\}$ heißen redundant bzgl. Q_{SG}
$\Leftrightarrow \forall K \varepsilon V.\ (i_1 \varepsilon INDEX(K) \Leftrightarrow i_2\ INDEX(K))$

(2) Sei $\{C_1,\ldots,c_{m'}\}$ die durch die Äquivalenzrelation "redundant bzgl. Q_{SG}" induzierte Zerlegung von $\{1,..,m\}$, und es gelte $min(C_1)<\ldots<min(C_{m'})$. Damit sind die Abbildungen r und R durch das Folgende eindeutig definiert:

(i) $r : \{1,\ldots,m\} \longrightarrow \{1,\ldots,m'\}$
$r = \lambda i.\ i \varepsilon C_{i'} \longrightarrow i'$

(ii) $R : V \longrightarrow (\{0,\ldots,max(n_1,\ldots,n_m)\},POT\{1,\ldots,m'\}$
$R = \lambda K.\ [LEVEL(K),\{r(i) \mid i \varepsilon INDEX(K)\}]$

(3) G = (b,V',E',f',g',t') heißt (reduzierter und minimaler) Instruktionshypothesengraph (IH-Graph) von Q $\Leftrightarrow$
G ergibt sich aus Q_{SG}, indem man jeden Knoten $K \varepsilon V$ durch den Knoten R(K) ersetzt, d.h.:

(i) $V' = \{R(K) \mid K \varepsilon V\}$

(ii) $E' = \{(R(K_1),R(K_2)) \mid (K_1,K_2) \varepsilon E\}$

(iii) $f' = \lambda K.\ K = R(K') \longrightarrow f(K')$

(iv) $g' = \lambda(K_1,K_2).\ (K_1,K_2)=(R(K_1'),R(K_2')) \longrightarrow g(K_1',K_2')$

(v) $t' = \lambda(K_1,K_2).\ (K_1,K_2)=(R(K_1'),R(K_2')) \longrightarrow t(K_1',K_2')$

(4) Der durch obige Definitionen für $V \neq \emptyset$ eindeutig bestimmte Instruktionshypothesengraph eines PSP Q wird bezeichnet durch IH-GRAPH(Q). Gilt $V=\emptyset$, so ist IH-GRAPH(Q) = ε .

Wenn nicht ausdrücklich auf das Gegenteil hingewiesen wird, gelte für die folgenden Betrachtungen die Vereinbarung, daß bei der Annahme eines PSP Q immer ein endliches PSP gemeint ist, für das IH-GRAPH(Q) $\neq \varepsilon$ gilt, das somit nach Satz 3.5 realisierbar ist.

Beispiel 3.7

1. Aus Beispiel 3.3 geht hervor, daß bei Q_1 die Indizes 1 und 4, bei Q_2 2 und 4, bei Q_3 1 und 4 und bei Q_4 1 und 5 redundant sind.

2. IH-Graph G_3 von Q_3: Knotenmarkierung f(K) steht rechts von K; Kantenmarkierung g(K,K') wird durch die links, t(K,K') durch die rechts von (K,K') stehende Menge wiedergegeben. Dabei werden folgende Abkürzungen benützt:
A: = {CL(1),A(i,1),ZR(i)}
B: = {(CL(1),nil),(A(i,1),nil),(ZR(1),true),(ZR(2),false),
(ZR(4),false),(ZR(5),false}
C: = {CL(1),CL(3),A(i,1),A(i,3),ZR(i)}

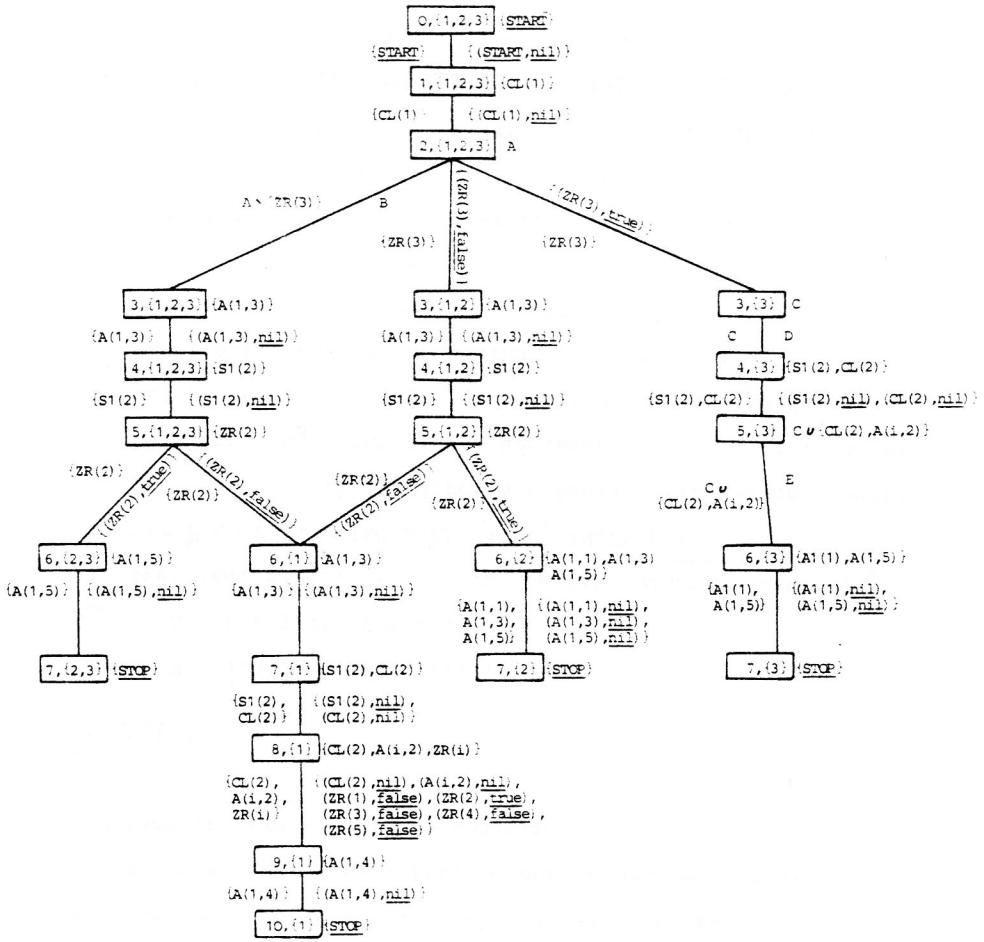


Abbildung 3.2: G_3 = IH-GRAPH(Q_3)

D: = {(CL(1),nil),(CL(3),nil),(A(i,1),nil),(A(i,3),nil), (ZR(1),true),(ZR(2),false),(ZR(3),true),(ZR(4),false), (ZR(5),false)}

E: = (D ∖ {(ZR(2),false)}) ∪ {(CL(2),nil),(A(i,2),nil), (ZR(2),true)}

Gemäß diesen Vereinbarungen ist G_3 in Abb. 3.2 angegeben.

3. G_4 = IH-GRAPH(Q_4) ergibt sich direkt aus der Graphendarstellung in Beispiel 3.3-4, indem in allen Knoten der redundante Index 5 entfernt wird.

Definition 3.9

Sei $Q \in PSP$, G = IH-GRAPH(Q), G = (b,V,E,f,g,t),
IHM $\subseteq$ INSTRUCTIONS.

(1) V ist linear geordnet durch die Relation $\prec_G$

(2) ROOT(G) heißt Wurzel von G und bezeichnet den eindeutig bestimmten Knoten $K \in V$ mit LEVEL(K) = 0.

(3) Sei $K \in V$, $I \in f(K)$, $I \neq$ STOP, $i \in$ INDEX(K).
$NF_G(K,I,i)$ heißt Nachfolger von K in G bzgl. Hypothese I und Index i und bezeichnet den eindeutig bestimmten Knoten $K' \in V$ mit:

$(K,K') \in E$, $I \in g(K,K')$, $i \in$ INDEX(K').

(4) Sei $K,K' \in V$.
K ist erreichbar in G durch einen IHM-Weg von K' <=>

$\exists i \in$ INDEX(K), $r \geq 1$, $K_1,\ldots,K_r \in V$ mit:

(i) $K_1 = K' \wedge K_r = K$

(ii) $\forall j \in \{1,\ldots,r-1\}. \exists I \in IHM \; K_{j+1} = NF_G(K_j,I,i)$

Beispiel 3.8

Für G = IH-GRAPH(Q_3) gilt z.B.

$[0,\{1,2,3\}]$ ist Wurzel von G.

$NF_G([2,\{1,2,3\}],ZR(1),1)$

$= NF_G([2,\{1,2,3\}],ZR(1),3) = [3,\{1,2,3\}]$

$NF_G([2,\{1,2,3\}],ZR(3),1) = [3,\{1,2\}]$

$NF_G([2,\{1,2,3\}],ZR(3),3) = [3,\{3\}]$

$NF_G([5,\{1,2\}],ZR(1),1)$ ist nicht definiert, da $ZR(1) \notin f([5,\{1,2\}])$

$NF_G([5,\{1,2\}],ZR(2),3)$ ist nicht definiert, da $3 \notin INDEX([5,\{1,2\}])$

Sei IM = {CL(1),ZR(2),A(1,3),S1(2)}.
$[7,\{1\}]$ und $[7,\{2,3\}]$ sind durch einen IM-Weg von $[0,\{1,2,3\}]$ erreichbar, nicht jedoch $[7,\{2\}]$ oder $[7,\{3\}]$, da beim Knoten $[2,\{1,2,3\}]$ nur Instruktion ZR(3) die dafür notwendigen Nachfolger auswählen kann.

3.2.5.2 Realisierungsgraph

Um auch für die Realisierung eines PSP Q durch ein Programm P eine dem IH-Graphen von Q analoge Darstellung zur Verfügung zu haben, werden in der Stufengraphdarstellung PRQ-SG(P,Q) die Indexmengen der Knoten entsprechend dem IH-Graphen reduziert.

Definition 3.10

Q sei PSP; P ε PROGRAMS realisiere Q, RPQ-SG(P,Q)=(V_p,E_p,a_p,c_p,u_p), G = IH-GRAPH(Q). Da Q realisierbar ist, ist nach Satz 3.5 G≠ε; sei R die für diesen Fall in Def. 3.8 benutzte Funktion.

R-GRAPH(P,Q) = $(V'_p,E'_p,a'_p,c'_p,u'_p)$ heißt Realisierungsgraph (R-Graph) von P und Q (mit gemäß G reduzierten Indexmengen)

<==>

(i) $V'_p = \{R(K) \mid K \in V_p\}$

(ii) $E'_p = \{(R(K_1),R(K_2)) \mid (K_1,K_2) \in E_p\}$

(iii) $a'_p = \lambda K.\ K = R(K') \longrightarrow a_p(K')$

(iv) $c'_p = \lambda(K_1,K_2).(K_1,K_2) = (R(K'_1),R(K'_2)) \longrightarrow c_p(K'_1,K'_2)$

(v) $u'_p = \lambda(K_1,K_2).(K_1,K_2) = (R(K'_1),R(K'_2)) \longrightarrow u_p(K'_1,K'_2)$

Beispiel 3.9

Entfernt man in den Graphendarstellungen aus Beipiel 3.4 in allen Knoten den Index 4, so erhält man die Realisierungsgraphen von P und Q_2 bzw. Q_3. Bei Q_3 ist zu beachten, daß Index 3, der hier zwar immer zusammen mit 2 vorkommt, nicht eliminiert werden kann, da in IH-GRAPH(Q_3) z.B. in den Knoten $[5,\{1,2\}]$ und $[5,\{3\}]$ 2 und 3 getrennt voneinander auftreten.

3.3 Ebenen der Hypothesenbildung

Die durch G = IH-GRAPH(Q) gegebene Darstellung eines PSP Q ermöglicht es, die Programmsynthese in zwei Ebenen aufzuspalten. Die eine Ebene (3.3.1) betrifft die Auswahl der möglichen Instruktionen; auf diese wird i.f. oft durch die Abkürzung IH (für Instruktionshypothesen oder I-Hypothesen) Bezug genommen. Die zweite Ebene (3.3.2) bezieht sich auf die Auswahl der LABEL, die zusammen mit den gewählten Instruktionen die Programmknoten des Zielprogramms bilden. In 3.3.3 werden die Beziehungen zu den bisher eingeführten Darstellungen aufgezeigt.

3.3.1 IH-Situationen

Um die Hypothesenbildung nicht unabhängig voneinander, sondern möglichst effizient miteinander gekoppelt vornehmen zu können, werden die Instruktionshypothesen nicht für alle Knoten von G gleichzeitig getroffen, sondern schrittweise bei der Wurzel von G beginnend bis hin zu den Endknoten von G. Die bis zu einem bestimmten Zeitpunkt angenommenen I-Hypothesen und die verbleibenden Möglichkeiten für die noch zu treffenden Entscheidungen werden in einer sog. IH-Situation festgehalten.

Eine IH-Situation IS ist ein Paar aus (A,R) und stellt eine Aufteilung von G in zwei Teilgraphen dar. Für die Knoten des Anfangsgraphen A ist durch die Auswahlfunktion a jeweils eine I-Hypothese aufgestellt worden; für die Knoten des Restgraphen R gibt f_R die möglichen I-Hypothesen an, aus denen noch jeweils eine ausgewählt werden muß.

Hierbei wird berücksichtigt, daß möglicherweise nicht mehr alle Instruktionen als mögliche Hypothesen zur Verfügung stehen, sondern nur noch eine Teilmenge IHM von INSTRUCTIONS. Bei der Definition von R ist in diesem Zusammenhang wichtig, daß nur noch die restlichen Knoten aus G aufgenommen werden, die für die durch A aufgestellten I-Hypothesen und die Einschränkung auf IHM noch relevant sind. Die Auswahl dieser Knoten verläuft ganz analog zu dem für die Erstellung der reduzierten Stufengraphdarstellung von MI benutzten Verfahren (Def. 3.5).

Die Folge der Knoten in R, die direkte Nachfolger der Endknoten von A sind, bilden die Front von IS, für deren Knoten die nächsten I-Hypothesen jeweils aufzustellen sind.

Def. 3.11

Q sei realisierbares PSP, G = IH-GRAPH(Q),

G = (b,V,E,f,g,t), IHM $\subseteq$ INSTRUCTIONS.

(1) $A = (V_A,E_A,a,c)$ ist ein aus G mit IH-Menge IHM ableitbarer IH-Anfangsgraph <=>

(i) $V_A \subseteq V$

$E_A = E \cap (V_A \times V_A)$

$a : V_A \longrightarrow IHM^1$

$c : E_A \longrightarrow TRANS'_b$

a heißt IH-Auswahlfunktion.

(ii) (V_A,E_A) ist ein Baum mit: $ROOT(V_A,E_A) = ROOT(G)$.

(iii) $\forall K \in V_A.\ a(K) \in f(K)$

(iv) $\forall (K,K') \in E_A,\ i \in INDEX(K').\ NF_G(K,a(K),i) = K'$.

(v) $c = \lambda(K,K').\ t^{a(K)}(K,K')$

(2) $FRONT_G(A) = (K_1,\ldots,K_r)$ heißt IH-Front von A <=>

(i) $\{K_1,\ldots,K_r\} = \{K \mid K \in V \setminus V_A, \exists K' \in V_A, i \in INDEX(K').\ K = NF_G(K',a(K'),i)\}$

(ii) $K_1 <_G \ldots <_G K_r$

(3) Sei A wie in (1), $(K_1,\ldots,K_r)$ wie in (2).

$FRONTE_G(A) = ((K'_1,K_1),\ldots,(K'_r,K_r))$ heißt Liste der Frontkanten von A <=>

$\forall i \in \{1,\ldots,r\}.\ (K'_i,K_i) \in E$

Bemerkung: i.f. wird der Index G meist weggelassen.

(4) Sei A wie in (1).

$(V_R, E_R, f_R, g_R, t_R)$ heißt gemäß IHM reduzierter IH-Restgraph von A <=>

sei:

(a) $V' = \{K \mid K \in V, \exists K' \in FRONT(A).$ K ist in G durch einen IHM^1-Weg erreichbar von $K'\}$

(b) $n = \max\{LEVEL(K) \mid K \in V'\} - \min\{LEVEL(K) \mid K \in V'\}$

Analog Def. 3.5 sei für $j \in \{0,\ldots,n\}$:

(c) $F_j : V' \longrightarrow POT(INSTRUCTIONS^1)$

$(j = 0)$ $F_0 = \lambda K.\ f(K) \setminus IHM^1$

$(j > 0)$ $F_j = \lambda K.\ F_{j-1}(K) \cup \bigcup_{K'} g(K,K')$

$(K,K') \in E \cap (V' \times V'), f(K') \setminus F_{j-1}(K') = \emptyset$

(d) $V'' = \{K \mid K \in V',\ f(K) \setminus F_n(K) \neq \emptyset\}$

Dann gilt:

(i) $V_R = \{K \mid K \in V'', \exists K' \in FRONT(A), \exists$ Weg in $E \cap (V'' \times V'')$ von K' nach $K\}$

(ii) $E_R = E \cap (V'' \times V'') \setminus \{(K,K') \mid (K,K') \in E \wedge g(K,K') \subseteq F_n(K)\}$

(iii) $f_R : V_R \longrightarrow POT(IHM^1)$

$f_R = \lambda K.\ f(K) \setminus F_n(K)$

(iv) $g_R : E_R \longrightarrow POT(IHM^1)$

$g_R = \lambda(K,K').\ g(K,K') \cap f_R(K)$

(v) $t_R : E_R \longrightarrow TRANS_b$

$t_R = \lambda(K,K').\ b \in \{s,a,as\} \longrightarrow t(K,K')$

$b = ss \longrightarrow t(K,K') \cap (g_R(K,K') \times TFN)$

(5) Sei A,R wie in (4), $FRONT_G(A) = (K_1,\ldots,K_r)$.

IS=(A,R) heißt aus G mit IH-Menge IHM ableitbare IH-Situation.

$FIH_G(IS) = M$ ist die Menge der möglichen Instruktionshypothesen an der IH-Front von A <=>

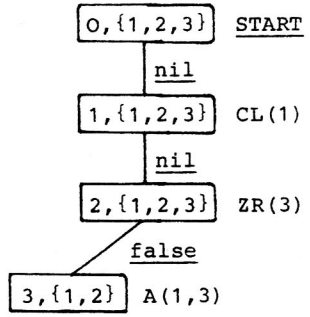
$$M = \begin{cases} f_R(K_1) \times \ldots \times f_R(K_r) & <=> \forall i \in \{1,\ldots,r\}.\ K_i \in V_R \\ \emptyset & <=> \exists i \in \{1,\ldots,r\}.\ K_i \notin V_R \end{cases}$$

INSTR (IS) $= \{a(K) \mid K \in V_A\} \setminus \{START, STOP\}$ heißt Instruktionenmenge des IH-Anfangsgraphen von IS

Beispiel 3.10

1. Sei IM = {CL(1),ZR(3),A(1,3),S1(2)}.

A_1: =



A_1 ist aus G_3 (Beispiel 3.7-2) mit IM ableitbarer IH-Anfangsgraph.

$FRONT_G(A_1) = ([4,\{1,2\}],[3,\{3\}])$

$FRONTE_G(A_1) = (([3,\{1,2\}],[4,\{1,2\}]),([2,\{1,2,3\}],[3,\{3\}]))$

Insbesondere gehört $[3,\{1,2,3\}]$ nicht zu $FRONT_G(A_1)$; es kann sogar keinen IH-Anfangsgraphen geben, zu dem sowohl $[3,\{1,2,3\}]$ als auch $[3,\{1,2\}]$ oder $[3,\{3\}]$ gehören, da die Auswahl einer Instruktion $a([2,\{1,2,3\}])$ aus der Menge $f([2,\{1,2,3\}])$ dies ausschließt.

Mit den Bezeichnungen von Def. 3.11, Abschnitt 4, wird nun der zugehörige Restgraph bestimmt. Danach gilt:

$V' = \{[4,\{1,2\}],[5,\{1,2\}],[3,\{3\}],[4,\{3\}],[5,\{3\}],[6,\{3\}]\}$

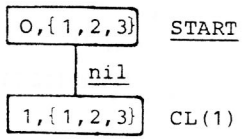
ist die Menge der durch einen IM-Weg von einem Frontknoten

aus erreichbaren Knoten; z.B. ist $[6,\{2\}]$ nicht erreichbar, da $f([5,\{1,2\}]) \cap IM = \emptyset$; und $[7,\{3\}]$ ist nicht erreichbar, da $f([6,\{3\}]) \cap IM = \emptyset$. Sei: $C = \{CL(1), CL(3), A(i,1), A(i,3), ZR(i)\}$, $A = C \setminus IM$. Damit gilt bei n = 3 :

K	$F_0(K)$	$F_1(K)$	$F_2(K)$	$F_3(K)$	$f(K) \setminus F_3(K)$
$[4,\{1,2\}]$	$\emptyset$	{S1(2)}	{S1(2)}	{S1(2)}	$\emptyset$
$[5,\{1,2\}]$	{ZR(2)}	{ZR(2)}	{ZR(2)}	{ZR(2)}	$\emptyset$
$[3,\{3\}]$	A	A	A	C	$\emptyset$
$[4,\{3\}]$	{CL(2)}	{CL(2)}	{S1(2), CL(2)}	{S1(2), CL(2)}	$\emptyset$
$[5,\{3\}]$	A ∪ {CL(2), A(i,2)}	C ∪ {CL(2), A(i,2)}	C ∪ {CL(2), A(i,2)}	C ∪ {CL(2), A(i,2)}	$\emptyset$
$[6,\{3\}]$	{A1(1), A(1,5)}	{A1(1), A(1,5)}	{A1(1), A(1,5)}	{A1(1), A(1,5)}	$\emptyset$

Der zu A_1 gehörige Restgraph ist also leer, und damit gilt auch $FIH_G(IS_1) = \emptyset$.

2. A_2 :=



ist ein mit INSTRUCTIONS aus G_3 ableitbarer Anfangsgraph, für den gilt: $FRONT(A_2) = ([2,\{1,2,3\}])$

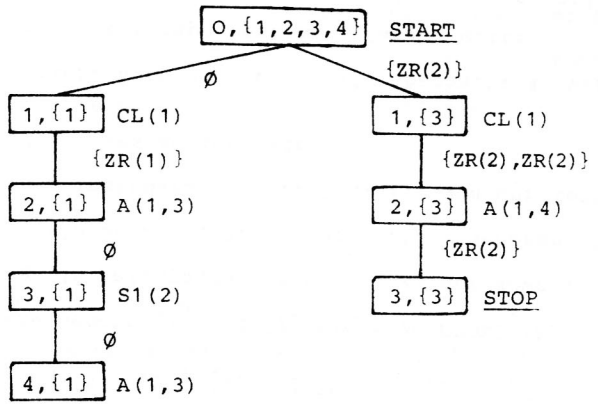
$FRONTE(A_2) = (([1,\{1,2,3\}],[2,\{1,2,3\}]))$

Der zugehörige Restgraph R_2 enthält alle übrigen Knoten aus G mit denselben Markierungen. Für $IS_2 = (A_2,R_2)$ gilt:

$FIH(IS_2) = f([2,\{1,2,3\}]) = \{CL(1), A(i,1), ZR(i)\}$

$INSTR(IS_2) = \{CL(1)\}$

3. A_6 :=



$IS_6 = (A_6,R_6)$ ist aus G_4 (Beispiel 3.7-3) mit INSTRUCTIONS ableitbare IH-Situation, wobei R_6 alle übrigen Knoten aus G_4 enthält.

Es gilt: $FRONT_{G_4}(A_6) = ([5,\{1\}],[1,\{2,4\}])$

$FRONTE_{G_4}(A_6) = (([4,\{1,\}],[5,\{1\}]),([0,\{1,2,3,4\}],[1,\{2,4\}]))$

$FIH_{G_4}(IS_6) = \{S1(2), CL(2)\} \times \{CL(1), A(i,2)\}$

$INSTR(IS_6) = \{S1(2), CL(1), A(1,3), A(1,4)\}$

Die Definition einer IH-Situation könnte vereinfacht werden, wenn man den Fall b = ss nicht berücksichtigen würde. Sie bestünde dann i.w. in der Angabe einer Zerlegung von V in V_A und V_R, c und t_R wären die Einschränkungen von t auf V_A und V_R, g wäre schon in G nicht mehr notwendig und könnte mit g_R auch in R entfallen, und zur Bildung von f_R würde der Mengendurchschnitt von f mit IHM^1 genommen.

Eine naheliegende Nachfolgerrelation zwischen IH-Situationen IS und IS' ist die, daß IS' aus IS entsteht, wenn man alle in IS angenommenen I-Hypothesen nach IS' übernimmt und eventuell weitere hinzufügt. Drei derartige Nachfolgerrelationen sowie

zwei Kriterien für kleinste (initiale) und größte (terminale) IH-Situationen unter diesen Ordnungen werden in der folgenden Situation eingeführt und einige ihrer Eigenschaften in dem daran anschließenden Satz gezeigt.

Definition 3.12

Sei $Q \in PSP$. Für $i \in \{1,2\}$ sei $IHM_i \subseteq$ INSTRUCTIONS,
IS_i aus G=IH-GRAPH(Q) mit IHM_i ableitbare IH-Situation,
$IS_i = (A_i, R_i)$, $A_i = (V_A^i, E_A^i, a_i, c_i)$, $R_i = (V_R^i, E_R^i, f_i, g_i, t_i)$.
Sei $INSTR(A_1) \notin IHM_2$.

(1) IS_2 ist Nachfolger von IS_1 <=>
  (i) $IHM_1 \supseteq IHM_2$
  (ii) $V_A^1 \subseteq V_A^2$
  (iii) $a_1 = a_{2/V_A^1}$

(2) IS_2 ist größter erzwungener Nachfolger von IS_1 gemäß IHM_2 <=>
  (i) IS_2 ist Nachfolger von IS_1
  (ii) $\forall K \in V_A^2 \setminus V_A^1 \; . \; f_1(K) \cap IHM_2^1 = \{a_2(K)\}$
  (iii) $\forall K \in FRONT(A_2) \; . \; |f_2(K)| \neq 1$

(3) Sei $FRONT(A_1) = (K_1,\ldots,K_r)$, $H = (h_1,\ldots,h_r)$ mögliche Instruktionshypothese an der IH-Front von A_1.
IS_2 ist direkter Nachfolger von IS_1 gemäß Hypothese H <=>
  (i) $IHM_1 = IHM_2$
  (ii) IS_2 ist Nachfolger von IS_1
  (iii) $V_A^2 = V_A^1 \cup \{K_1,\ldots,K_r\}$
  (vi) $\forall j \in \{1,\ldots,r\} \; . \; a_2(K_j) = h_j$

(4) IS_1 heißt initiale IH-Situation von G <=>
  (i) IHM_1 = INSTRUCTIONS
  (ii) $V_A^1 = \{ROOT(G)\}$

(5) IS_1 heißt terminale IH-Situation von G <=>
  $FRONT_G(A_1) = \varepsilon$

Satz 3.7 Es seien die Bezeichnungen aus Def. 3.12 gegeben.

Behauptung:

(1) IS,IS' sind initiale IH-Situationen von G => IS = IS'
  Notation: Die damit eindeutig bestimmte initiale IH-Situation von G wird i.f. bezeichnet durch IH-INIT(G).

(2) IS_2, IS_2' sind größte erzwungene Nachfolger von IS_1 gemäß IHM_2 => $IS_2 = IS_2'$
  Notation: GSUC(IS_1,IHM_2) bezeichnet den eindeutig bestimmten größten erzwungenen Nachfolger von IS_1 gemäß IHM_2.

(3) IS_2, IS_2' sind direkte Nachfolger von IS_1 gemäß Hypothese H => $IS_2 = IS_2'$
  Notation: DSUC(IS_1,H) bezeichnet den eindeutig bestimmten direkten Nachfolger von IS_1 gemäß Hypothese H.

Eigenschaften von IH-INIT, GSUC und DSUC:

(4) IS ist aus G ableitbare IH-Situation => IS Nachfolger von IH-INIT(G)

(5) IS_1 nicht-terminale IH-Situation $\wedge$ IS_2 terminaler Nachfolger von IS_1

=> $\exists$ H ε FIH(IS_1). IS_2 Nachfolger von DSUC(IS_1,H)

(6) IS_2 terminaler Nachfolger von IS_1 $\wedge$ INSTR(IS_2) $\subseteq$ IHM

=> IS_2 terminaler Nachfolger von GSUC(IS_1,IHM)

(7) IS_1 besitzt keinen terminalen Nachfolger <=> IS_1 ist nicht-terminal $\wedge$ FIH(IS_1) = $\emptyset$.

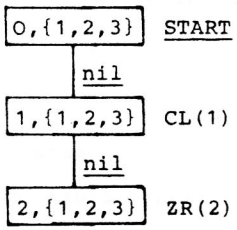
(8) $\forall$H ε FIH(IS_1)($\exists IS_2$. IS_2 ist terminaler Nachfolger von DSUC(IS_1,H))

Beweis:

Aussagen (1) - (6) ergeben sich direkt aus den entsprechenden Definitionen, (7) und (8) werden analog zu dem Vorgehen in den Sätzen 3.4 und 3.5 bewiesen.

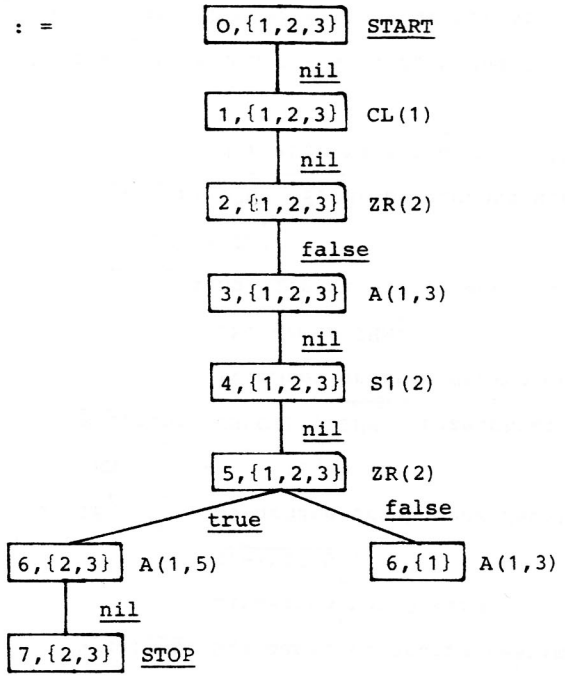
Beispiel 3.11

1. IS_3 = (A_3,R_3) = DSUC(IS_2,(ZR(2))), wobei IS_2 die im vorigen Beispiel 3.10-2 gegebene IH-Situation ist und A_3 skizziert wird durch:



2. IS_4 = (A_4,R_4) = GSUC(IS_3,INSTRUCTIONS) mit:

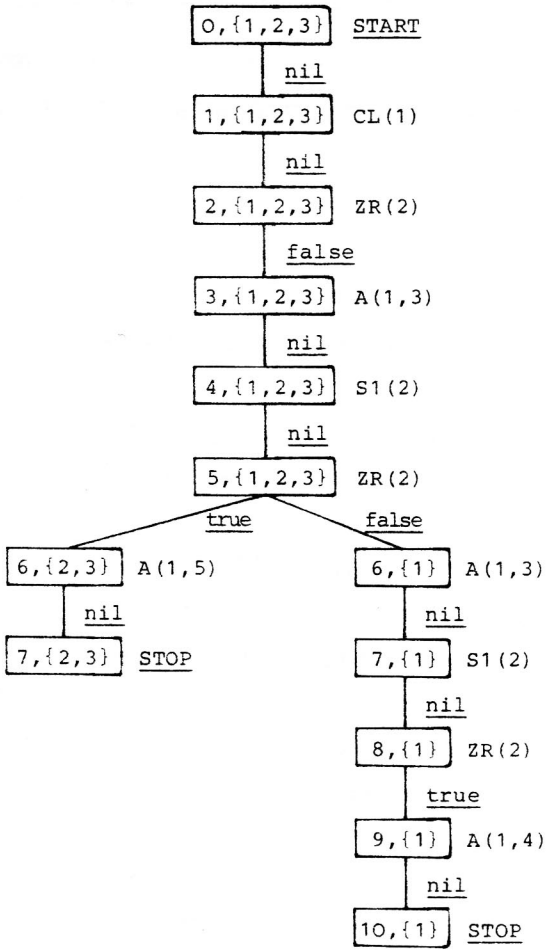
A_4 : =



3. $IS_5 = (A_5, R_5) = GSUC(IS_3, IM)$ mit:

IM := {A(1,3), CL(1), S1(2), A(1,5), ZR(2)}

A_5 :=



R_5 ist leer, IS_5 ist terminale IH-Situation.

4. Für IS_6 (Beispiel 3.10-3) gilt:

$IS_6 = GSUC(IH\text{-}INIT(G_4), INSTRUCTIONS)$

3.3.2 Markierungen für IH-Situationen

Auf der zweiten Ebene der Hypothesenbildung (auf die i.f. meist durch die Bezeichner u oder U hingewiesen wird) werden die LABEL der Programmknoten des Zielprogramms erstellt. Die wesentliche Eigenschaft einer Hypothese u über die möglichen LABEL ist die, daß analog zur Eindeutigkeitseigenschaft von Programmen die durch u und einen IH-Anfangsgraphen A definierte Tripelmenge U-GRAPH(A,u) nur eindeutige Transitionen enthält.

Definition 3.13

Q sei PSP, IS = (A,R) aus IH-GRAPH(Q) ableitbare IH-Situation, $A = (V_A, E_A, a, c)$. u sei eine Abbildung $u : V_A \longrightarrow \mathbb{N}$;
IM $\subseteq$ INSTRUCTIONS, I $\in$ INSTRUCTIONS.

(1) U-GRAPH(A,u) = $\{((u(K), a(K)), c(K,K'), (u(K'), a(K'))) \mid (K,K') \in E_A\}$

(2) u heißt eindeutige Markierung für A <=>

(2.1) $\forall$ (X,c,Y), (X,c',Y') $\in$ U-GRAPH(A,u). (c=c' => Y = Y')

(2.2) $\forall K \in V_A$. a(K) $\in$ {START, STOP} => u(K) = 1

(3) Die Menge der U-Knoten von A und u ist definiert durch:

(3.1) U-NODES(A,u,I) = $\{(u(K), a(K)) \mid K \in V_A, a(K) = I\}$

(3,2) U-NODES(A,u,IM) = $\{(u(K), a(K)) \mid K \in V_A, a(K) \in IM\}$

(3.3) U-NODES(A,u) = U-NODES(A,u,INSTRUCTIONS)

(4) Die Anzahl der U-Knoten von A und u ist analog definiert durch:

(4.1) UL(A,u,I) = |U-NODES(A,u,I)|

(4.2) UL(A,u,IM) = |U-NODES(A,u,IM)|

(4.3) UL(A,u) = |U-NODES(A,u)|

(5) UM = (UG,u) heißt eindeutiges Markierungspaar von A <=>

u ist eindeutige Markierung für A $\wedge$ UG = U-GRAPH(A,u)

oder: u = λK. 1 $\wedge$ UG = $\emptyset$

(6) U-DUMMY = ($\emptyset$,λK.1) heißt triviales Markierungspaar.

Beispiel 3.12

Mit den Bezeichnungen des vorigen Beispiels 3.11 gilt:

1. u = λK. 1 ist eindeutige Markierung für A_4.

   U-GRAPH(A_4,u) = {((1,START),nil,(1,CL(1))),((CL(1)),nil,(1,ZR(2))),
   ((1,ZR(2)),false,(1,A(1,3)),((1,A(1,3)),nil,(1,S1(2))),
   ((1,S1(2)),nil,(1,ZR(2))),((1,ZR(2)),true,(1,A(1,5))),
   ((1,A(1,5)),nil,(1,STOP))}

   UL(A_4,u) = 5

2. u = λK. 1 ist keine eindeutige Markierung für A_5, da wegen der Kanten ([5,{1,2,3}],[6,{1,2}]) und ([8,{1}],[9,{1}]) in A_5 gilt:

   ((1,ZR(2)),true,(1,A(1,5))) ε U-GRAPH(A_5,u)

   ((1,ZR(2)),true,(1,A(1,4))) ε U-GRAPH(A_5,u)

   Es gilt: UL(A_5,u) = 6

3.3.3 Beziehungen zu Standarddarstellungen und Realisierungsgraphen

Durch die Bildung des IH-Graphen G von Q = (b,M) wurde es möglich, die Menge der IH-Situationen zu definieren. In der Standarddarstellung Q_{ST} sind alle durch M gegebenen Folgen eindeutig identifizierbar. Es wird gezeigt, daß dies auch für alle aus G ableitbaren terminalen IH-Situationen gilt.

Definition 3.14

Q sei PSP, dim(Q) = (m,(n_1,...,n_m)). IS = (A,R) sei aus G = IH-GRAPH(Q) ableitbare terminale IH-Situation, A = (V_A,E_A,a,c). r sei die in Definition 3.8 benützte Abbildung r : {1,...,m} —> INDEX(ROOT(G)).

Sei iε{1,...,m}.

PATH(A,i) = (K_o,...,$K_{n_i'}$) heißt i-Pfad in A <=>

(i) K_o = ROOT(G)

(ii) $\forall$ jε{1,...,n_i'}. $K_j \varepsilon V_A \wedge r(i) \varepsilon$ INDEX(K_j) $\wedge$ (K_{j-1},K_j) εE_A

(iii) $\forall K \varepsilon V_A$. ($K_{n_i'}$,K) $\notin E_A$

Beispiel 3.13

In A_5 aus Beispiel 3.11 gilt:

PATH(A_5,2) = ([0,{1,2,3}],[1,{1,2,3}],[2,{1,2,3}],[3,{1,2,3}],
[4,{1,2,3}],[5,{1,2,3}],[6,{2,3}],[7,{2,3}])

Da Indizes 1 und 4 redundant sind (Beispiel 3.7-1), gilt ausserdem z.B. PATH(A_5,4) = PATH(A_5,1).

Der i-Pfad in einer terminalen IH-Situation IS = (A,R) ist eindeutig bestimmt. Der folgende Satz drückt aus, daß er unter den durch A aufgestellten I-Hypothesen (für jede terminale IH-Situation) gerade die i-te Beispielsfolge des PSP Q 'erklärt' und darüberhinaus die zugehörigen b'-Transitionen der Standarddarstellung von Q entsprechend korrekt wiedergibt.

Satz 3.8

Q sei PSP, $dim(Q) = (m,(n_1,\ldots,n_m))$. IS = (A,R) sei aus G = IH-GRAPH(Q) ableitbare terminale IH-Situation, $A = (V_A,E_A,a,c)$. Q_{ST} = (b,S,MI,MT) sei das standardisierte PSP von Q mit den Bezeichnungen von Definition 3.3.

Sei $i \in \{1,\ldots,m\}$, $PATH(A,i) = (K_{i,0},\ldots,K_{i,n_i'})$.

Behauptung:

(1) $n_i' = n_i$

(2) $(a(K_{i,0}) = \underline{START}) \wedge (a(K_{i,n_i}) = \underline{STOP})$
$(\forall j \in \{0,\ldots,n_i\}.\ a(K_{i,j}) \in MI_{i,j})$

(3) Für $j \in \{1,\ldots,n_i\}$ sei:
$(j = 1)\quad s'_{i,1} = st_i$
$(j > 1)\quad s'_{i,j} = \delta(a(K_{i,j-1}),s_{i,j-1})$
dann gilt:
$\forall j \in \{1,\ldots,n_i\}.\ s'_{i,j} = s_{i,j}$

(4) $\forall j \in \{0,\ldots,n_i-1\}.\ (K_{i,j},K_{i,j+1}) \in E_A \wedge$
$c(K_{i,j},K_{i,j+1}) = MT_{i,j}^{a(K_{i,j})}$

(5) $b \in \{\underline{s},\underline{ss}\} \Rightarrow c(K_{i,0},K_{i,1}) = \underline{nil} \wedge$
$\forall j \in \{1,\ldots,n_i-1\}.\ c(K_{i,j},K_{i,j+1}) = \gamma(a(K_{i,j}),s_{i,j})$

(6) $b \in \{\underline{a},\underline{as}\} \Rightarrow \forall j \in \{0,\ldots,n_i-1\}.\ c(K_{i,j},K_{i,j+1}) = \bar{\gamma}(s_{i,j+1},\underline{true})$

Beweis: folgt mit Hilfe der Definitionen 3.3 (standardisiertes PSP), 3.8 (IH-GRAPH), 3.11 (IH-Situation) und 3.14 (PATH).

Für die Stufengraphdarstellung eines PSP Q war gezeigt worden, daß sie alle derartigen Darstellungen von Realisierungen von Q 'umfaßt' (Satz 3.4). Hier wird nun diese Beziehung zwischen Instruktionshypothesen- und Realisierungsgraph präzisiert. Jede Realisierung von Q definiert auf eindeutige Weise eine aus G ableitbare terminale IH-Situation IS = (A,R) und darüberhinaus eine eindeutige Markierung für A.

Satz 3.9

Q = (b,M) sei PSP; P ∈ PROGRAMS realisiere Q,
G = IH-GRAPH(Q), R-GRAPH(P,Q) = (V,E,a,c,u), A = (V,E,a,c).

Behauptung:

(1) $(A,(\emptyset,\ldots,\emptyset))$ ist aus G ableitbare terminale IH-Situation

(2) u ist eindeutige Markierung für A

(3) $UL(A,u) = L_b(P)$

Beweis:

(1) folgt aus Satz 3.4 und Def. 3.8 und 3.10

(2) folgt mit Hilfe der Eindeutigkeit von P und den Sätzen 3.1 und 3.2

(3) Definition von L_b (Abschnitt 2.2) und Def. 3.4 und 3.13.

Beispiel 3.14

Das Beispielprogramm P (Beispiel 2.2) realisiert Q_3 (Beispiel 2.3-3). Entfernt man in der Graphendarstellung in Beispiel 3.4.-2 in allen Knoten den redundanten Index 4, so erhält man den Realisierungsgraphen von P und Q_3. Dieser definiert ge-

rade die terminale IH-Situation $IS_5 = (A_5,R_5)$ aus Beispiel 3.11-3. Für einen Knoten K gebe u(K) gerade das LABEL wieder, das im R-Graphen angegeben ist; dann gilt:

(i) u ist eindeutige Markierung für A_5

(ii) U-GRAPH(A_5,u) = P

Als "Umkehrung" zu Satz 3.9 wird später gezeigt werden, daß für jedes terminale IS' = (A',R') mit eindeutiger Markierung u' ein Programm P' existiert, das Q realisiert, sowohl die durch A' aufgestellten I-Hypothesen als auch die durch u' gegebenen Hypothesen bzgl. der LABEL der Programmknoten benützt und für das $L_b(P') = UL(A',u')$ gilt.

Damit ist die Programmsynthese auf folgende Problemstellung reduziert worden: suche eine terminale IH-Situation IS = (A,R) mit zugehöriger eindeutiger Markierung u, sodaß UL(A,u) minimal ist.

Kapitel 4 IN PA VERWANDTE STRATEGIEN UND METHODEN

4.1 Überblick über den PSA PA

4.1.1 Der Suchraum von PA

Der Suchraum von PA wird durch den Instruktionshypothesengraph G eines PSP Q bestimmt. Bezeichne SR(L) alle Paare (IS,u), wobei IS = (A,R) eine aus G ableitbare terminale IH-Situation, u eine eindeutige Markierung für A und UL(A,u) = L ist. Die am Ende des vorigen Kapitels formulierte Problemstellung läßt sich damit ausdrücken durch: finde ein $L_z \in \mathbb{N}$, sodaß SR(1),SR(2),... ...,$SR(L_z-1)$ leer sind, und ein Paar (IS,u) aus $SR(L_z)$.

4.1.2 Vorgehensweise von PA

Zu einem gegebenen PSP Q wird der entsprechende IH-Graph G gebildet und nach dem in Satz 3.5 angegebenen Kriterium die Realisierbarkeit von Q überprüft. Ist G = ε, so ist Q nicht realisierbar und der Algorithmus bricht an dieser Stelle ab.

Sei also G ≠ ε. Zunächst wird eine untere Grenze LMIN bestimmt, für die SR(LMIN-1) = ∅ gilt. Offensichtlich folgt aus SR(L) = ∅ auch SR(L-1) = ∅; ist LMIN bestimmt, so sind SR(1),SR(2),...,SR(LMIN-1) für die Programmsynthese folglich nicht mehr relevant, da diese Suchräume alle leer sind. Der Suchraumindex L wird mit LMIN initialisiert.

Ausgehend von der initialen IH-Situation IS_1 von G werden immer weitere Instruktionshypothesen aufgestellt, wobei die Front der sich ergebenden Nachfolgersituationen $IS_2, IS_3, \ldots$ immer weiter hin zu den Endknoten von G (i.f. bezeichnet als "nach rechts") verschoben wird. Für jede der $IS_i = (A_i, R_i)$ wird dabei überprüft, ob es überhaupt einen terminalen Nachfolger $IS_i' = (A_i', R_i')$ geben kann, sodaß für irgendein u (IS_i', u) in SR(L) ist. Sobald feststeht, daß dies nicht der Fall ist, wird an der aktuellen Front eine andere I-Hypothese aufgestellt und das Verfahren fortgesetzt. Gibt es an einer Front keine alternative I-Hypothese mehr, so erfolgt eine Zurücksetzung der aktuellen Front zur vorherigen (die Front wird "nach links" verschoben). Ist dies nicht mehr möglich, da bereits die am weitesten links stehende Front erreicht ist, so ist bei aktuellem Suchraumindex L SR(L) leer und L wird um 1 inkrementiert. Der Algorithmus bricht ab, sobald die aktuelle Front die Endknoten von G erreicht hat und eine Markierung u gefunden worden ist, sodaß UL(A,u) = L ist. Für den Fall $b \in \{\underline{a}, \underline{as}\}$ müssen allerdings die Transitionen, die aus Mengen von Tests bestehen, noch umgesetzt werden in Programmtransitionen; für $b \in \{\underline{s}, \underline{ss}\}$ erübrigt sich dieser Schritt. Das erhaltene Programm P realisiert Q und ist diesbezüglich minimal.

4.2 LB-Situationen

In 4.2.1 werden LB-Situationen definiert, in 4.2.2 ein Satz angegeben, der ein Verfahren zu ihrer Berechnung liefert, und in 4.2.3 wird gezeigt, inwiefern LB-Situationen ein monotones Maß für IH-Situationen sind.

4.2.1 Definition von LB-Situationen

Ein entscheidender Punkt der in 4.1.2 beschriebenen Vorgehensweise ist der, ob es für eine IH-Situation IS = (A,R) einen terminalen Nachfolger IS' = (A',R') und dafür eine LABEL-Hypothese u gibt, sodaß UL(A',u) = L.

Ist IS selbst schon terminal, so hängt dieses Problem nur von den möglichen LABEL-Markierungen u für A ab. Die Zahl der zu überprüfenden LABEL-Hypothesen kann erheblich reduziert werden, wenn für Knoten K und K' aus A von vorneherein festgestellt werden kann, ob sie gleiche LABEL haben können oder nicht. Daraus ließe sich auch eine Mindestanzahl der benötigten Knoten errechnen.

Ist IS nicht terminal, so hängt die Frage nach möglicher gleicher LABEL-Markierung von K und K' wesentlich von den bereits aufgestellten und den noch anzunehmenden I-Hypothesen ab. Ausserdem ergibt sich die folgende Schwierigkeit: während bei $b \in \{\underline{s}, \underline{a}, \underline{as}\}$ die Vereinigung der Knotenmengen V_A und V_R von A und R immer die Knotenmenge V_G von G ergibt, ist dies bei $b = \underline{ss}$ i.a. nicht der Fall (vgl. G_3 in Beispiel 3.7 und IS_4 in Beispiel 3.11).

Eine Lösung dieses Problems ergibt sich durch die Bildung des sog. IH-Baumes T von IS.

Für $b \in \{\underline{a}, \underline{s}, \underline{as}\}$ ist T nichts anderes als die Zusammenfassung von A und R zu einem einzigen Graphen gemäß den durch FRONTE(A) gegebenen Kanten, ist also von der Knotenstruktur her mit G

identisch. Im Unterschied zu G sind die nicht gewählten, links von der Front von IS liegenden I-Hypothesen in T entfernt, und rechts von der Front sind nur noch die I-Hypothesenmengen vorhanden, wie sie durch R vorgegeben sind.

Für b = ss wird R überführt in R', das als eine Folge von Bäumen $(T_1,\ldots,T_r)$ angesehen werden kann, die jeweils den von $(K_1,\ldots,K_r)$ = FRONT(A) erreichbaren Knoten in R entsprechen. Für die Frontknoten $K_1,\ldots,K_r$ sind INDEX(K_i) und INDEX(K_j) für $i\neq j$ paarweise disjunkt und damit auch die von K_i bzw. K_j erreichbaren Knotenmengen.

Für jeden der nicht mit {STOP} markierten Knoten K aus R bilden die Indexmengen der direkten Nachfolger K' von K, die mit jeweils gleicher Kantenmarkierung g(K,K') erreichbar sind, eine Zerlegung von INDEX(K). Seien $Z_1,\ldots,Z_s$ die verschiedenen Zerlegungen von INDEX(K), dann sind die Nachfolger von K in der transformierten Form R' von R definiert durch diejenige Zerlegung Z von INDEX(K), die Verfeinerung aller Z_i ist und darüberhinaus die gröbste Zerlegung, die diese Bedingung erfüllt, wobei sich dieser Prozeß von der Stufe mit kleinstem Index bis hin zur höchsten Stufe fortsetzt. Für einen Knoten $K = [j,\{i_1,\ldots,i_t\}]$ aus R' gilt also in der Terminologie von 3.2.1.: unter allen mit A beginnenden I-Hypothesen werden die Elemente $MI_{i_1,j},\ldots,MI_{i_t,j}$ zusammengefaßt, d.h. in allen Q realisierenden Programmen P, die die durch A bereits vorgegebenen I-Hypothesen benützen, werden die den Indizes $(i_1,j),\ldots,(i_t,j)$ entsprechenden Speicherbelegungsübergänge durch einen einzigen Programmknoten erzeugt. Der IH-Baum von IS ent-

steht nun dadurch, daß man die Bäume $T_1,\ldots,T_r$, deren Wurzeln ja gerade die Frontknoten sind, an die Endknoten von A 'anhängt' und die Markierungsfunktionen entsprechend G vereinheitlicht.

Definition 4.1

Q = (b,M) sei realisierbares PSP, IS = (A,R) sei aus IH-GRAPH(Q mit IHM ⊆ INSTRUCTIONS ableitbare IH-Situation, $A = (V_A,E_A,a,c)$ $R = (V_R,E_R,f_R,g_R,t_R)$.

(1) (V'_R,E'_R,f'_R,t'_R) heißt transformierter (vereinfachter) Restgraph von IS <=>

(i) $b\in\{a,s,as\}$ => $(V'_R,E'_R,f'_R,t'_R) = (V_R,E_R,f_R,t_R)$

(ii) b = ss =>

sei: (a) minl = min{LEVEL(K) | $K\in V_R$}

(b) maxl = max{LEVEL(K) | $K\in V_R$}

(c) für $l\in\{minl,minl+1,\ldots,maxl\}$ sei:

(c1) $V_l = \{K \mid K\in V_R,\ LEVEL(K) = l\}$

(c2) $IND_l = \bigcup_{K\in V_l} INDEX(K)$

(c3) $\{IND_l^1,\ldots,IND_l^{r_l}\}$ die durch $\equiv_l$:

$i\equiv_l j$ <=> ($\forall K\in V_l . i\in INDEX(K) <=> j\in INDEX(K)$)

induzierte Zerlegung von IND_l

(c4) $V'_l = \{[l,IND_l^i] \mid i\in\{1,\ldots,r_l\}\}$

dann gilt:

(ii.1) $V'_R = V'_{minl} \cup \ldots \cup V'_{maxl}$

(ii.2) $E'_R = \{(K,K') \mid K,K'\in V'_R, LEVEL(K') = LEVEL(K)+1,$ $INDEX(K') \cap INDEX(K) \neq \emptyset\}$

(ii.3) $f'_R : V'_R \longrightarrow POT(IHM)$

$$f'_R = \lambda K. \bigcup_{K' \in V_{LEVEL(K)}} f_R(K')$$

$$INDEX(K) \cap INDEX(K') \neq \emptyset$$

(ii.4) $t'_R : E'_R \longrightarrow TRANS_b$

$$t'_R = \lambda(K,K'). \bigcup_{(K_1,K_2) \in E_R} t_R(K_1,K_2)$$

$$INDEX(K') \cap INDEX(K_2) \neq \emptyset$$

(2) Sei (V'_R,E'_R,f'_R,t'_R) der vereinfachte Restgraph von IS.

$T = (V_T,E_T,f_T,t_T)$ heißt IH-Baum von IS <=>

(i) $V_T = V_A \cup V'_R$

(ii) $E_T = E_A \cup FRONTE(A) \cup E'_R$

(iii) $f_T : V_T \longrightarrow POT(IHM)$

$f_T = \lambda K.\ K \in V_A \longrightarrow \{a(K)\}$

$K \in V'_R \longrightarrow f'_R(K)$

(iv) $t_T : E_T \longrightarrow TRANS_b$

$t_T = \lambda(K,K').\ K \in V_A \longrightarrow (b = ss \longrightarrow \{(a(K),c(K,K')\},$

$b \in \{s,a,as\} \longrightarrow c(K,K'))$

$K \in V'_R \longrightarrow t'_R(K,K')$

(3) Für eine IH-Situation IS bezeichnet IH-TREE (IS) den durch (1) und (2) eindeutig bestimmten IH-Baum von IS.

(4) Sei $T = IH\text{-}TREE(IS), K \in V_T, i \in INDEX(K)$.

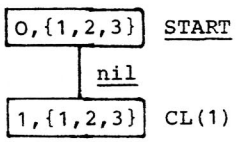
$NF_T(K,i)$ heißt Nachfolger von K in T bezgl. Index i und bezeichnet den eindeutig (falls K' überhaupt existiert) bestimmten Knoten $K' \in V_T$ mit:

$$(K,K') \in E_T \wedge i \in INDEX(K')$$

Beispiel 4.1

1. Sei $IS_2 = (A_2,R_2)$ die in Beispiel 3.10-2 angegebene IH-Situation mit:

$A_2 =$



Mit den Bezeichnungen von Def. 4.1 gilt dann:

minl = 2

maxl = 10

für l = 2 : $IND_1 = \{1,2,3\}$

$V'_1 = \{[1,\{1,2,3\}]\}$

für l = 3,4,5 : $IND_1 = \{1,2,3\}$

$V'_1 = \{[1,\{1,2\}],[1,\{3\}]\}$

für l = 6,7 : $IND_1 = \{1,2,3\}$

$V'_1 = \{[1,\{1\}],[1,\{2\}],[1,\{3\}]\}$

für l = 8,9,10 : $IND_1 = \{1\}$

$V'_1 = \{[1,\{1\}]\}$

Im anschließend wiedergegebenen IH-TREE(IS_2) gelten folgende Vereinbarungen:

A:= {CL(1),A(i,1)ZR(i)}

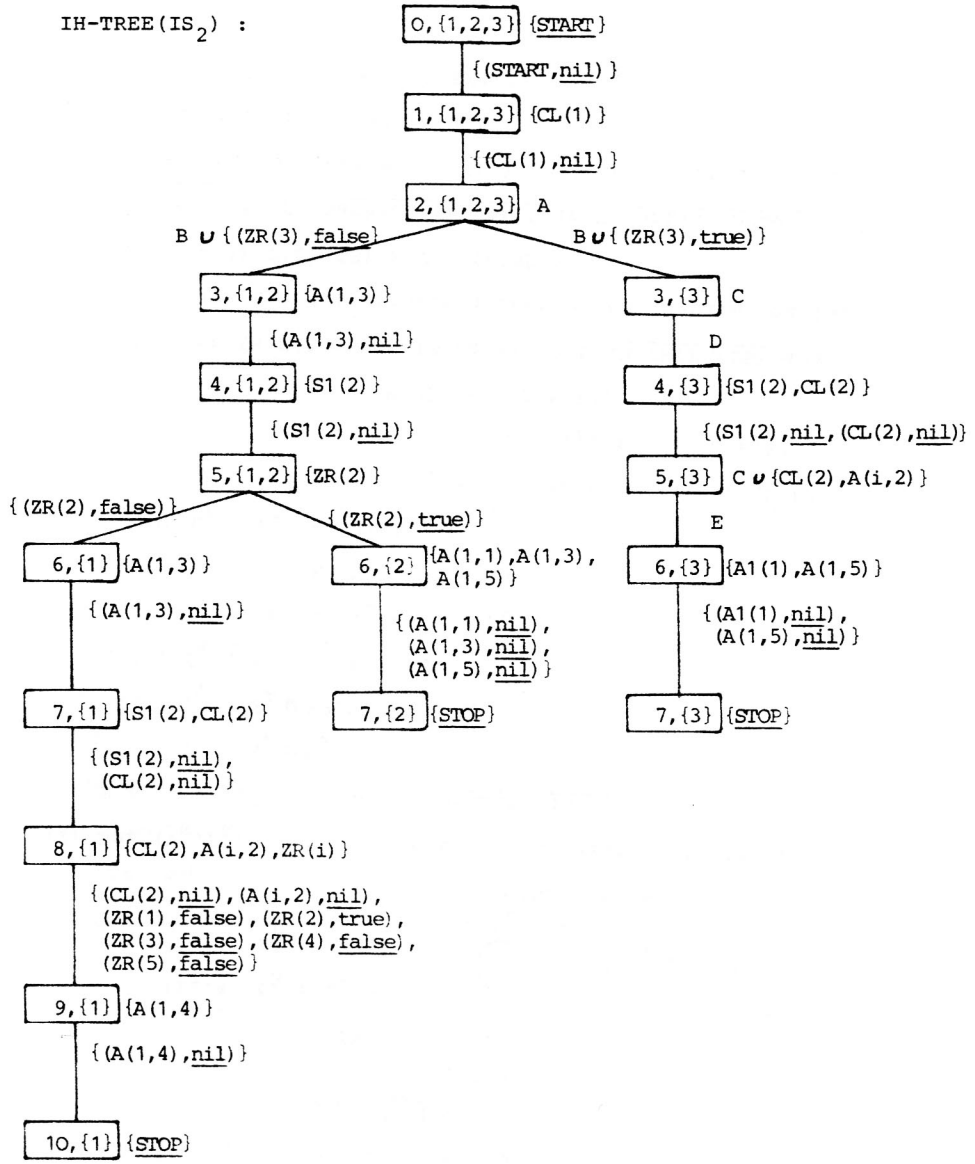
B:= {(CL(1),nil),(A(i,1),nil),(ZR(1),true),(ZR(2),false),
(ZR(4),false),(ZR(5),false)}

C:= {CL(1),CL(3),A(i,1),A(i,3),ZR(i)}

D:= {(CL(1),nil),(CL(3),nil),(A(i,1),nil),(A(i,3),nil),
(ZR(1),true),(ZR(2),false),(ZR(3),true),(ZR(4),false),
(ZR(5),false)}

E:= (D \ {(ZR(2),false)}) ∪ {(CL(2),nil),(A(i,2),nil),
(ZR(2),true)}

IH-TREE(IS_2) :



$NF_T([2,\{1,2,3\}], 2) = [3,\{1,2\}]$

$NF_T([2,\{1,2,3\}], 3) = [3,\{3\}]$

2. Für die in Beispiel 3.10-3 angegebene IH-Situation IS_6 gilt: IH-TREE(IS_6) ist gleich dem IH-Graphen von Q_4 (Beispiel 3.3-4).

In einem IH-Baum T heißen zwei Knoten K,K' unterscheidbar, wenn es kein Q realisierendes Programm P gibt, das die K und K' entsprechenden Speicherbelegungsübergänge durch denselben Programmknoten erzeugt. Das ist sicherlich dann der Fall, wenn es keine gemeinsame I-Hypothese für K und K' gibt, aber auch dann, wenn alle gemeinsamen I-Hypothesen bei gleicher LABEL-Hypothese zum Widerspruch führen; dies wird in der folgenden Definition präzisiert.

Definition 4.2

Q sei PSP, IS = (A,R) aus IH-GRAPH(Q) ableitbare IH-Situation, T = IH-TREE(IS), T = (V,E,f,t), V' = $V \setminus \{K \mid K \in V, f(K) \subseteq \{START, STOP\}\}$.

(1) Sei $K_1, K_2 \in V'$, $n \geq 1$.

(i) $K_1 \sharp_0 K_2 \iff f(K_1) \cap f(K_2) = \emptyset$

$K_1 \sharp_n K_2 \iff \forall I \in f(K_1) \cap f(K_2)$.

$\exists i_1 \in INDEX(K_1), i_2 \in INDEX(K_2)$.

$(t^I(K_1, NF_T(K_1,i_1)) = t^I(K_2, NF_T(K_2,i_2)))$

$\wedge (NF_T(K_1,i_1) \sharp_{n-1} NF_T(K_1,i_1))$

(ii) $K_1 \sharp K_2 \iff \exists n \geq 0.\ K_1 \sharp_n K_2$

(iii) $K_1 \sharp^+ K_2 \iff K_1 \sharp K_2 \wedge f(K_1) \cap f(K_2) \neq \emptyset$

Gilt $K_1 \sharp_n K_2$, so heißen K_1 und K_2 n-unterscheidbar

Gilt $K_1 \sharp K_2$, so heißen K_1 und K_2 unterscheidbar.

(2) Falls notwendig, werden die in (1) eingeführten Relationen $\sharp$, $\sharp_n$ usw. zusätzlich mit dem Index T versehen, wodurch der Bezug zum entsprechenden IH-Baum eindeutig gemacht wird.

Beispiel 4.2

In IH-TREE(IS_2) (Beispiel 4.1-1) gelten:

$[9,\{1\}] \sharp_0 [6,\{2\}]$

$\neg([8,\{1\}] \sharp_0 [5,\{1,2\}])$

$[8,\{1\}] \sharp_1 [5,\{1,2\}]$

$\neg([7,\{1\}] \sharp_0 [4,\{1,2\}])$

$\neg([7,\{1\}] \sharp_1 [4,\{1,2\}])$

$[7,\{1\}] \sharp_2 [4,\{1,2\}]$

$\neg([6,\{1\}] \sharp_2 [3,\{1,2\}])$

$[6,\{1\}] \sharp_3 [3,\{1,2\}]$

Der folgende Satz liefert die Grundlage für die Berechnung der in einem IH-Baum T geltenden Unterscheidbarkeitsrelationen.

Satz 4.1

Mit den Bezeichnungen von Def. 4.2 gelten, wobei $\dim(Q) = (m,(n_1,\ldots,n_m))$ sei:

(1) $\forall K,K' \in V'.$

$[K \sharp K' \Rightarrow \exists n<\max(n_1,\ldots,n_m)-1.\ K \sharp_n K']$

(2) $\forall n \in \mathbb{N}.\ \forall K,K' \in V'.$

$[K \sharp_n K' \Rightarrow \forall n'>n.\ K \sharp_{n'} K']$

(3) $\forall n \in \mathbb{N}.$

$[(\forall K,K' \in V'.\ K \sharp_n K' \Leftrightarrow K \sharp_{n+1} K')$

$\Leftrightarrow (\forall K,K' \in V'.\ K \sharp K' \Leftrightarrow K \sharp_n K')]$

Beweis:

Es ist zu beachten, daß $\max(n_1,\ldots,n_m) = \max\{\text{LEVEL}(K) \mid K \in V\}$ ist. Die Aussagen (1)-(3) werden leicht durch vollständige Induktion bewiesen.

Nach Aussage (1) des Satzes 4.1 sind unterscheidbare Knoten zumindest (n-2)-unterscheidbar, wenn n die maximale Länge der zugrunde liegenden Beispielsfolgen in Q ist. Nach (2) und (3) kann $\sharp$ durch Bestimmung von $\sharp_0$, $\sharp_1$, $\sharp_2$ usw. berechnet werden, bis sich zum ersten Mal $\sharp_n$ äquivalent $\sharp_{n+1}$ ergibt. Dieses Verfahren bricht somit nach (1) immer ab; in der Praxis aber oft viel früher als erst bei der hier angegebenen oberen Grenze: In Beispiel 4.2 gilt z.B. $\sharp_3$ äquivalent $\sharp$, während $\max(n_1,\ldots,n_4) = 10$ ist.

Bevor alle für die Programmsynthese relevanten Informationen über die in einem IH-Baum T geltenden Unterscheidbarkeitsrelationen in einer LB-Situation zusammengefaßt werden können, sind noch die in Def. 4.3 eingeführten Hilfsdefinitionen notwendig. Die Motivation für ihre Bezeichnungen läßt sich aus dem folgenden ableiten. Sind in T zwei Knoten K und K' unterscheidbar, so können K und K' nicht demselben Programmknoten entsprechen; andererseits wird aber jedem Knoten von T ein Programmknoten zugeordnet. Sind also z.B. n Knoten von T paarweise verschieden, so hat jedes Q realisierende Programm P,

das die in T aufgestellten I-Hypothesen benutzt, mindestens n Programmknoten. Formal werden diese Überlegungen in den Sätzen des Abschnitts 4.3 bewiesen.

Definition 4.3

Sei Q, IS,T,V',$\sharp$ wie in Def. 4.2,

$I \in$ INSTRUCTIONS; $M,M_1,M_2 \subseteq$ INSTRUCTIONS und es gelte $\max(\emptyset) = 0$.

(1) $LB_T(I) = \max\{r \mid \exists K_1,\ldots,K_r \in V_T(I).\ \forall s\in\{1,\ldots,r-1\}.\ (\forall t\in\{s+1,\ldots,r\}.\ K_s \sharp K_t)\}$

heißt Mindestanzahl der I-Knoten von Q bzgl. IS

<=>

$V_T(I) = \{K \mid K \in V'.\ f(K) = \{I\}\}$

(2) $LB_T(M) = \max\{r \mid \exists K_1,\ldots,K_r \in V_T(M).\ \forall s\in\{1,\ldots,r-1\}.\ (\forall t\in\{s+1,\ldots,r\}.\ K_s \sharp K_t)$

heißt Mindestanzahl der I-aus-M-Knoten von Q bzgl. IS

<=>

$V_T(M) = \{K \mid K \in V'.\ f(K) \cap M \neq \emptyset\}$

(3) $LB_T(M_1,M_2) = \max\{r \mid \exists K_1,\ldots,K_r \in V_T(M_1,M_2).\ \forall s\in\{1,\ldots,r-1\}\ (\forall t\in\{s+1,\ldots,r\}.\ K_s \sharp K_t)\}$

heißt Mindestanzahl der von allen I-aus-M_2 verschiedenen I-aus-M_1-Knoten von Q bzgl. IS

<=>

$V_T(M_1,M_2) = \{K \mid K \in V'.\ f(K) \cap M_1 \neq \emptyset \wedge f(K) \cap M_2 = \emptyset\}$

Beispiel 4.3

Sei T = IH-TREE(IS_2) (Beispiel 4.1-1).

I	$V_T(I)$	$LB_T(I)$	
S1(2)	{[4,{1,2}]}	1	
CL(1)	{[1,{1,2,3}]}	1	
A(1,3)	{[3,{1,2}],[6,{1}]}	2	(da [3,{1,2}] $\sharp_3$ [6,{1}])
A(1,4)	{[9,{1}]}	1	
ZR(2)	{[5,{1,2}]}	1	
sonstiges I	$\emptyset$	0	

LB_T(INSTRUCTIONS) = 9

da die folgenden Knoten paarweise unterscheidbar sind:

[1,{1,2,3}],[2,{1,2,3}],[3,{1,2}],[4,{1,2}],[5,{1,2}],[6,{1}],

[7,{1}],[9,{1}],[6,{2}]

Sei IM = {S1(2),CL(1),A(1,3),A(1,4),ZR(2)}.

V_T(INSTRUCTIONS,IM) ={[6,{3}]}

LB_T(INSTRUCTIONS,IM) = 1

Eine LB-Situation von IS ist ein 6-Tupel, das i.w. verschiedene untere Grenzen für die Instruktions- und LABEL-Hypothesen von IS angibt. Mit den Bezeichnungen der folgenden Definition gilt: Jedes aus einem terminalen Nachfolger IS' von IS hervorgehende Programm P, das Q realisiert, erfüllt die folgenden Bedingungen:

1. P hat mindestens die Länge Lmin.
2. Für den Restgraph R werden in P noch mindestens Lbr Programmknoten benötigt, die für den Anfangsgraph A

nicht benötigt werden und mit Instruktionen nicht aus IM markiert sind.

4. Alle K' aus d(K) haben von K verschiedene LABEL.
5. P enthält für jedes I aus IM mindestens einen I-Knoten.
6. Es gibt in P mindestens l verschiedene Programmknoten mit Instruktionsmarkierung aus IM.
7. Falls die Länge $L_b(P)$ kleiner oder gleich L ist, so sind alle in P auftretenden Instruktionen in IH-SET(LBS,L) enthalten.

Alle diese Aussagen werden ebenfalls in Abschnitt 4.3 bewiesen.

Definition 4.4

Q sei PSP, G = IH-GRAPH(Q), IS = (A,R) sei eine aus G ableitbare IH-Situation, $A = (V_A, E_A, a, c)$.

(1) LB-SIT(G,IS) = LBS heißt LB-Situation von IS <=>

(i) IS hat keine terminale Nachfolgesituation =>

LBS = ε

(ii) IS hat eine terminale Nachfolgesituation =>

sei T = IH-TREE(IS), $T = (V_T, E_T, f_T, t_T)$,

dann gilt:

LBS = (Lmin,Lbr,Lb,d,IM,l) mit:

(ii.1) $Lmin = LB_T(INSTRUCTIONS)$

(ii.2) $Lbr = LB_T(INSTRUCTIONS, IM)$

(ii.3) $Lb : INSTRUCTIONS \longrightarrow \mathbb{N}$

$Lb = \lambda I.\ LB_T(I)$

(ii.4) $d : V_A \longrightarrow POT(V_A)$

$d = \lambda K.\ \{K' \mid K' \in V_A, K \neq_T^+ K'\}$

(ii.5) $IM = \{I \mid I \in INSTRUCTIONS, \exists K \in V_T.\ f_T(K) = \{I\}\}$

(ii.6) $l = \sum_{I \in IM} Lb(I)$

(2) Sei LBS wie in (1).(ii), $L \in \mathbb{N}$.

IH-SET(LBS,L) = IHM heißt relevante IH-Menge bzgl. LBS und L <=>

$$IHM = \begin{cases} IM & <=> \ l \geq L \\ INSTRUCTIONS & <=> \ l < L \end{cases}$$

Beispiel 4.4

Mit den Werten des vorigen Beispiels ergibt sich:

$LBS = LB\text{-}SIT(G_3, IS_2) = (9,1,Lb,d,\{S1(2),CL(1),A(1,3),A(1,4),ZR(2)\},6)$

wobei gilt:

(i) Da der Anfangsgraph von IS_2 nur die Knoten $[0,\{1,2,3\}]$ und $[1,\{1,2,3\}]$ enthält, gilt in diesem speziellen Fall $d = \lambda K.\emptyset$

(ii) für $I \in \{S1(2),CL(1),A(1,4),ZR(2)\}$: $Lb(I) = 1$

$I = A(1,3)$: $Lb(I) = 2$

sonstiges I : $Lb(I) = 0$

Für diese LB-Situation und L = 9 ergibt sich:

IH-SET(LBS,L) = INSTRUCTIONS

4.2.2 Berechnung von LB-Situationen

Zur Berechnung einer LB-Situation können die benötigten Werte $LB_T(INSTRUCTIONS)$ und $LB_T(INSTRUCTIONS, IM)$ auf einfachere Summenformeln zurückgeführt werden. Hierzu werden die in T auftretenden Instruktionen derartig in Klassen zerlegt, daß sich daraus auch eine Zerlegung der Knotenmenge von T ergibt, was zu den Gleichungen des folgenden Satzes führt.

Satz 4.2

Sei T,V' wie in Def. 4.2; IM $\subseteq$ INSTRUCTIONS.

Weiterhin sei:

(a) $HM = \bigcup_{K \in V'} f(K)$

(b) R^t die transitive Hülle der auf HM wie folgt definierten Relation R :

$(I_1, I_2) \in R \iff \exists K \in V'.\ I_1 \in f(K) \wedge I_2 \in f(K)$

(c) $\{M_1, \ldots, M_r\}$ die durch die Äquivalenzrelation R^t auf der Menge HM induzierte Zerlegung.

$\{M_1, \ldots, M_r\}$ heißt Instruktionszerlegung (I-Zerlegung) von T.

Behauptung:

(1) $\{V_T(M_1), \ldots, V_T(M_r)\}$ ist Zerlegung von V'.

(2) Für alle Teilmengen $\{i_1, \ldots, i_s\}$ von $\{1, \ldots, r\}$ gilt:

(2.1) $LB_T(M_{i_1}) + \ldots + LB_T(M_{i_s}) = LB_T(M_{i_1} \cup \ldots \cup M_{i_s})$

(2.2) $LB_T(M_{i_1}, IM) + \ldots + LB_T(M_{i_s}, IM) = LB_T(M_{i_1} \cup \ldots \cup M_{i_s}, IM)$

(3) $LB_T(INSTRUCTIONS) = \sum_{i=1}^{r} LB_T(M_i)$

(4) $LB_T(INSTRUCTIONS, IM) = \sum_{i=1}^{r} LB_T(M_i, IM)$

(5) $\forall i \in \{1, \ldots, r\}.$

(5.1) $V_T(M_i, IM) \subseteq V_T(M_i)$

(5.2) $LB_T(M_i, IM) \leq LB_T(M_i)$

(6) $\forall i \in \{1, \ldots, r\}.\ M_i \cap IM = \emptyset \Rightarrow$

(6.1) $V_T(M_i, IM) = V_T(M_i)$

(6.2) $LB_T(M_i, IM) = LB_T(M_i)$

Beweis: Die Aussagen (1)-(6) ergeben sich insbesondere mit Hilfe der Def. 4.2 und 4.3.

Beispiel 4.5

1. Für das vorige Beispiel ergibt sich mit den Bezeichnungen von Satz 4.2:

HM = {A1(1),S1(2),CL(1),CL(3),A(i,1),A(i,2),A(i,3),A(1,4), A(1,5),ZR(i)}

I-Zerlegung $\{M_1, M_2\}$:

$M_1 = HM \setminus \{A(1,4)\}$

$M_2 = \{A(1,4)\}$

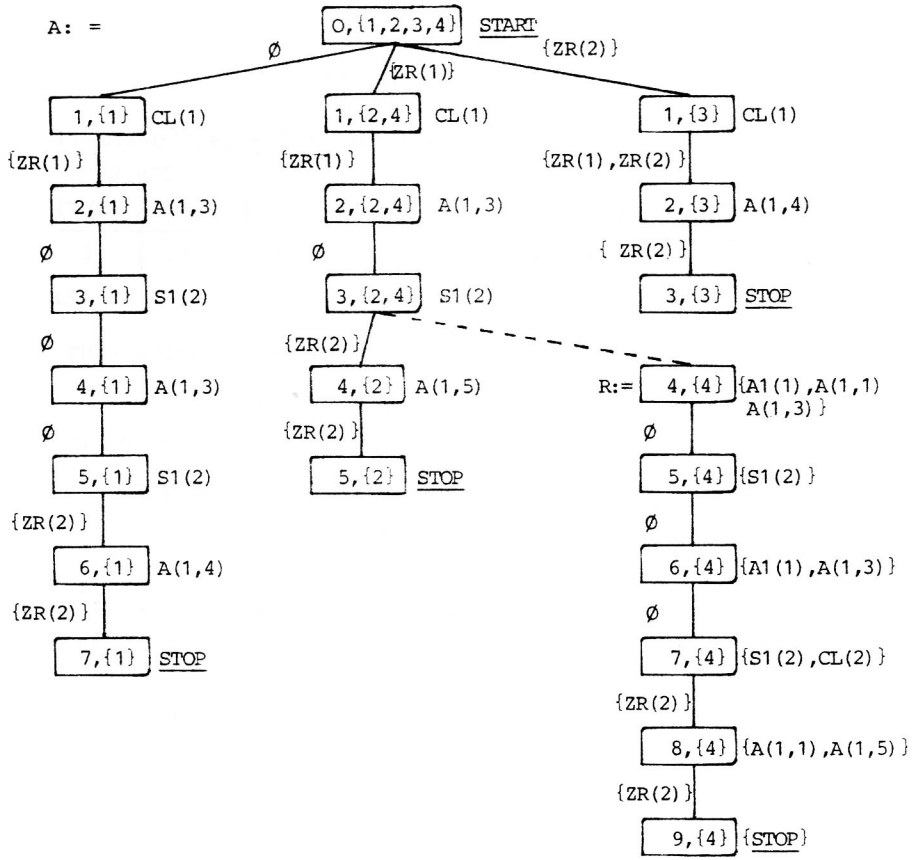
$LB_T(M_1) = 8$

$LB_T(M_2) = 1$

$V_T(M_1, IM) = \{[6, \{3\}]\}$, $LB_T(M_1, IM) = 1$

$V_T(M_2, IM) = \emptyset$ $LB_T(M_2, IM) = 0$

2. Als weiteres Beispiel sei die aus G_4 = IH-GRAPH(Q_4) (Beispiel 3.3-4) ableitbare IH-Situation IS = (A,R) wie folgt gegeben:



Der IH-Baum T von IS ergibt sich durch Substition der Knotenmarkierungen a(K) durch {a(K)} und durch eine zusätzliche, mit Ø markierte Kante von [3,{2,4}] nach [4,{4}].

Zur Bestimmung von LB-SIT(G_4,IS) = (Lmin,Lbr,Lb,d,IM,l) werden zunächst die in T geltenden Unterscheidbarkeitsrelationen angegeben.

	K	{K' | K $\stackrel{+}{\rightarrow}$ K'}	d(K)	f(K)	$V_T(M_1)$	$V_T(M_2)$	$V_T(M_3)$	$V_T(M_4)$
Knoten aus A	[1,{1}]	Ø	Ø	CL(1)	*			
	[2,{1}]	[4,{4}]	Ø	A(1,3)		*		
	[3,{1}]	[5,{4}]	Ø	S1(2)			*	
	[4,{1}]	[2,{2,4}],[6,{4}]	[2,{2,4}]	A(1,3)		*		
	[5,{1}]	[3,{2,4}],[7,{4}]	[3,{2,4}]	S1(2)			*	
	[6,{1}]	Ø	Ø	A(1,4)				*
	[1,{2,4}]	Ø	Ø	CL(1)	*			
	[2,{2,4}]	[4,{1}]	[4,{1}]	A(1,3)		*		
	[3,{2,4}]	[5,{1}]	[5,{1}]	S1(2)			*	
	[4,{2}]	Ø	Ø	A(1,5)		*		
	[1,{3}]	Ø	Ø	CL(1)	*			
	[2,{3}]	Ø	Ø	A(1,4)				*
Knoten aus F	[4,{4}]	[2,{1}]	-	A1(1),A(1,1),A(1,3)		*		
	[5,{4}]	[3,{1}]	-	S1(2)			*	
	[6,{4}]	[4,{1}]	-	A1(1),A(1,3)		*		
	[7,{4}]	[5,{1}]	-	S1(2),CL(2)			*	
	[8,{4}]	Ø	-	A(1,1),A(1,5)		*		

Mit den Bezeichnungen des Satzes 4.2 gilt:

HM = {A1(1),S1(2),CL(1),CL(2),A(1,1),A(1,3),A(1,4),A(1,5)}

Instruktionszerlegung von T ist $\{M_1,\ldots,M_4\}$ mit:

M_1 = {CL(1)} M_2 = {A1(1),A(1,1),A(1,3),A(1,5)}

M_3 = {S1(2),CL(2)} M_4 = {A(1,4)}

Die Zerlegung $\{V_T(M_1),V_T(M_2),V_T(M_3),V_T(M_4)\}$ der Knotenmenge ist in der obigen Tabelle angegeben.

Wegen: $[4,\{1\}] \neq [2,\{2,4\}]$

$[4,\{1\}] \neq [8,\{4\}]$

$[2,\{2,4\}] \neq [8,\{4\}]$

und: $[3,\{1\}] \neq [5,\{1\}]$

gilt:

$$LB_T(\text{INSTRUCTIONS}) = \sum_{i=1}^{4} LB_T(M_i) = 1 + 3 + 2 + 1 = 7 = \text{Lmin}$$

Für IM der LB-Situation gilt:

IM = {S1(2),CL(1),A(1,3),A(1,4),A(1,5)}

und somit:

V_T(INSTRUCTIONS,IM) = Ø

LB_T(INSTRUCTIONS,IM) = 0 = Lbr

Für Lb gilt:

I	S1(2)	CL(1)	A(1,3)	A(1,4)	A(1,5)	sonst. I
Lb(I)	2	1	2	1	1	0

LB-SIT(G_4,IS) = (7,0,Lb,d,IM,7)

4.2.3 LB-Situationen als Maß für Instruktionshypothesen

In Def. 4.4 wurden LB-Situationen als Zusammenfassung von verschiedenen unteren Grenzen für die Instruktions- und LABEL-Hypothesen einer IH-Situation IS eingeführt. Als nächstes ist nun die Frage zu stellen, wie sich die LB-Situationen von IS und IS' zueinander verhalten, wenn IS' terminaler Nachfolger von IS ist. Hierbei ergibt sich jedoch wieder eine Schwierigkeit für den Fall b = ss. Wenn T und T' die zu IS und IS' zugehörigen IH-Bäume sind, so sind die Knotenmengen von T und T' für b∈{s,a,as} jeweils identisch, während dies für b = ss i.a. nicht der Fall ist. Allerdings läßt sich auch für b = ss für jeden Knoten K aus T ein Knoten F(K) aus T' finden, der K entspricht. Die dabei benützte Abbildung F wird als Korrespondenzabbildung zwischen T und T' bezeichnet und wird durch den nachfolgenden Satz eindeutig festgelegt. Für b∈{s,a,as} ist F die Identitätsabbildung.

Die für das weitere Vorgehen wichtigste Aussage des Satzes 4.3 ist unter (6) gegeben: sind in T zwei Knoten K und K' unterscheidbar, so sind auch die entsprechenden Knoten F(K) und F(K') in T' unterscheidbar.

Satz 4.3

Sei Q = (b,M) ein PSP; IS, IS' aus IH-GRAPH(Q) ableitbare IH-Situationen. IS' sei terminaler Nachfolger von IS;

IS = (A,R) = $((V_A,E_A,a,c),(V_R,E_R,f_R,g_R,t_R))$,

IS' = (A',R') = $((V'_A,E'_A,a',c'),(Ø,Ø,Ø,Ø,Ø))$.

Sei $T = IH\text{-}TREE(IS) = (V_T, E_T, f_T, t_T)$,

$T' = IH\text{-}TREE(IS') = (V'_T, E'_T, f'_T, t'_T)$.

Behauptung:

(1) $\forall K \in V_T [\exists K' \in V'_T .\ LEVEL(K) = LEVEL(K') \wedge INDEX(K) \subseteq INDEX(K')]$

(2) $\forall K \in V_T [\forall K', K" .\ LEVEL(K) = LEVEL(K') \wedge INDEX(K) \subseteq INDEX(K')$
$\wedge LEVEL(K) = LEVEL(K") \wedge INDEX(K) \subseteq INDEX(K") \Rightarrow K' = K"]$

(3) Die Abbildung $F : V_T \longrightarrow V'_T$ ist wohldefiniert und eindeutig festgelegt durch:

$F = \lambda K .\ K' \in V'_T \wedge LEVEL(K) = LEVEL(K') \wedge INDEX(K) \subseteq INDEX(K') \longrightarrow K'$

Notation: F heißt Korrespondenzabbildung zwischen T und T' und wird bezeichnet durch CF(T,T').

Eigenschaften von F:

(4) $\forall K \in V_T .\ f'_T(F(K)) \subseteq f_T(K)$

(5) $\forall K \in V_T ,\ f_T(K) \neq \{STOP\}$.
$\forall i \in INDEX(K) .\ \forall I \in f'_T(F(K))$
$F(NF_T(K,i)) = NF_{T'}(F(K),i)$
$t^I_T(K, NF_T(K,i)) = t'^I_T(F(K), NF_{T'}(F(K),i))$

(6) $\forall K_x, K_y \in V_T .\ K_x \ddagger_T K_y \Rightarrow F(K_x) \ddagger_{T'} F(K_y)$

Beweis:

(1) + (2): Für $b \in \{s, a, as\}$ gilt $V_T = V'_T$, woraus (1) und (2) direkt folgen; sei also b = ss .

(i) Da $V_A \subseteq V_T$, $V_A \subseteq V'_A$ und $V'_A \subseteq V'_T$, gilt auch $V_A \subseteq V'_T$, woraus für $K \in V_A$ Aussage (1) direkt folgt. Da für $K \in V_A$ und $K' \in V'_T \setminus V_A$ $LEVEL(K) \neq LEVEL(K')$ oder $INDEX(K) \nsubseteq INDEX(K')$ gilt, gilt für $K \in V_A$ auch Aussage (2).

(ii) Sei $K \in V_T \setminus V_A$, $l = LEVEL(K)$.
Sei IND_l und $\{IND^1_l, \ldots, IND^{r_l}_l\}$ wie in Def. 4.1
Sei weiterhin:

$$IND'_l = \bigcup_{\substack{K \in V'_T \setminus V_A \\ l = LEVEL(K)}} INDEX(K)$$

$IZ_l = \{INDEX(K) \mid K \in V'_T \setminus V_A, l = LEVEL(K)\}$

Damit gilt:
$IND_l = IND'_l$ und $\{IND^1_l, \ldots, IND^{r_l}_l\}$ ist eine Verfeinerung von IZ_l. Somit gibt es für K also genau ein $IK \in IZ_l$, sodaß $INDEX(K) \subseteq IK$, woraus Aussagen (1) und (2) für $K \in V_T \setminus V_A$ folgen.

(3) : folgt direkt aus (1) und (2)

(4) + (5): Folgerungen aus den Spezifikationen von T,T' und F.

(6) : Sei $K_x, K_y \in V_T$.
Für $n \geq 0$ sei A[n] die Aussage:
$A[n] : K_x \ddagger_{T,n} K_y \Rightarrow F(K_x) \ddagger_{T',n} F(K_y)$
Für alle $n \geq 0$ wird A[n] durch Induktion bewiesen, woraus (6) folgt.

n = 0 : (i) $K_x \ddagger_{T,0} K_y \Rightarrow f_T(K_x) \cap f_T(K_y) = \emptyset$
(ii) $f'_T(F(K_x)) \subseteq f_T(K_x) \wedge f'_T(F(K_x)) \subseteq f_T(K_y)$
$\Rightarrow f'_T(F(K_x)) \cap f'_T(F(K_y)) = \emptyset$
$\Rightarrow F(K_x) \ddagger_{T',0} F(K_y)$

n > 0 : Für n>0 sei A[n] bereits bewiesen.

n + 1 : Falls $f'_T(F(K_x)) \cap f'_T(F(K_y)) = \emptyset$, gilt $F(K_x) \ddagger_{T',0} F(K_y)$ und damit natürlich auch $F(K_x) \ddagger_{T,n+1} F(K_y)$.

Sei also $f'_T(F(K_x)) \cap f'_T(F(K_y)) \neq \emptyset$. Da IS' terminal ist, gibt es also ein $I \varepsilon INSTRUCTIONS$, sodaß $\{I\} = f'_T(F(K_x)) = f'_T(F(K_y))$. Wegen (4) gilt dann auch $I \varepsilon f_T(K_x) \wedge I \varepsilon f_T(K_y)$. Da $K_x \not\equiv_{T,n+1} K_y$ nach Voraussetzung, gilt folgendes:

(i) $\exists\, i_x \varepsilon INDEX(K_x),\ i_y \varepsilon INDEX(K_y)$.
$t^I(K_x, NF_T(K_{x,i_x})) = t^I(K_y, NF_T(K_y,i_y))$
$\wedge\ NF_T(K_x,i_x) \not\equiv_{T,n} NF_T(K_y,i_y)$

(ii) $F(NF_T(K_x,i_x)) \not\equiv_{T',n} F(NF_T(K_y,i_y))$ [Ind.-annahme]

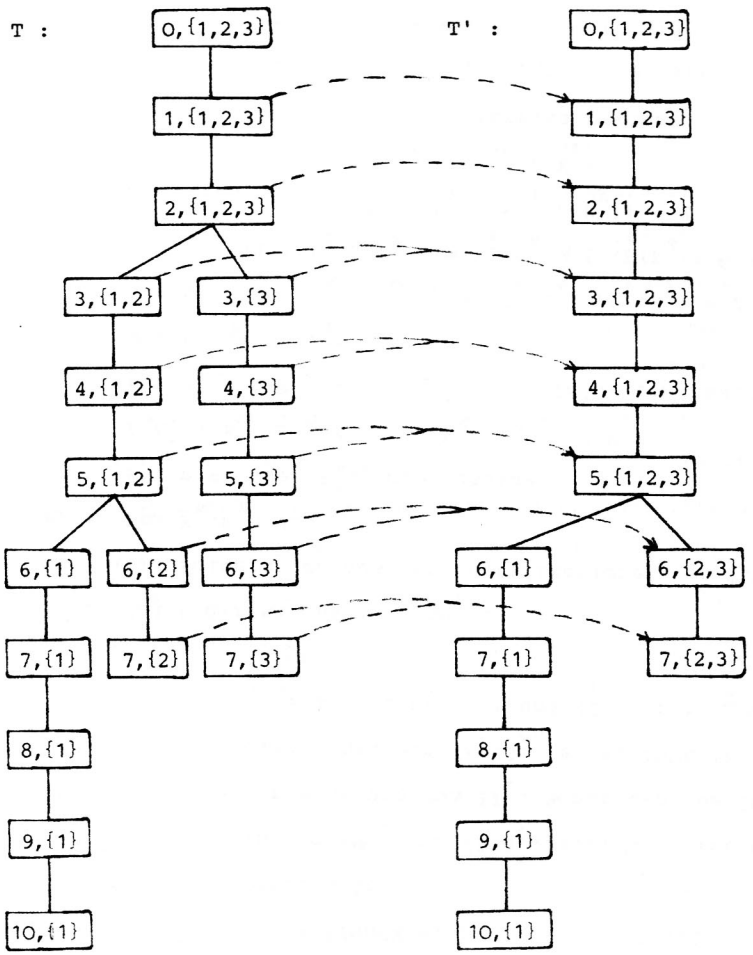
(iii) $NF_{T'}(F(K_x),i_x) \not\equiv_{T',n} NF_{T'}(F(K_y),i_y)$ [wegen (5)]

(iv) $t'^I_T(F(K_x), NF_{T'}(F(K_x),i_x)) = t'^I_T(F(K_y), NF_{T'}(K_y,i_y))$ [(5)]

$\Rightarrow F(K_x) \not\equiv_{T',n+1} F(K_y)$

Beispiel 4.6

Zur Illustration des Satzes dienen die aus G_3 ableitbaren IH-Situationen IS_2 (Beispiel 3.10-2) und IS_5 (Beispiel 3.11-3). $T = IH\text{-}TREE(IS_2)$ ist in Beispiel 4.1-1 angegeben; $T' = IH\text{-}TREE(IS_5)$ ergibt sich aus der Graphenstellung für A_5 (Beispiel 3.11-3), indem man eine Knotenmarkierung a(K) durch die Menge {a(K)} und eine Kantenmarkierung c(K,K') durch die Menge {(a(K),c(K,K'))} ersetzt. Der Übersichtlichkeit halber werden hier beide IH-Bäume noch einmal skizziert:



Die gebrochenen Linien deuten die Abbildung $F = CF(T,T')$ an.

Mit Hilfe der Korrespondenzabbildung F können nun die im folgenden Satz aufgeführten Relationen zwischen den LB-Situationen von IS und von einem terminalen Nachfolger IS' bewiesen werden. Diese Relationen erlauben es, die in der LB-Situation von IS gegebene Menge von unteren Grenzen als Maß für die "Aufwendigkeit" der durch IS gewählten Instruktionshypothesen zu bezeichnen. Inwiefern es sich dabei um die Aufwendigkeit der Hypothesen handelt, wurde bereits in der Erläuterung zur Def. 4.4 gesagt. Aufgrund des folgenden Satzes gilt nun (mit den jeweils notwendigen Modifikationen): jede untere Grenze für IS ist auch eine untere Grenze für jeden terminalen Nachfolger von IS.

Satz 4.4

Q sei PSP; IS,IS' aus G = IH-GRAPH(Q) ableitbare IH-Situationen; IS' sei terminaler Nachfolger von IS = (A,R),
wobei A = (V_A,E_A,a,c).
LB = LB-SIT(G,IS) = (Lmin,Lbr,Lb,d,IM,l)
LB'= LB-SIT(G,IS')= (Lmin',Lbr',Lb',d',IM',l').

Behauptung:

(1) $Lmin \leq Lmin'$

(2) $\forall$ IεINSTRUCTIONS. $Lb(I) \leq Lb'(I)$

(3) $Lbr \leq \sum_{I \in IM' \setminus IM} Lb'(I)$

(4) $\forall K \in V_A.\ d(K) \subseteq d'(K) \cap V_A$

(5) $IM \subseteq IM'$

(6) $l \leq l'$

(7) Lbr' = 0

(8) $\forall$ IεINSTRUCTIONS. IεIM' <=> Lb'(I) ≠ 0

(9) Lmin' = l'

Beweis:

Sei T = IH-TREE(IS) = (V_T,E_T,f_T,t_T),

T' = IH-TREE(IS') = (V'_T,E'_T,f'_T,t'_T), F = CF(T,T')

(1) $Lmin = LB_T(INSTRUCTIONS)$ [Def.4.4]

$\Rightarrow \exists K_1,\ldots,K_{lmin} \in V_T$

$\forall s \in \{1,\ldots,Lmin-1\}(\forall t \in \{s+1,\ldots,Lmin\}.\ K_s \not\equiv_T K_t)$

$\Rightarrow \forall s \in \{1,\ldots,Lmin-1\}(\forall t \in \{s+1,\ldots,Lmin\}.\ F(K_s) \not\equiv_{T'} F(K_t))$ [Satz 4.3 (6)]

(2) analog zu (1)

(3) $Lbr = LB_T(INSTRUCTIONS,IM)$

$\Rightarrow \exists\ K_1,\ldots,K_{Lbr}\ .$

$(\forall i \in \{1,\ldots,Lbr\}.\ f_T(K_i) \subseteq INSTRUCTIONS$

$f_T(K_i) \cap IM = \emptyset$

$\wedge \forall s \in \{1,\ldots,Lbr-1\}(\forall t \in \{s+1,\ldots,Lbr\}.\ K_s \not\equiv K_t))$

$\Rightarrow \forall i \in \{1,\ldots,Lbr\}(f'_T(F(K_i)) \subseteq f_T(K_i)$ [Satz 4.3 (4)]

$\wedge\ f'_T(F(K_i)) \subseteq IM'$ [Def. 4.4]

$\wedge\ f'_T(F(K_i)) \cap IM = \emptyset$

$\wedge\ f'_T(F(K_i)) \subseteq IM' \setminus IM$

$\wedge\ |f'_T(F(K_i))| = 1)$ [IS' terminal]

$\wedge \forall s \in \{1,\ldots,Lbr-1\}(\forall t \in \{s+1,\ldots,Lbr\}.$

$F(K_s) \not\equiv_{T'} F(K_t))$ [Satz 4.3 (6)]

$\Rightarrow Lbr \leq \sum_{I \in IM' \setminus IM} Lb'(I)$

(4) folgt direkt aus $V_A \subseteq V_T$ und Satz 4.3 (6)

(5) gilt wegen Def. 3.12 (1)

(6) Folgerung aus (2) und (5)

(7) - (9) folgen direkt aus Def. 4.4, da IS' terminal.

Beispiel 4.7

Für IS_2 und IS_5 aus den vorigen Beispielen gilt:

LB = LB-SIT(G_3,IS_2) = (9,1,Lb,d,IM,6) [Beispiel 4.4]

LB'= LB-SIT(G_3,IS_5) = (9,0,Lb',d',IM',9)

I	Lb(I)	Lb'(I)
S1(2)	1	2
CL(1)	1	1
A(1,3)	2	2
A(1,4)	1	1
A(1,5)	0	1
ZR(2)	1	2
sonst. I	0	0

IM = {S1(2),CL(1),A(1,3),A(1,4),ZR(2)}

IM'= IM $\cup$ {A(1,5)}

Für L = 9 gilt:

IH-SET(LB,L) = INSTRUCTIONS

IH-SET(LB',L)= IM'

d und d' sind nur für Knoten des entsprechenden Anfangsgraphen bestimmt; es ergibt sich:

K	d(K)	d'(K)
[1,{1,2,3}]	∅	∅
[2,{1,2,3}]	-	{[5,{1,2,3}]}
[3,{1,2,3}]	-	{[6,{1}]}
[4,{1,2,3}]	-	{[7,{1}]}
[5,{1,2,3}]	-	{[2,{1,2,3}],[8,{1}]}
[6,{1}]	-	{[3,{1,2,3}]}
[6,{2,3}]	-	∅
[7,{1}]	-	{[4,{1,2,3}]}
[8,{1}]	-	{[5,{1,2,3}]}
[9,{1}]	-	∅

4.3 Hypothesenreduktion

Das in Abschnitt 4.2 eingeführte Konzept der Unterscheidbarkeitsrelationen und LB-Situationen kann dazu benützt werden, die Anzahl der zu überprüfenden Instruktions- und LABEL-Hypothesen zu reduzieren. In 4.3.1 werden Eigenschaften bewiesen, die in erster Linie der LABEL-Hypothesenreduktion dienen, und in 4.3.2 werden Möglichkeiten der Instruktionshypothesenreduktion aufgezeigt.

4.3.1 LABEL-Hypothesenreduktion

Eine eindeutige Markierungsfunktion (oder: LABEL-Hypothese) u für einen IH-Anfangsgraphen A mit der Auswahlfunktion a ordnet jedem Knoten K von A einen Programmknoten des zu synthetisierenden Programms P zu, nämlich (u(K),a(K)).
In Satz 4.5 wird die wesentlichste Eigenschaft der Unterscheidbarkeitsrelation bewiesen: sind zwei Knoten K und K' von A unterscheidbar, so müssen auch die ihnen zugeordneten Programmknoten verschieden sein. Gilt insbesondere a(K) = a(K'), so muß also u(K) ≠ u(K') gelten. Es genügt folglich, LABEL-Hypothesen zu berücksichtigen, die diese Eigenschaft haben.

Satz 4.5

Q sei PSP, IS = (A,R) aus G = IH-GRAPH(Q) ableitbare terminale IH-Situation. u sei eindeutige Markierung für A = (V_A,E_A,a,c);
T = IH-TREE(IS) = (V_T,E_T,f_T,t_T).
Da IS terminal, gilt also $V_A = V_T$. Sei '$\neq$' := '$\neq_T$' .

Behauptung: $\forall K,K' \in V_A.\ K \neq K' \Rightarrow (u(K),a(K)) \neq (u(K'),a(K'))$

Beweis:

Sei $K,K' \in V_A$. Für $n \geq 0$ sei A[n] die Aussage:
A[n] : $K \neq_n K' \Rightarrow (u(K),a(K)) \neq (u(K'),a(K'))$
Für alle $n \geq 0$ wird A[n] durch Induktion bewiesen, woraus die Behauptung des Satzes folgt.

n = 0 : $K \neq_0 K' \Rightarrow f_T(K) \cap f_T(K') = \emptyset$
$\Rightarrow a(K) \neq a(K')$
$\Rightarrow (u(K),a(K)) \neq (u(K'),a(K'))$

n > 0 : für n > 0 sei A[n] bereits bewiesen.

n + 1 : Falls a(K) ≠ a(K'), ist nichts mehr zu zeigen.
Sei also a(K) = a(K') = I.
$K \neq_{n+1} K' \Rightarrow \exists\ i \in INDEX(K), i' \in INDEX(K').$
$(t_T^I(K,NF_T(K,i)) = t_T^I(K',NF_T(K',i'))$
$\wedge\ NF_T(K,i) \neq_n NF_T(K',i))$
Sei $K_1 = NF_T(K,i)$ und $K_1' = NF_T(K',i')$
$\Rightarrow (u(K_1),a(K_1)) \neq (u(K_1'),a(K_1'))$ [Ind.Annahme]

Widerspruchsannahme: u(K) = u(K')

$((u(K),a(K)),t_T^I(K,K_1),(u(K_1),a(K_1))) \in$ U-GRAPH(A,u)

$((u(K'),a(K')),t_T^I(K',K_1'),(u(K_1'),a(K_1'))) \in$ U-GRAPH(A,u)

$\Rightarrow (u(K_1),a(K_1)) = (u(K_1'),a(K_1'))$ [Eindeutigkeit von u]

Widerspruch!

Es muß also u(K) ≠ u(K') gelten und damit natürlich auch $(u(K),a(K)) \neq (u(K'),a(K'))$.

Beispiel 4.8

In IH-TREE(IS_5), der sich aus der Graphendarstellung aus Beispiel 3.11-3 ergibt, gilt für
K = [5,{1,2,3}], K' = [2,{1,2,3}] und K" = [8,{1}]:

(i) $K \neq K'$

(ii) $K \neq K''$

Für A_5 brauchen damit nur solche LABEL-Hypothesen u überprüft zu werden, für die

$$u(K) \neq u(K') \quad \text{und} \quad u(K) \neq u(K'')$$

gilt.

In Satz 4.6 werden aufgrund der LB-Situation von IS ableitbare Beziehungen zwischen IS und einem terminalen Nachfolger IS' bewiesen, die ebenfalls zu einer Reduktion der zu überprüfenden LABEL-Hypothesen führen.

Satz 4.6

Sei IS,LB,IS' wie in Satz 4.4; u sei eindeutige Markierung für A'.

Behauptung:

(1) $Lmin \leq UL(A',u)$

(2) $Lbr \leq UL(A',u,INSTRUCTIONS \setminus IM)$

(3) $\forall I \in INSTRUCTIONS.\ Lb(I) \leq UL(A',u,I)$

(4) $\forall K \in V_A (\forall K' \in d(K).\ u(K) \neq u(K'))$

(5) $u_{|V_A}$ ist eindeutige Markierung für A.

(6) $\forall K \in V_A (\forall K' \in d(K).\ u_{|V_A}(K) \neq u_{|V_A}(K'))$

(7) $\sum_{I \in IM} \max\{Lb(I), UL(A,u_{|V_A},I)\} \leq UL(A',u) - Lbr$

Beweis:

Sei LB',T,T',E wie im Beweis von Satz 4.4

(1) $Lmin \leq Lmin' = LB_T'(INSTRUCTIONS)$ [Satz 4.4 (1)]

$\Rightarrow \exists K_1,\ldots,K_{Lmin'} \in V_T'$.
$\forall s \in \{1,\ldots,Lmin'-1\} (\forall t \in \{s+1,\ldots,Lmin'\}.\ K_s \neq_T K_t)$

$\Rightarrow \forall s \in \{1,\ldots,Lmin'-1\} (\forall t \in \{s+1,\ldots,Lmin'\}.$ [Satz 4.5]
$(u(K_s),a(K_s)) \neq (u(K_t),a(K_t)))$

$\Rightarrow Lmin' \leq UL(A',u)$

$\Rightarrow Lmin \leq UL(A',u)$

(2) + (3) analog zu (1) mit Hilfe von Satz 4.4 (2) + (3) und Satz 4.5

(4) $K \in V_A \wedge K' \in d(K)$

$\Rightarrow a(K) = a(K') \wedge K \neq_T K'$

$\Rightarrow (u(K),a(K)) \neq (u(K'),a(K'))$ [Satz 4.5]

$\Rightarrow u(K) \neq u(K')$

(5) trivial, da IS' Nachfolger von IS

(6) Folgerung aus (4) und Satz 4.4 (4)

(7) $\sum_{I \in IM} \max\{Lb(I), UL(A,u_{|V_A},I)\}$

$\leq \sum_{I \in IM} \max\{UL(A',u,I), UL(A,u_{|V_A},I)$ [nach (3)]

$= \sum_{I \in IM} UL(A',u,I)$

$= \sum_{I \in INSTRUCTIONS} UL(A',u,I) - \sum_{I \in INSTRUCTIONS \setminus IM} UL(A',u,I)$

$= UL(A',u) - UL(A',u,INSTRUCTIONS \setminus IM)$ [Def. 3.13]

$\leq UL(A',U) - Lbr$ [nach (2)]

Aus den Punkten (1) - (4) dieses Satzes folgen direkt: jede auf IS aufbauende Hypothesenbildung führt zu einem Programm mindestens der Länge Lmin; für den Restgraph R werden

in P noch mindestens Lbr Programmknoten benötigt, die mit Instruktionen markiert sind, die nicht in IM enthalten sind; P besitzt mindestens Lb(I) I-Knoten; für K aus V_A haben alle K' aus d(K) von K verschiedene LABEL.

Aus den Punkten (5) - (7) läßt sich ableiten: Die Einschränkung von u auf V_A ist eindeutige Markierung für A, unter der für $K \in V_A$ alle K' aus d(K) von K verschiedene LABEL haben, wobei die Relation unter (7) wie folgt interpretiert wird: Jeder I-Knoten in u-GRAPH(A,$u_{|V_A}$) kommt auch in U-GRAPH(A',u) vor. In U-GRAPH(A',u) treten aber andererseits mindestens Lb(I) I-Knoten auf. Weiterhin ist jede Instruktion, die in U-GRAPH(A,$u_{|V_A}$) enthalten ist, auch in IM. Nimmt man also jeweils das Maximum aus Mindestanzahl der I-Knoten für A' und tatsächlicher Anzahl der bereits für A "verbrauchten" I-Knoten und bildet die Summe darüber für alle I aus IM, so kann diese Summe nicht größer sein als die Gesamtzahl der Programmknoten von A' und u abzüglich der Mindestanzahl der Knoten, die in U-GRAPH(A',u) mit Instruktionen markiert sind, die nicht in IM enthalten sind.

Beispiel 4.9

Die Aussagen des Satzes 4.6 sollen an den IH-Situationen $IS = IS_2$ und $IS' = IS_5$ (vgl. Beispiele 3.10-2, 3.11-3, 4.6 und 4.7) verdeutlicht werden. u sei die eindeutige Markierung für A' (= A_5), die den Knoten [5,{1,2,3}], [6,{1}] und [7,{1}] eine 2, allen übrigen Knoten eine 1 zuordnet. Dann gilt mit den Bezeichnungen des Satzes 4.6:

Lmin = 9 = UL(A',u)

Lbr = 1 = UL(A',u,INSTRUCTIONS $\setminus$ IM)

I	Lb(I)	UL(A',u,I)	UL(A,$u_{\|V_A}$,I)	max{Lb(I), UL(A,$u_{\|V_A}$,I)}
S1(2)	1	2	0	1
CL(1)	1	1	1	1
A(1,3)	2	2	0	2
A(1,4)	1	1	0	1
A(1,5)	0	1	0	0
ZR(2)	1	2	0	1
sonst. I	0	0	0	0

$$\sum_{I \in IM} \max\{Lb(I), UL(A,u_{|V_A},I)\} = 6 \leq 9 - 1 = UL(A',u) - Lbr$$

In Satz 4.7 werden die Ergebnisse zur LABEL-Hypothesenreduktion aus den vorigen Sätzen zusammengefaßt und erweitert. Die Aussage dieses Satzes kann man sich wie folgt verdeutlichen.

Bei der Ausführung von PA(Q) liege eine IH-Situation IS = (A,R) vor, die LB-Situation von IS sei (Lmin,Lbr,Lb,d,IM,l), es sei $L \geq Lmin$ und es werden gerade überprüft, ob es ein Programm der Länge L gibt, das Q realisiert. Gibt es nun keine eindeutige Markierung u für A, unter der für K' aus d(K) $u(K) \neq u(K')$ und für die $\sum_{I \in IM} \max\{Lb(I), UL(A,u,I)\} \leq L - Lbr$ gilt, so kann daraus bereits geschlossen werden, daß keine auf IS aufbauende Hypothesenbildung zu einem Programm der Länge kleiner oder gleich L führen kann.

Satz 4.7

Q sei PSP; IS = (A,R) aus G = IH-GRAPH(Q) ableitbare IH-Situation, A = (V_A,E_A,a,c), LB-SIT(G,IS) = (Lmin,Lbr,Lb,d,IM,l). Sei L,N $\in \mathbb{N}$, L $\geq$ Lmin, N = L - Lbr.

Behauptung:

$$[\forall u.\ u \text{ eindeutige Markierung für } A.$$
$$((\forall K \in V_A.(\forall K' \in d(K).\ u(K) \neq u(K'))) \Rightarrow \sum_{I \in IM} \max\{Lb(I),UL(A,u,I)\} > N)] \Rightarrow$$
$$[\forall IS'.\ IS' = (A',R') \text{ terminaler Nachfolger von IS.}$$
$$(\forall u.\ u \text{ eindeutige Markierung für } A' \Rightarrow UL(A',u) > L)]$$

Beweis:

Es seien die in der Beh. angegebenen Voraussetzungen erfüllt.

Widerspruchsannahme: IS' = (A',R') sei terminaler Nachfolger von IS; u sei eindeutige Markierung für A', UL(A',u) $\leq$ L.

Dann gilt:

(i) $u_{|V_A}$ ist eindeutige Markierung für A [Satz 4.6 (5)]

(ii) $\forall K \in V_A(\forall K' \in d(K).\ u_{|V_A}(K) \neq u_{|V_A}(K'))$ [Satz 4.6 (6)]

(iii) $\sum_{I \in IM} \max\{Lb(I),UL(A,u_{|V_A},I)\} \leq UL(A',u)-Lbr$ [Satz 4.6 (7)]

$\leq L - Lbr = N$ [Voraussetzung]

im Widerspruch zur Voraussetzung.

4.3.2 Instruktionshypothesenreduktion

In Satz 4.8 wird gezeigt, daß jede auf einer IH-Situation IS' aufbauende Hypothesenbildung, die zu einem Programm der Länge L führt, nur Elemente aus der relevanten Instruktionshypothesenmenge bzgl. der LB-Situation von IS und L verwendet, wenn IS' ein Nachfolger von IS ist.

Satz 4.8

Q sei PSP; IS,IS' seien aus G = IH-GRAPH(Q) ableitbare IH-Situationen, für die eine terminale Nachfolgersituation existiert. IS' sei Nachfolger von IS, LB = LB-SIT(G,IS), L $\in \mathbb{N}$.

Behauptung:

$$\forall IS''.\ IS'' = (A,R) \text{ ist terminaler Nachfolger von IS'}$$
$$(\forall u.\ u \text{ ist eindeutige Markierung für } A.$$
$$UL(A,u) \leq L \Rightarrow INSTR(IS'') \subseteq IH\text{-}SET(LB,L))$$

Beweis:

Ist LB = ε, so gibt es keinen terminalen Nachfolger von IS.

Sei also LB = (Lmin,Lbr,Lb,d,IM,l).

Ist l < L, so gilt der Satz wegen IH-SET(LB,L) = INSTRUCTIONS.

Sei daher l $\geq$ L, woraus IH-SET(LB,L) = IM folgt.

Weiterhin sei IS'' = (A,R) ein terminaler Nachfolger von IS', u sei eindeutige Markierung für A, UL(A,u) $\leq$ L.

Dann gilt:

$$L \leq l = \sum_{I \in IM} Lb(I) \qquad \text{[Vor. u. Def. 4.4]}$$

$$\leq \sum_{I \in IM} UL(A,u,I) \qquad \text{[Satz 4.6 (3)]}$$

$$\leq \sum_{I \in INSTRUCTIONS} UL(A,u,I)$$

$$= UL(A,u) \leq L \qquad \text{[Def. 3.13 u. Vor.]}$$

und damit auch die Gleichheit der angeführten Werte.

Aus der Tatsache, daß

$$\forall I \in INSTRUCTIONS \setminus IM.\ UL(A,u,I) = 0$$

gilt, folgt nun direkt INSTR(IS") $\subseteq$ IM.

Mit Hilfe von Satz 4.8 wird nun folgende Möglichkeit der Instruktionshypothesenreduktion bewiesen: Um zu überprüfen, ob eine auf einer IH-Situation IS aufbauende Hypothesenbildung zu einem Programm der Länge L führt, genügt es, lediglich die Hypothesen zu berücksichtigen, die sich aus dem größten erzwungenen Nachfolger von IS gemäß IH-SET(LB,L) ergeben, wenn LB die LB-Situation von IS ist.

Satz 4.9

Q sei PSP; IS aus G = IH-GRAPH(Q) ableitbare IH-Situation, für die eine terminale Nachfolgesituation existiert. Sei LB = LB-SIT(G,IS), LB = (Lmin,Lbr,Lb,d,IM,l), L $\in \mathbb{N}$.

Behauptung:

[$\forall$ IS'. IS' = (A,R) ist terminaler Nachfolger von GSUC(IS,IH-SET(LB,L)).

($\forall$ u. u ist eindeutige Markierung für A => UL(A,u) > L)]

=>

[$\forall$ IS'. IS' = (A,R) ist terminaler Nachfolger von IS.

($\forall$ u. u ist eindeutige Markierung für A => UL(A,u) > L)]

Beweis:

Es seien die in der Beh. angegebenen Voraussetzungen erfüllt.

Widerspruchsannahme: IS' = (A,R) sei terminaler Nachfolger von IS, u sei eindeutige Markierung für A, UL(A,u) $\leq$ L.

Dann gilt:

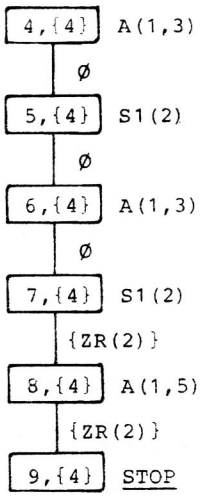
INSTR(IS') $\subseteq$ IH-SET(LB,L) [Satz 4.8]

$\Rightarrow$ IS' ist terminaler Nachfolger von GSUC(IS,IH-SET(LB,L)) [Satz 3.7 (6)]

im Widerspruch zur Voraussetzung.

Beispiel 4.10

Gegeben sei IS = (A,R) wie in Beispiel 4.5-2. Ist L = 7, so genügt es nach Satz 4.9, lediglich die terminalen Nachfolger von GSUC(IS,IH-SET(LB,7)) zu überprüfen. Da IH-SET(LB,7) = IM, ist IS' = (A',R') = GSUC(IS,IM) selbst schon terminal, wobei die Knoten von R in IS' wie folgt markiert sind:



Gibt es für IS' keine eindeutige Markierung u, für die UL(A',u) ≤ 7 gilt, so gibt es überhaupt keinen terminalen Nachfolger von IS, für den dies möglich ist.

4.4 Heuristiken für die Sortierung von Instruktionshypothesen

Die aktuelle LABEL-Hypothese eines IH-Anfangsgraphen wird dazu benützt, die als nächstes aufzustellende Instruktionshypothese auszuwählen (4.4.1); darüberhinaus werden alle an einer Front relevanten Instruktionshypothesen heuristisch geordnet, nachdem die als nicht relevant erkannten Hypothesen entfernt worden sind (4.4.2, 4.4.3). In 4.4.4 werden die Ergebnisse zusammengefaßt und ihre hier wesentlichen Eigenschaften bewiesen.

4.4.1 Vorschlagshypothese

Ist für einen Anfangsgraphen A einer IH-Situation IS = (A,R) eine eindeutige LABEL-Hypothese u aufgestellt worden, so muß im nächsten Schritt eine neue Instruktionshypothese entlang der aktuellen Front F von IS aufgestellt werden. A und u legen aber mit UG = U-GRAPH(A,u) bereits eine Menge von Programmtransitionen fest, die Teilmenge des Zielprogramms ist (bzw. sein kann).

Benutzt man die Vorgängerknoten von F in A zusammen mit den zur Front F führenden Übergängen (das sind die Kantenmarkierungen) und wählt dazu jeweils die sich gemäß den Transitionen von UG ergebenden Nachfolgeinstruktionen, so erhält man die sog. Vorschlagshypothese, wobei an den Stellen, an denen keine zutreffende Programmtransition in UG enthalten ist, die Pseudoinstruktion dummy eingesetzt wird. Da u eindeutige Markierung für A ist, ist damit die Vorschlagshypothese in jeder Komponente eindeutig definiert.

Beispiel 4.11

Für IS = (A,R) aus Beispiel 4.5-2 gilt:

FRONT(A) = ([4,{4}]).

u = λK. K∈{[4,{1}],[5,{1}]} —> 2, true —> 1

ist eindeutige LABEL-Hypothese für A.

Der zu [4,{4}] gehörige Vorgängerknoten ist [3,{2,4}], der zugehörige Übergang ist t_{G_4} (([3,{2,4}],[4,{4}])) = Ø .

In U-GRAPH(A,u) gibt es eine Programmtransition

t_r = ((u([3,{2,4}]),a([3,{2,4}])), t_{G_4}(([3,{2,4}],[4,{3}])),X)
= ((1, S1(2)),Ø,(2,A(1,3)))

Gemäß LABEL-Hypothese u und Übergangsmarkierung Ø ergibt sich als Vorschlagshypothese VH für F = ([4,{4}]) somit VH = (A(1,3)).

4.4.2 Hypothesenreduktion

Nur der Teil der an einer aktuellen Front F anliegenden Instruktionshypothesen ist für die weitere Hypothesenbildung zu berücksichtigen, der in der relevanten IH-Menge bzgl. der aktuellen LB-Situation und dem aktuellen Suchraumindex L enthalten ist. Dieses wurde bereits in Satz 4.8 formal bewiesen.

Beispiel 4.11 (Fortsetzung)

Sei LB = LB-SIT(G_4,IS).

Für L = 9 gibt es an der Front F wegen IH-SET(LB,9)=INSTRUCTIONS drei in Frage kommende Hypothesen: (A1(1)), (A(1,1)) und (A(1,3)). Für L = 7 ist wegen IH-SET(LB,L) = IM nur die Hypothese (A(1,3)) relevant.

4.4.3 Ordnungskriterien

Die vorliegenden Instruktionshypothesen der Front F werden nach einer bestimmten Heuristik linear geordnet; in dieser Reihenfolge werden die Hypothesen aufgestellt und überprüft, ob sie zum Erfolg führen. Die erste Hypothese unter diesem Ordnungskriterium ist die Vorschlagshypothese, bei der zuvor, falls notwendig, die dummy-Komponenten durch Instruktionen ersetzt worden sind.

Die Sortierung der übrigen Instruktionshypothesen basiert auf der Häufigkeit, mit der die einzelnen Instruktionen I unter der bisherigen Hypothesenbildung auftreten. Als Kriterium der Häufigkeit wird hierbei die Mindestanzahl von Programmknoten mit Instruktion I herangezogen, die durch Lb(I) gegeben ist, wobei Lb durch die aktuelle LB-Situation geliefert wird. Gilt für zwei Instruktionen I und I' Lb(I) = Lb(I'), so erfolgt die weitere Sortierung durch die lineare Ordnung, die auf INSTRUCTIONS definiert ist. Gilt Lb(I) > Lb(I'), so wird Hypothese I als wahrscheinlicher angesehen. Für I gibt es dann nämlich mehr Möglichkeiten, den betreffenden Knoten einem Programmknoten zuzuordnen, der sowieso schon benötigt wurde, während für I' eventuell eher ein neuer Programmknoten zu den bisher benötigten hinzukommt. Für Lb(I') = 0 ist es z.B. möglich, daß ein minimales Programm P überhaupt keinen I'-Knoten enthält, während bei Auswahl von I' als Hypothese jedes daraus resultierende Programm mindestens einen I'-Knoten besitzt.

Die beschriebene lineare Ordnung wird durch komponentenweise Anwendung zu einer lexikographischen Ordnung über dem Raum der Instruktionshypothesen erweitert.

Beispiel 4.11 (Fortsetzung)

Es gilt:

VH = (A(1,3))

I	A1(1)	A(1,1)	A(1,3)
Lb(I)	0	0	1

Als lineare Ordnung der Hypothesen ergibt sich:

$(A(1,3)) \lhd (A1(1)) \lhd (A(1,1))$

Dieselbe Ordnungsrelation ergäbe sich, falls die Vorschlagshypothese VH = (dummy) wäre, da

$Lb(A(1,3)) > Lb(A1(1))$

$Lb(A(1,3)) > Lb(A(1,1))$

$Lb(A1(1)) = Lb(A(1,1))$ und $A1(1) <_I A(1,1)$,

wobei $<_I$ die auf INSTRUCTIONS definierte Ordnung ist.

4.4.4 Instruktionshypothesenraum

Die Überlegungen dieses Abschnitts werden in der folgenden Definition präzisiert.

Definition 4.5

Q sei PSP; IS = (A,R) = $((V_A,E_A,a,c),(V_R,E_R,f_R,g_R,t_R))$
sei aus G = (b,V_G,E_G,f_G,g_G,t_G) = IH-GRAPH(Q) ableitbare, nichtterminale IH-Situation, für die eine terminale Nachfolgersituation existiert.
Sei $FRONT_G(A) = (K_1,\ldots,K_r)$, $FRONTE_G(A) = ((K_1',K),\ldots,(K_r',K_r))$
LB = (Lmin,Lbr,Lb,d,IM,l) = LB-SIT(G,IS).
UM = (UG,u) sei eindeutiges Markierungspaar für A,
$L \in \mathbb{N}$, $<_I$ lineare Ordnung auf INSTRUCTIONS.

(1) $VH = (vh_1,\ldots,vh_r)$ heißt IH-Vorschlagshypothese von UM für IS <=>

$\forall i \in \{1,\ldots,r\}$.

$$vh_i = \begin{cases} I & <=> \exists\, X \in \text{U-NODES}(A,u), n \in \mathbb{N} \\ & \quad ((u(K_i'),a(K_i'),t_G^{a(K_i')}(K_i',K_i),X) \in UG \\ & \quad \wedge\, I = (n,I)) \\ dummy & <=> \text{sonst} \end{cases}$$

(2) $HR = (H_{i,j})_{\substack{1\le i\le r \\ 1\le j\le s_r}}$ heißt gemäß UM,LB und L reduzierter und vorsortierter I-Hypothesenraum von IS <=>

$\forall i \in \{1,\ldots,r\}$.

(i) $\{H_{i,1},\ldots,H_{i,s_i}\} = f_R(K_i) \wedge \text{IH-SET}(LB,L)$

(ii) $vh_i \in f_R(K_i) \Rightarrow (H_{i,1} = vh_i) \wedge$
$(\forall j \in \{2,\ldots,s_i-1\}.\ Lb(H_{i,j}) > Lb(H_{i,j+1})$
$\vee (Lb(H_{i,j}) = Lb(H_{i,j+1}) \wedge H_{i,j} <_I H_{i,j+1}))$

(iii) $vh_i \notin f_R(K_i) \Rightarrow (\forall j \in \{1,\ldots,s_i-1\}.\ Lb(H_{i,j}) > Lb(H_{i,j+1})$
$\vee (Lb(H_{i,j}) = Lb(H_{i,j+1}) \wedge H_{i,j} <_I H_{i,j+1}))$

(3) $H = (H_{1,j_1},\ldots,H_{r,j_r})$ heißt Hypothese aus HR ($H \varepsilon HR$) <=>

$\forall i \varepsilon \{1,\ldots,r\}.\ 1 \leq j_i \leq s_i$

(4) HMIN(HR) = $(H_{1,1},\ldots,H_{r,1})$ heißt kleinste,

HMAX(HR) = $(H_{1,s_1},\ldots,H_{r,s_r})$ heißt größte Hypothese von HR.

(5) Sei H wie in (3), $H \neq$ HMAX(HR).

HNEXT(HR,H) = H' heißt direkter Nachfolger von H in HR

($H <_{dir} H'$) <=>

$H' = (H_{1,j},\ldots,H_{t-1,j_{t-1}},H_{t,j_t+1},H_{t+1,1},\ldots,H_{r,1})$

mit: $t = \max\{t' \mid t' \varepsilon \{1,\ldots,r\},\ j_{t'} < s_{t'}\}$

(6) $<_{HR}$ heißt Ordnungsrelation von HR und bezeichnet die transitive Hülle der Relation $<_{dir}$.

(7) SORTHR(G,L,IS,LB,UM) bezeichnet den durch (1) - (6) definierten I-Hypothesenraum.

Beispiel 4.11 (Fortsetzung)

1. Für L = 9 gilt:

$$HR = \begin{pmatrix} A(1,3) \\ A1(1) \\ A(1,1) \end{pmatrix}$$

HMIN(HR) = (A(1,3))

HMAX(HR) = (A(1,1))

HNEXT(HR,(A(1,3))) = (A1(1))

2. Für L = 7 gilt:

HR = (A(1,3))

HMIN(HR) = HMAX(HR) = (A(1,3))

Satz 4.10

Seien Q,G,IS,LB,UM,L wie in Def. 4.5 .

Sei HR = SORTHR(G,L,IS,LB,UM), es gelte $L \geq Lmin$.

Behauptung:

(1) HR ist eindeutig bestimmt.

(2) HMIN(HR) und HMAX(HR) sind wohldefiniert.

(3) Für $H \varepsilon HR, H \neq$ HMAX(HR), ist HNEXT(HR,H) eindeutig bestimmt.

(4) $<_{HR}$ ist lineare Ordnung aller $H \varepsilon HR$.

(5) [$\forall H \varepsilon HR$

($\forall$ IS' = (A',R'). IS' ist terminaler Nachfolger von DSUC(IS,H)

($\forall$u. u ist eindeutige Markierung für A' => UL(A',u) > L))]

=>

[$\forall$ IS' = (A',R'). IS' ist terminaler Nachfolger von IS

($\forall$u. u ist eindeutige Markierung für A' => UL(A',u) > L)]

Beweis:

(1) - (4) ergeben sich leicht aus den entsprechenden Definitionen. Die Eindeutigkeit der HR mit-definierenden IH-Vorschlagshypothese folgt aus der Tatsache, daß UM eindeutiges Markierungspaar für A ist. Mit Hilfe von Satz 3.7 (7) kann gezeigt werden, daß HR nicht leer ist.

(5) Es seien die in der Beh. (5) angegebenen Voraussetzungen erfüllt.

Widerspruchsannahme: $\exists$ IS' = (A',R'), u .

(IS' ist terminaler Nachfolger von IS $\wedge$

u ist eindeutige Markierung für A' $\wedge$ UL(A',u) $\leq$ L)

Dann gilt aber:

($\exists$ H $\in$ FIH_G(IS).

(IS' ist term. Nachfolger von DSUC(IS,H)) [Satz 3.7(5)]

$\wedge$ (INSTR(IS') $\subseteq$ IH-SET(LB,L)) [Satz 4.8]

$\wedge$ ($\forall$ H $\in$ FIH_G(IS). H$\in$(IH-SET(LB,L)) $^{r}\Rightarrow$ H$\in$HR) [Def.4.5(2)]

$\Rightarrow$ $\exists$ H$\in$HR. IS' ist term. Nachfolger von DSUC(IS,H)

im Widerspruch zur Voraussetzung

Satz 4.10 liefert die Grundlage für die i.f. skizzierte Vorgehensweise zur Überprüfung der an der Front von IS vorliegenden Instruktionshypothesen:

Zunächst wird der Instruktionshypothesenraum HR = SORTHR(G,L,IS,LB,UM) berechnet. Beginnend mit H = HMIN(HR), gefolgt von H' = HNEXT(HR,H), H" = HNEXT(HR,H') usw. werden DSUC(IS,H), DSUC(IS,H') usw. überprüft, bis eine Hypothese zum Erfolg führt oder bis HMAX(HR) erreicht ist und auch diese Hypothese nicht erfolgreich ist. Ist letzteres der Fall, so sind damit alle terminalen Nachfolger von IS als nicht mehr relevant erkannt und man muß zur vorherigen Front (nach 'links') zurückkehren.

4.5 Programmkonstruktion aus IH-Situationen

Die Anforderungen an einen Algorithmus zur Bestimmung einer eindeutigen Markierung (LABEL-Hypothese) für einen Anfangsgraphen werden in 4.5.1, die an einen Algorithmus zur Bildung von Programmtransitionen aus vorliegender Instruktions- und LABEL-Hypothese in 4.5.2 angegeben. In 4.5.3 wird gezeigt, wie die bisher entwickelten Verfahren tatsächlich zu einer minimalen Realisierung von PSP führen.

4.5.1 LABEL-Hypothesenbildung

Bei den in diesem Kapitel entworfenen Vorgehensweisen zur Hypothesenreduktion sowohl auf der Ebene der Instruktions- als auch der LABEL-Hypothesen wurden lediglich LABEL-Hypothesen für Anfangsgraphen A von IH-Situationen IS = (A,R) benötigt, die die durch die aktuelle LB-Situation gegebenen Informationen berücksichtigen, die in den Sätzen 4.5, 4.6 und 4.7 zum Ausdruck kommen. Diesem wird in der folgenden Definition Rechnung getragen.

Zunächst wird die Menge der U-Argumente bestimmt. Ein U-Argument wird aus den drei Komponenten Suchraumindex L, IH-Situation IS und zugehöriger LB-Situation LB gebildet und ist ein 6-Tupel, das alle für die LABEL-Hypothesenbildung notwendigen Informationen enthält. Zur Vereinfachung wird die Argumentmenge eines U-Algorithmus anschließend auf die Menge der 6-Tupel reduziert, die auch tatsächlich auf diese Art und

Weise gebildet werden können. Ein U-Algorithmus ist nun eine Abbildung UA in die Menge aller eindeutigen Markierungspaare, wobei U-DUMMY geliefert wird genau dann, wenn es keine eindeutige Markierung u für A gibt, unter der durch d unterschiedene Knoten auch verschiedene LABEL haben und unter der die Summe aus den jeweiligen Maxima von Mindestanzahl von I-Knoten bzgl. IS und tatsächlicher Anzahl von I-Knoten bei A und u kleiner oder gleich ist als die Differenz von Suchraumindex L und Mindestanzahl der noch für den Restgraphen R zusätzlich benötigten Knoten (vgl. Sätze 4.6 und 4.7); andernfalls liefert UA eine solche eindeutige Markierung u zusammen mit U-GRAPH(A,u).

Definition 4.6

(1) Q sei PSP, IS = (A,R) sei aus G = IH-GRAPH(Q) ableitbare IH-Situation, für die ein terminaler Nachfolger existiert; LB = LB-SIT(G,IS), $L \in \mathbb{N}$.
U-ARG(L,IS,LB) = (N,A,Lb,d,IM,l) heißt
U-Argument von L, IS und LB <=>
LB = (Lmin,Lbr,Lb,d,IM,l) mit: N = L - Lbr

(2) U = (N,A,Lb,d,IM,l) heißt relevantes U-Argument <=>
$\exists$ Q,L,IS,LB wie in (1) mit:
U = U-ARG(L,IS,LB) $\wedge$ N $\geq$ Lmin-Lbr

(3) U-ARGS bezeichnet die Menge aller relevanten U-Argumente.

(4) U-VAL bezeichne die Menge aller eindeutigen Markierungspaare.
Eine Abbildung UA : U-ARGS —> U-VAL heißt U-Algorithmus <=>
$\forall X \in$ U-ARGS. X = (N,A,Lb,d,IM,l), A = (V_A, E_A, a, c) .
UA(X) = (UG,u) ist eindeutiges Markierungspaar von A
mit:

$$[UG \neq \emptyset \Rightarrow (\forall K \in V_A (\forall K' \in d(K).\ u(K) \neq u(K')) \wedge \sum_{I \in IM} \max\{Lb(I), UL(A,u,I)\} \leq N)]$$

$$[UG = \emptyset \Rightarrow (\forall u.\ u \text{ eindeutige Markierung für } A \wedge \forall K \in V_A (\forall K' \in d(K).\ u(K) \neq u(K')) \Rightarrow \sum_{I \in IM} \max\{Lb(I), UL(A,u,I)\} > N)]$$

Die an einen U-Algorithmus gestellten Forderungen sind zwar prinzipiell ausreichend, jedoch läßt sich die Anzahl der möglichen LABEL-Hypothesen erheblich reduzieren. So kann o.B.d.A. gefordert werden:

Sind in U-GRAPH(A,u) k Knoten mit der Instruktionsmarkierung I, so sollen für diese Knoten genau die LABEL 1,2,...,k verwandt werden.

In den folgenden Ausführungen wird deutlich, wie die 6 Werte eines U-Arguments von einem U-Algorithmus benutzt werden können. Sei X = (N,A,Lb,d,IM,l) ein relevantes U-Argument, wobei IS = (A,R) terminal sei; ist IS nicht terminal, so gelten die anschließenden Überlegungen ebenso, es muß dann jedoch noch zwischen Knoten aus A und R unterschieden werden. Es soll eine Markierung u gefunden werden, unter der

(*) $\sum_{I \in IM} \max\{Lb(I), UL(A,u,I)\} \leq N$

gilt und unter der durch L unterschiedene Knoten auch unterschiedliche LABEL haben. Nun gilt aber $l = \sum_{I \in IM} Lb(I)$ und wegen $l \leq Lmin-Lbr = N$ auch $l \leq N$.

Falls $l = N$, so kommen für einen Knoten K mit Instruktion I genau die LABEL $1,2,\ldots,Lb(I)$ in Frage, falls $l < N$ darüber hinaus maximal noch $Lb(I)+1$, $Lb(I)+2,\ldots,Lb(I)+(N-l)$; wenn jedoch schon für ein anderes I' die LABEL $1,2,\ldots,Lb(I')$, $Lb(I')+1,\ldots,Lb(I')+k$ $(1 \leq k \leq N-l)$ verwandt worden sind, so stehen für I nur noch die LABEL $1,2,\ldots,Lb(I),Lb(I)+1,\ldots$ $\ldots,Lb(I)+(N-(l+k))$ zur Verfügung. Diese gegenseitige Abhängigkeit besteht zwischen allen $I \in IM$; wird sie nicht beachtet, kann auch die obige Relation (*) nicht mehr erfüllt werden.

Ein U-Algorithmus sollte somit u.a. über folgende Größen Buch führen:

1. Für jedes $I \in IM$: Wieviele LABEL sind bereits für I-Knoten verwandt worden? Jedesmal, wenn bereits Lb(I) oder mehr verschiedene LABEL verwandt worden sind und noch ein zusätzliches LABEL hinzugenommen werden soll, muß der unter 2. genannte Wert F um 1 dekrementiert werden.
2. Wie oft kann noch irgendeiner Instruktion I ein "zusätzliches" LABEL zugeordnet werden? Bezeichnet F den jeweils aktuellen Wert dieses Parameters, so zeigt der Versuch, F einen Wert kleiner 0 zuzuweisen, einen Fehler an und es muß eine andere LABEL-Hypothese gewählt werden. F wird mit N-l initialisiert.

3. Für jeden Knoten K von A:
   Welche LABEL dürfen K nicht zugeordnet werden, da bereits ein K' mit $K \in d(K')$ dieses LABEL hat?

Das hier skizzierte Verfahren hat Ähnlichkeiten mit dem in [Biermann,Baum,Petry 1976] beschriebenen Ansatz. Jedoch werden in der zitierten Arbeit nur eine bestimmte Art von Instruktionsfolgen, bei denen alle erfolgreich ausgeführten Tests explizit angegeben sind, berücksichtigt, und das in dem Papier beschriebene Verfahren ist dort nur als sequentiell für jeweils genau eine Instruktionsfolge anwendbar ausgearbeitet. In [Hwa,Wrightson 1973] ist das Verfahren aus [Biermann 1972] in ähnlicher Weise für die parallele Bearbeitung von mehreren Berechnungsfolgen einer Turingmaschine erweitert und benützt worden.

In der vorliegenden Arbeit jedoch ist die Bearbeitung jeder der vier dargestellten Arten von sowohl Instruktions- als auch Speicherbelegungsfolgen (insbesondere letztere wurden bisher in den anderen Arbeiten überhaupt nicht betrachtet) möglich, wobei die parallele Bearbeitung von mehreren Beispielsfolgen in allen vier Fällen gewährleistet wird und dadurch sowie durch die Aufteilung der Speicherbelegungsfolgen in Teilfolgen, für die Hypothesen bereits überprüft werden können, schärfere Randbedingungen (Ein/Ausgabebedingungen für einen U-Algorithmus) geschaffen und eine bessere Ausnützung der zuvor erarbeiteten und nun zur Verfügung stehenden Informationen unterstützt werden.

An zwei einfachen Beispielen soll die Vorgehensweise eines U-Algorithmus verdeutlicht werden.

Beispiel 4.12

1. Sei A der in Beispiel 3.11-2 angegebene Anfangsgraph A_4, N = 9, l = 8 und X = (N,A,Lb,d,IM,l) mit:

I	S1(2)	CL(1)	A(1,3)	A(1,4)	A(1,5)	ZR(2)	sonst.I
Lb(I)	1	1	2	1	1	2	0

K	[1,{1,2,3}]	[2,{1,2,3}]	[3,{1,2,3}]	[4,{1,2,3}]	[5,{1,2,3}]	[6,{2,3}]	[6,{1}]
d(K)	∅	[5,{1,2,3}]	[6,{1}]	∅	[2,{1,2,3}]	∅	[3,{1,2,3}]

Es können F = N - l = 1 'zusätzliche' LABEL verwandt werden.

(1) [0,{1,2,3}] wird immer LABEL 1 zugeordnet (START-Knoten).

(2) Für die Knoten [1,{1,2,3}] bis [4,{1,2,3}] gilt jeweils, daß die ihnen zugeordnete Instruktion das erste Mal auftritt; alle diese Knoten enthalten LABEL 1. Bei [2,{1,2,3}] bzw. |3,{1,2,3}| wird festgestellt, daß [5,{1,2,3}] bzw. [6,{1}] ein anderes LABEL zugeordnet werden muß, da sie durch d unterschieden werden.

(3) Da bisher nur LABEL 1 für ZR(2)-Knoten (bzw. A(1,3)-Knoten) verwandt wurde, 1 aber für [5,{1,2,3}] bzw. |6,{1}| verboten ist, erhalten diese beiden Knoten zwangsläufig LABEL 2.

(4) A(1,5) tritt zum ersten Mal auf, [6,{2,3}] erhält LABEL 1.

(5) [7,{2,3}] kann als STOP-Knoten nur LABEL 1 zugewiesen werden.

Die in Def. 4.6 geforderten Nachbedingungen sind erfüllt.

2. Sei IS = (A,R) die in Beispiel 3.11-3 angegebene terminale IH-Situation IS_5; LB = LB-SIT(G_3,IS) ist identisch mit der in Beispiel 4.7 angegebenen LB-Situation LB'.
Sei L = 9.
U-ARG(L,IS,LB) = (N,A,Lb,d,IM,l)
= (9,A,Lb,d,IM,9)
Da F = N - l = 0, können nur LABEL bis Lb(I) verwandt werden. IS ist Nachfolger der unter 1. betrachteten IH-Situation, die Schritte (1) bis (5) verlaufen hier analog.

(6) [7,{1}] kann zur durch Funktion d erzwungenen Unterscheidung von [4,{1,2,3}] nur LABEL 2 zugeordnet werden.

(7) Für ZR(2) sind bereits LABEL 1 und 2 benutzt worden; da Lb(ZR(2)) = 2 und F = 0, kommen auch nur diese beiden LABEL für [8,{1}] in Frage, wobei 2 ausscheidet, da [8,{1}] ∈ d ([5,{1,2,3}]) und u([5,{1,2,3}]) = 2 . Bleibt als einzige Alternative u([8,{1}]) = 1.

(8) [9,{1}] und [10,{1}] kann nur LABEL 1 zugeordnet werden.

Für UM = (UG,u) = U-ALG(U-ARG(L,IS,LB)) gilt damit: UG ist identisch mit dem in Beispiel 2.2 angegebenen Programm P. Die für U-ALG geforderte Nachbedingung ist erfüllt.

Die beiden Beispiele zeigen, daß bei optimaler Ausnutzung der mit einem U-Argument X = (N,A,Lb,d,IM,l) gegebenen Informationen die Auswahl der LABEL-Hypothesen durch einen U-Algorithmus sehr vereinfacht werden kann. In diesen Beispielen ist es sogar so, daß an keiner einzigen Stelle eine andere

LABEL-Hypothese gewählt werden kann und N,A,Lb,d,IM und l somit zwangsläufig zu einer bestimmten Hypothese führen. I.a. wird dies zwar nicht immer so sein; die Reduzierung der an jedem Punkt zur Verfügung stehenden Alternativen verhindert jedoch weitgehend ein unnötiges Aufstellen und späteres Verwerfen von falschen Hypothesen.

4.5.2 Programmtransitionenbildung

Ist für eine terminale IH-Situation IS = (A,R) eine eindeutige Markierung u gefunden worden, so besteht UG = U-GRAPH(A,u) aus Transitionen der Art (K,m,K') mit Programmknoten K und K'. Für den Fall $b \in \{\underline{s},\underline{ss}\}$ ist $m \in \{\underline{true},\underline{false},\underline{nil}\}$ und das zu synthetisierende Programm P ist identisch mit UG; für $b \in \{\underline{a},\underline{as}\}$ ist $m \subseteq$ TESTS und (K,m,K') muß noch in Programmtransitionen überführt werden.

Die Überführung in eine Menge P von Programmtransitionen muß folgende Bedingungen erfüllen:

1. P ist ein Programm.
2. Die Menge der mit Zuweisungsinstruktionen markierten Programmknoten von P ist genau die Menge der Programmknoten von UG.
3. Für jedes Tripel (K,m,K') in UG existiert in P ein Weg von K nach K' derart, daß zwischen K und K' nur Testknoten durchlaufen werden und die dazugehörigen Übergänge genau dann <u>true</u> sind, wenn die durchlaufene Testinstruktion in m enthalten ist, andernfalls <u>false</u>.

Als Argument für einen derartigen Überführungsalgorithmus werden genau alle die Paare aus Problemspezifizierungsmarke b und U-Graph UG zugelassen, die bei der bisher beschriebenen Vorgehensweise auch tatsächlich auftreten können.

<u>Definition 4.7</u>

(1) Q = (b,M) sei PSP, IS = (A,R) sei aus IH-GRAPH(Q) ableitbare terminale IH-Situation; UM = (UG,u) sei eindeutiges Markierungspaar für A, UM ≠ U-DUMMY.
T-ARG(Q,UM) = (b,UG) heißt <u>T-Argument von Q und UM</u>.

(2) X = (b,UG) heißt <u>relevantes T-Argument</u> <=>
$\exists$ Q,UM wie in (1) mit: X = T-ARG(Q,UM)

(3) <u>T-ARGS</u> bezeichnet die Menge aller relevanten T-Argumente.

(4) Eine Abbildung TA : T-ARGS —> PROGRAMS heißt
<u>T-Algorithmus</u> <=>

$\forall X \in$ T-ARGS. X = (b,UG)
$[b \in \{\underline{s},\underline{ss}\} \Rightarrow TA(X) = UG]$
$\wedge [b \in \{\underline{a},\underline{as}\} \Rightarrow$
$(NODES(UG) = NODES(TA(X)) \wedge (\mathbb{N} \times ASSIGNMENTS^{1}))$
$\wedge (\forall (K,m,K') \in UG. \exists K_0,\ldots,K_{r+1} \in NODES(TA(X)), r \geq 0.$
$(K = K_0 \quad K' = K_{r+1})$
$\wedge ((K_0,\underline{nil},K_1) \in TA(X))$
$\wedge (\forall i \in \{1,\ldots,r\}. INSTR(K_i) \in TESTS$
$\wedge (INSTR(K_i) \in m \Rightarrow (K_i,\underline{true},K_{i+1}) \in TA(X))$
$\wedge (INSTR(K_i) \notin m \Rightarrow (K_i,\underline{false},K_{i+1}) \in TA(X))))]$

Die an einen T-Algorithmus gestellten Anforderungen beruhen wesentlich darauf, daß u eine eindeutige Markierungsfunktion ist (vgl. Def. 3.13) :
sind in UG zwei Tripel in den ersten beiden Komponenten gleich, so sind sie es auch in der dritten Komponente, oder anders ausgedrückt: muß es nach obiger Bedingung von einem Knoten K

zwei Wege zu verschiedenen Knoten K' und K" geben, so kann dies wie gefordert realisiert werden, da es mindestens eine Testinstruktion gibt, die die Übergänge in UG von K nach K' bzw. K" unterscheidet.

Beispiel 4.13

Mit der Stufengraphdarstellung der Realisierung von Q_2 durch das Beispielprogramm P ist sowohl eine aus IH-GRAPH(Q_2) ableitbare terminale IH-Situation wie auch eine eindeutige Markierungsfunktion gegeben (siehe Beispiel 3.4 -1). Der resultierende U-Graph UG ist:

UG = { ((1,START),{ZR(1)},(1,CL(1))),
((1,START),{ZR(2)},(1,Cl(1))),
((1,START),Ø, (1,CL(1))),
((1,CL(1)),{ZR(1)},(1,A(1,3))),
((1,CL(1)),{ZR(1),ZR(2)},(1,A(1,4))),
((1,A(1,3)),Ø, (1,S1(2))),
((1,S1(2)),{ZR(2)},(1,A(1,5))),
((1,S1(2)),Ø, (2,A(1,3))),
((1,A(1,5)),{ZR(2)},(1,STOP)),
((2,A(1,3)),Ø, (2,S1(2))),
((2,S1(2)),{ZR(2)},(1,A(1,4))),
((1,A(1,4)),{ZR(2)},(1,STOP)) }

Da ein Zuweisungsknoten in einem Programm nur einen Ausgang haben darf, müssen zwischen den Knoten mit mehreren Ausgängen in UG und den Nachfolgeknoten geeignete Tests eingeführt werden. Am einfachsten ist dies zu lösen, indem man

für jeden Knoten K die Einfügungen unabhängig von den anderen vornimmt, d.h.:

(1) (1,START) hat als einzigen Nachfolger (1,CL(1)).
Es genügt also, für die drei Tripel in UG die Programmtransition
(i) ((1,START),nil,(1,CL(1)))
in das zu synthetisierende Programm P' aufzunehmen.

(2) (1,CL(1)) hat zwei Nachfolger. Ein Vergleich der beiden Kantenmarkierungen {ZR(1)} und {ZR(1),ZR(2)} zeigt, daß nur die Testinstruktion ZR(2) beide Wege unterscheiden kann. Da ZR(2) in der zu (1,A(1,3)) führenden Kantenmarkierung nicht enthalten ist, muß vom einzufügenden Testknoten der false-Ausgang nach (1,A(1,3)) führen, der true-Zweig analog zu (1,A(1,4)). Die Programmtransititonen sind:
(ii) ((1,CL(1)),nil,(1,ZR(2)))
(iii) ((1,ZR(2)),true,(1,A(1,4)))
(iv) ((1,ZR(2)),false,(1,A(1,3)))

(3) (1,A(1,3)) hat nur einen Nachfolger; P' enthält:
(v) ((1,A(1,3),nil,(1,S1(2)))

(4) (1,S1(2)) hat zwei Nachfolger; analog zu (2) folgt, daß eine Testinstruktion ZR(2) eingefügt werden muß. Da die Einfügungen der Testknoten für jeden Knoten unabhängig von anderen vorgenommen werden sollen, wird durch Benutzung der Markierung 2 ein neuer Testknoten erzeugt:
(vi) ((1,S1(2)),nil,(2,ZR(2)))
(vii) ((2,ZR(2)),true,(1,A(1,5)))
(viii) ((2,ZR(2)),false,(2,A(1,3)))

(5) Für (2,A(1,3)),(2,S1(2)),(1,A(1,4)) und (1,A(1,5)) ergibt sich analog zu (3):

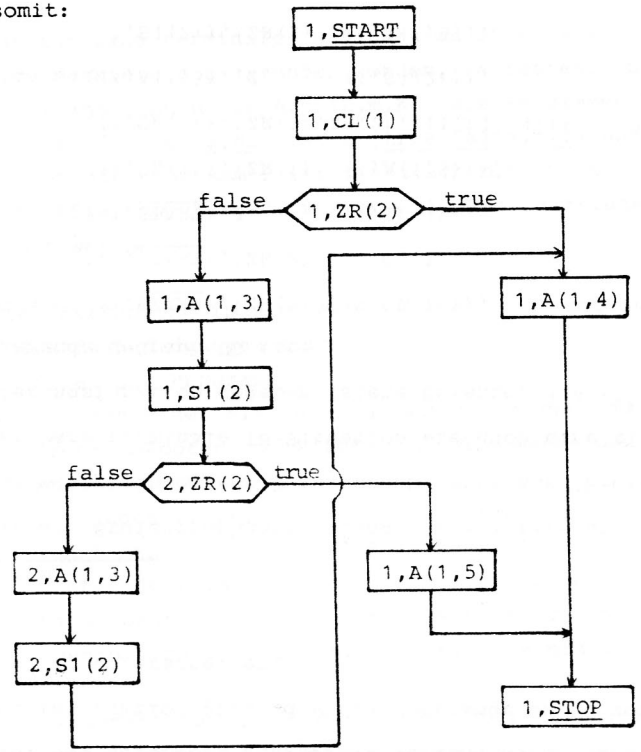
(ix) ((2,A(1,3)),nil,(2,S1(2)))

(x) ((2,S1(2)),nil,(1,A(1,4)))

(xi) ((1,A(1,4)),nil,(1,STOP))

(xii) ((1,A(1,5)),nil,(1,STOP))

Als ein minimales, Q_2 realisierendes Programm P' erhält man somit:



4.5.3 Nachweis der Realisierungseigenschaften

In Satz 4.11 wird gezeigt, daß ein T-Algorithmus angewandt auf einen U-Graph, der aus einer aus IH-GRAPH(Q) ableitbaren terminalen IH-Situation IS = (A,R) und einer eindeutigen Markierung u für A entsteht, ein Programm P liefert, das Q realisiert.

Satz 4.11

Q = (b,M) sei PSP, dim(Q) = $(m,(n_1,\ldots,n_m))$. IS = (A,R) sei aus G = IH-GRAPH(Q) ableitbare terminale IH-Situation, $A = (V_A,E_A,a,c)$. UM = (UG,u) sei eindeutiges Markierungspaar für A, UM ≠ U-DUMMY. T-ALG sei ein T-Algorithmus, P = T-ALG(b,UG). $\{st_1,\ldots,st_m\}$ seien die M entsprechenden Anfangsspeicherbelegungen.

Für $i \in \{1,\ldots,m\}$ sei:

- PATH(A,i) = $(K_{i,0},\ldots,K_{i,n_i})$
- für $j \in \{1,\ldots,n_i\}$ sei:

  $(j = 1)$ $s_{i,1} = st_1$

  $t_{i,0} = \underline{nil}$

  $(j > 1)$ $s_{i,j} = \delta(a(K_{i,j}),s_{i,j-1})$

  $$t_{i,j-1} = \begin{cases} \gamma(a(K_{i,j-1}),s_{i,j-1}) & \text{<=> } b \in \{\underline{s},\underline{ss}\} \\ \underline{nil} & \text{<=> } b \in \{\underline{a},\underline{as}\} \end{cases}$$

Behauptung:

$\forall\, i \in \{1,\ldots,m\}$.

(1) falls $b \in \{\underline{s},\underline{ss}\}$:

$$c(P,st_i) = (u(K_{i,0}),a(K_{i,0})),(t_{i,0},s_{i,1}),(u(K_{i,1}),a(K_{i,1})),\ldots$$

$$\ldots,(t_{n_i-1},s_{n_i}),(u(K_{i,n_i}),a(K_{i,n_i})))$$

(2) falls $b \in \{\underline{a},\underline{as}\}$:

es gibt eindeutig bestimmte $TS_{i,0},\ldots,TS_{i,n_i-1}$ mit:

$$c(P,st_i) = ((u(K_{i,0}),a(K_{i,0})),(t_{i,0},s_{i,1}))\bullet TS_{i,0}\bullet((u(K_{i,1}),a(K_{i,1})),$$

$$(t_{i,1},s_{i,2}))\bullet TS_{i,1}\bullet\ldots\bullet TS_{i,n_i-1}\bullet((u(K_{i,n_i}),a(K_{i,n_i})))$$

(3) Die in (1) bzw. (2) bestimmte Folge $c(P,st_i)$ ist die Berechnungsfolge in Standardbezeichnung von P bei st_i bei Realisierung von Q (vgl. Def. 3.4).

Beweis:

Zunächst werden die für den Fall $b \in \{\underline{a},\underline{as}\}$ benötigten Teilfolgen $TS_{i,j}$ konstruiert; sei $j \in \{0,\ldots,n_i-1\}$. Nach Def. von PATH(A,i) gilt $(K_{i,j},K_{i,j+1}) \in E$.

Sei $c(K_{i,j},K_{i,j+1}) = C$; da UG = U-GRAPH(A,u), ist

$((u(K_{i,j}),a(K_{i,j})),C,(u(K_{i,j+1}),a(K_{i,j+1}))) \in UG$.

Da T-ALG T-Algorithmus ist, folgt:

$$\exists\, x^0_{i,j},\ldots,x^{k(i,j)+1}_{i,j} \in NODES(P),\ k(i,j) \geq 0 \ .$$

mit:

$$(X^0_{i,j} = (u(K_{i,j}),a(K_{i,j})))$$

$$\wedge\ (X^{k(i,j)+1}_{i,j} = (u(K_{i,j+1}),a(K_{i,j+1})))$$

$$\wedge\ ((X^0_{i,j},\underline{nil},X^1_{i,j}) \in P)$$

$$\wedge\ (\forall r \in \{1,\ldots,k(i,j)\}.\ INSTR(X^r_{i,j}) \in TESTS$$

$$\wedge (INSTR(X^r_{i,j}) \in C \Rightarrow (X^r_{i,j},\underline{true},X^{r+1}_{i,j}) \in P)$$

$$\wedge (INSTR(X^r_{i,j}) \notin C \Rightarrow (X^r_{i,j},\underline{false},X^{r+1}_{i,j}) \in P))$$

Da P ein Programm ist, folgt mit der Eindeutigkeitseigenschaft für Programme (s. Kap. 2) darüberhinaus, daß mit dem obigen die $X^0_{i,j},\ldots,X^{k(i,j)+1}_{i,j}$ eindeutig bestimmt sind.

$TS_{i,j}$ ergibt sich damit aus:

$$TS_{i,j} = (X^1_{i,j},(t^1_{i,j},s_{i,j+1}),X^2_{i,j},(t^2_{i,j},s_{i,j+1}),X^3_{i,j},\ldots,X^{k(i,j)}_{i,j},$$

$$(t^{k(i,j)}_{i,j},s_{i,j+1}))$$

wobei für $r \in \{1,\ldots,k(i,j)\}$

$$t^r_{i,j} = \gamma(INSTR(X^r_{i,j}),s_{i,j+1})$$

In $TS_{i,j}$ treten nur Testinstruktionen auf, während alle übrigen Instruktionen, die in $c(P,st_i)$ auftreten, keine Tests sind.

Sowohl im Fall $b \in \{\underline{s},\underline{ss}\}$ als auch im Fall $b \in \{\underline{a},\underline{as}\}$ erfüllen damit die angegebenen Bezeichnungen für $c(P,st_i)$ die Anforderungen für die Standardbezeichnung bei Realisierung von Q. Es bleibt noch zu zeigen, daß in beiden Fällen tatsächlich die vollständige und termininierende Berechnungsfolge von P

bei st_i vorliegt, d.h. im einzelnen (vgl. Def. 2.6):

(i) $(u(K_{i,0}),a(K_{i,0})) = (1,\underline{START})$

(ii) $(t_{i,0},s_{i,0}) = (\underline{nil},st_i)$

(iii) $[\forall j\varepsilon\{2,\ldots,n_i\}.$

$\Gamma(a(K_{i,j-1}),s_{i,j-1}) = (t_{i,j-1},s_{i,j})]$

$\wedge\,[b\varepsilon\{\underline{a},\underline{as}\} \Rightarrow \forall j\varepsilon\{0,\ldots,n_i-1\}.\ \forall r\varepsilon\{1,\ldots,k(i,j)\}.$

$\Gamma(INSTR(X^r_{i,j}),s_{i,j+1}) = (t^r_{i,j},s_{i,j+1})]$

(iv) $\forall j\varepsilon\{0,\ldots,n_i-1\}.$

$[b\varepsilon\{\underline{s},\underline{ss}\} \Rightarrow \Phi(P,(u(K_{i,j}),a(K_{i,j})),t_{i,j}) = (u(K_{i,j+1}),a(K_{i,j+1}))]$

$[b\varepsilon\{\underline{a},\underline{as}\} \Rightarrow \Phi(P,(u(K_{i,j}),a(K_{i,j})),t_{i,j}) = X^1_{i,j}$

$(\forall r\varepsilon\{1,\ldots,k(i,j)\}.\ \Phi(P,X^r_{i,j},t^r_{i,j}) = X^{r+1}_{i,j})]$

(v) $(u(K_{i,n_i}),a(K_{i,n_i})) = (1,\underline{STOP})$

(i) und (v) gelten nach Satz 3.8 (2); (ii) und (iii) gelten nach Voraussetzung und aufgrund der Definition von $s_{i,j}$, $t_{i,j}$, $t^r_{i,j}$. (iv) wird i.f. für die Fälle (1) und (2) getrennt gezeigt.

$b\varepsilon\{\underline{s},\underline{ss}\}$:

(j=0) : $\Phi(P,(u(K_{i,0}),a(K_{i,0})),t_{i,0})$

$= \Phi(P,(u(K_{i,0}),a(K_{i,0})),\underline{nil})$ [Vor.]

$= \Phi(P,(u(K_{i,0}),a(K_{i,0})),c(K_{i,0},K_{i,1}))$ [Satz 3.8 (5)]

$= (u(K_{i,0}),a(K_{i,1}))$ [da P=U-GRAPH(A,u)]

(j>0) : $\Phi(P,(u(K_{i,j}),a(K_{i,j})),t_{i,j})$

$= \Phi(P,(u(K_{i,j}),a(K_{i,j})),\gamma(a(K_{i,j}),s_{i,j}))$ [Vor.]

$= \Phi(P,(u(K_{i,j}),a(K_{i,j})),c(K_{i,j},K_{i,j+1}))$ [Satz 3.8(3)+(5)]

$= (u(K_{i,j+1}),a(K_{i,j+1}))$ [da P=U-GRAPH(A,u)]

$b\varepsilon\{\underline{a},\underline{as}\}$:

Sei $j\varepsilon\{0,\ldots,n_i-1\}$.

$\Phi(P,(u(K_{i,j}),a(K_{i,j})),t_{i,j})$

$= \Phi(P,X^0_{i,j},\underline{nil})$ [Vor.+Def.von $TS_{i,j}$]

$= X^1_{i,j}$

Falls $k(i,j) = 0$, so ist nichts mehr zu zeigen; sei also $k(i,j) > 0$, $r\varepsilon\{1,\ldots,k(i,j)\}$. Es gilt:

$INSTR(X^r_{i,j})\varepsilon TESTS \Rightarrow \gamma(INSTR(X^r_{i,j}),s_{i,j+1})\varepsilon\{\underline{true},\underline{false}\}$

Aus $t^r_{i,j} = \gamma(INSTR(X^r_{i,j}),s_{i,j+1})$ folgt:

$t^r_{i,j} = \underline{true} \Rightarrow INSTR(X^r_{i,j})\varepsilon c(K_{i,j},K_{i,j+1})$ [Satz 3.8(3)+(6)]

$t^r_{i,j} = \underline{false} \Rightarrow INSTR(X^r_{i,j})\notin c(K_{i,j},K_{i,j+1})$ [Satz 3.8(3)+(6)]

$\Rightarrow (X^r_{i,j},t^r_{i,j},X^{r+1}_{i,j}) \varepsilon P$ [Def. von $TS_{i,j}$]

$\Rightarrow \Phi(P,X^r_{i,j},t^r_{i,j}) = X^{r+1}_{i,j}$

Dies schließt den Beweis von Satz 4.11 ab.

Mit Hilfe des Satzes 4.11 kann gefolgert werden, daß unter den gleichen Voraussetzungen das synthetisierte Programm P Q L-realisiert, wenn der vorliegende U-Graph L Knoten enthält.

Satz 4.12

Q = (b,M) sei PSP, G = IH-GRAPH(Q), G ≠ ε;
IS = (A,R) sei aus G ableitbare terminale IH-Situation.
UM = (UG,u) sei eindeutiges Markierungspaar für A,
UM ≠ U-DUMMY, UL(A,u) = L ; T-ALG sei T-Algorithmus.

Behauptung: T-ALG(b,UG) L-realisiert Q .

Beweis:
Aus den Definitionen von L_b und UL ergibt sich L_b(T-ALG(b,UG)) = UL(A,u). Daß T-ALG(b,UG) Q realisiert, wurde in den Sätzen 3.8 und 4.11 gezeigt.

Gibt es keine andere terminale IH-Situation, für die eine eindeutige Markierung u zu einem U-Graphen mit weniger als L Knoten führt, so stellt bei sonst gleichen Voraussetzungen wie in Satz 4.12 das erhaltene Programm P eine minimale Realisierung von Q dar.

Satz 4.13

Q = (b,M) sei PSP, G = IH-GRAPH(Q), G ≠ ε ;
IS = (A,R) sei aus G ableitbare terminale IH-Situation.
UM = (UG,u) sei eindeutiges Markierungspaar für A,

UM ≠ U-DUMMY, UL(A,u) ≤ L ; T-ALG sei T-Algorithmus.
Weiterhin gelte:
∀IS'. IS' = (A',R') ist aus G ableitbare terminale IH-Situation.
(∀u. u eindeutige Markierung für A => UL(A,u) ≥ L)

Behauptung: T-ALG(b,UG) ist minimal bzgl. Q .

Beweis:
Aus den Voraussetzungen folgt sofort, daß UL(A,u) = L .
Nach Satz 4.12 L-realisiert P = T-ALG(b,UG) Q . Es bleibt somit zu zeigen:

$\forall P' \varepsilon PROGRAMS.\ P'$ realisiert $Q \Rightarrow L_b(P) \leq L_b(P')$

Widerspruchsannahme:

$\exists P' \varepsilon PROGRAMS.\ P'$ realisiert $Q \wedge L_b(P) > L_b(P')$

Sei (V',E',a',c',u') = R-GRAPH(P',Q) [vgl. Def. 3.10],
A' = (V',E',a',c'), R' = (Ø,Ø,Ø,Ø,Ø).
Nach Satz 3.9 gilt:

(i) (A',R') ist aus G ableitbare terminale IH-Situation.
(ii) u' ist eindeutige Markierung für A'.
(iii) UL(A',u') = L_b(P')
=> UL(A',u') < L_b(P) = UL(A,u) = L
im Widerspruch zur Voraussetzung.

Die in diesem Kapitel beschriebenen Strategien und Methoden werden im PSA PA benützt, wobei die angegebenen Sätze als Grundlage für den formalen Korrektheitsbeweis von PA dienen.

Kapitel 5 DER PROGRAMMSYNTHESEALGORITHMUS PA

In Abschnitt 5.1 wird PA mit Hilfe der in Kap. 3 und 4 erarbeiteten Darstellungsarten und Vorgehensweisen definiert. Nach einem formalen Korrektheitsbeweis für PA in 5.2 wird nachgewiesen, daß PA die von einem PSA geforderten Eigenschaften besitzt. Anhand eines Beispiels wird PA in 5.3 erläutert, und in 5.4 folgen Hinweise darauf, wie dieser Programmsynthesealgorithmus günstig implementiert werden kann.

5.1 Definition von PA

Zu Anfang von Kapitel 4 war bereits ein Überblick über den Algorithmus PA gegeben worden. Die bei der Ausführung von PA anfallenden Zwischenwerte werden in einer Datenstruktur ASTACK der Art Keller gespeichert. Bei Verschiebung der Front nach rechts wird der Keller um 1 Element erweitert (PUSH-Operation) und die Frontnummer FNR um 1 erhöht; bei Frontverschiebung nach links wird ein Kellerelement entfernt (POP-Operation) und FNR um 1 dekrementiert, während die Initialwerte für FNR und ASTACK 0 bzw. empty sind. Sowohl FNR = 0 als auch ASTACK = empty zeigen mithin an, daß es sich um die erste Front handelt.

Ein einzelnes Kellerelement selbst ist ein Quadrupel bestehend aus einer IH-Situation IS, zugehöriger LB-Situation, dem Hypothesenraum HR an der Front von IS und der aus HR zuletzt gewählten Hypothese H.

Der Sonderfall, daß es für ein realisierbares PSP Q nur eine einzige terminale IH-Situation gibt (das trifft bei $b\varepsilon\{s,a\}$ zu, bei $b\varepsilon\{ss,as\}$ i.a. nicht, ist aber auch dort möglich), wird in PA gesondert behandelt [vgl. Abb. 5.1.-b-], es entfällt nämlich das Aufstellen der Instruktionshypothesen. Gibt es mehr als eine terminale IH-Situation, so wird wie in Abschnitt 4.1 skizziert verfahren [vgl. Abb. 5.1-c-].

Definition 5.1

(1) ASTACK bezeichnet einen Keller, dessen Elemente 4-Tupel der Art (IS,LB,HR,H) sind, wobei IS eine IH-Situation, LB eine LB-Situation, HR ein Hypothesenraum und H eine Hypothese aus HR ist. Die auf ASTACK erklärten Operationen sind wie üblich definiert:

PUSH = λel. λst. (el,st)

POP = λst. st = (el,st') —> st'

TOP = λst. st = (el,st') —> el

Der leere Keller wird mit empty bezeichnet.

(2) INIT-VALUES(G) bezeichnet für einen IH-Graphen G das 5-Tupel (L,FNR,IS,LB,ASTACK) mit:

(i) FNR = O

(ii) IS = GSUC(IH-INIT(G),INSTRUCTIONS)

(iii) LB = LB-SIT(G,IS)

(iv) LB = (Lmin,Lbr,Lb,d,IM,l) => L = Lmin

(v) ASTACK = empty

(3) U-ALG ist ein U-Algorithmus.

(4) T-ALG ist ein T-Algorithmus.

(5) TERMINAL(G,IS) ist ein boolescher Ausdruck, für den gilt:

TERMINAL(G,IS) = true <=> IS ist aus G ableitbare terminale IH-Situation

(6) GE(L,LB) ist ein boolescher Ausdruck, für den gilt:

GE(L,LB) = true <=> (LB≠ε) ∧ (LB=(Lmin,Lbr,Lb,d,IM,l) => L ≥ Lmin)

(7) PA ist der in Abb. 5.1 a-c dargestellte Programmsynthesealgorithmus.

Eine wichtige Anmerkung ist noch zu machen. PA ist unabhängig von dem einer Beispielsfolgenmenge zugrunde liegenden Berechnungsmodell und dessen Instruktions- und Speicherstruktur konzipiert. Demzufolge sind auch die Mengen STORES, ASSIGNMENTS und TESTS Eingabeparameter für PA, da sie bei der Berechnung des IH-Graphen G des eingegebenen PSP benötigt werden [vgl. Kap. 3], der Einfachheit halber soll aber an dieser Stelle hierauf nicht weiter eingegangen werden; liegt nämlich G vor, so sind in G alle für die weitere Programmsynthese notwendigen Informationen vollständig enthalten.

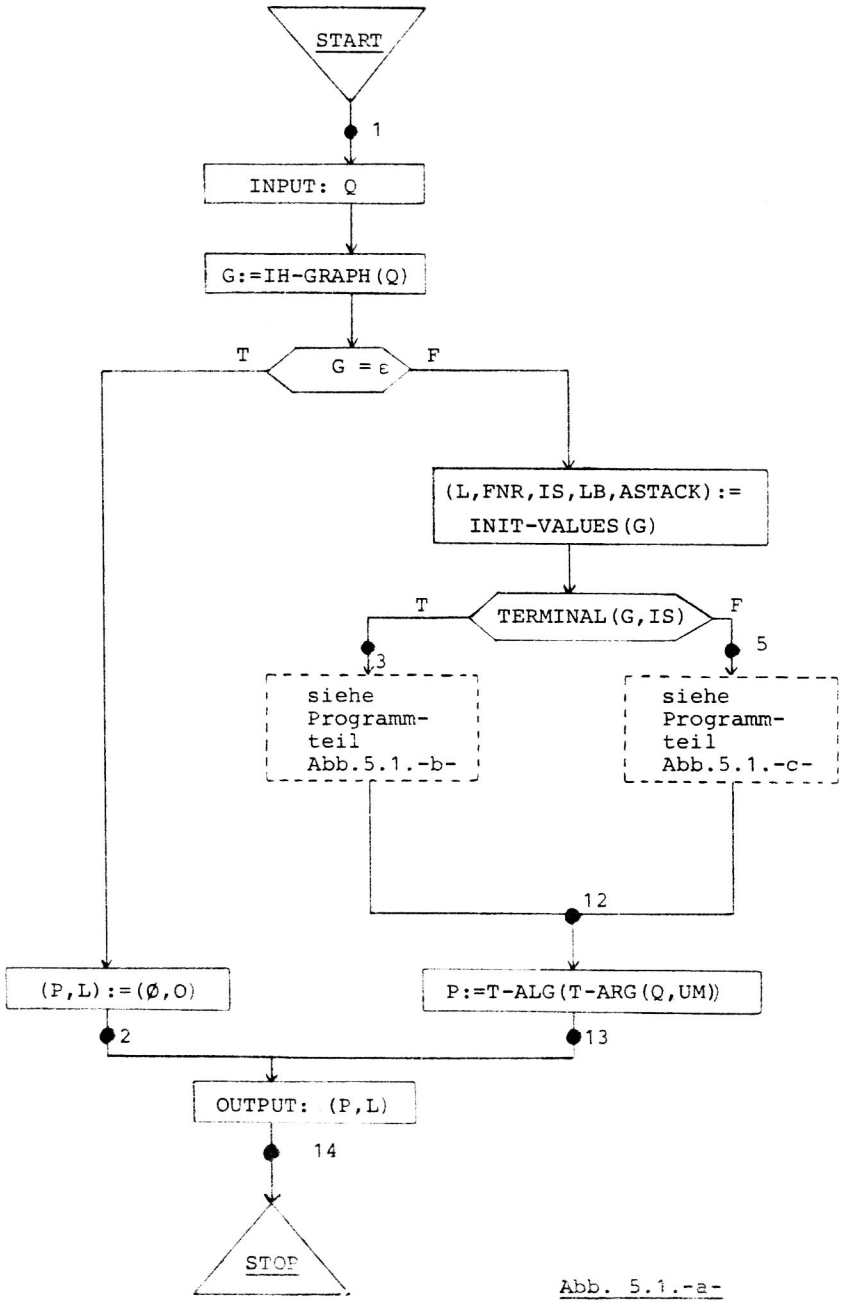


Abb. 5.1.-a-

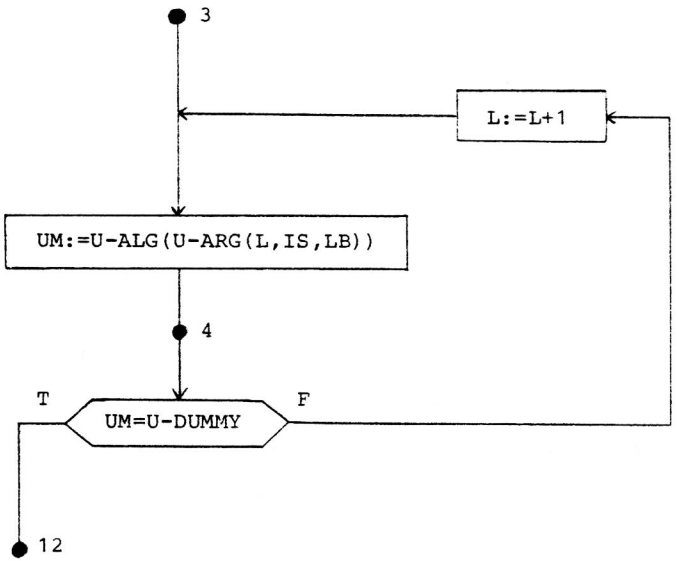


Abb. 5.1-b-

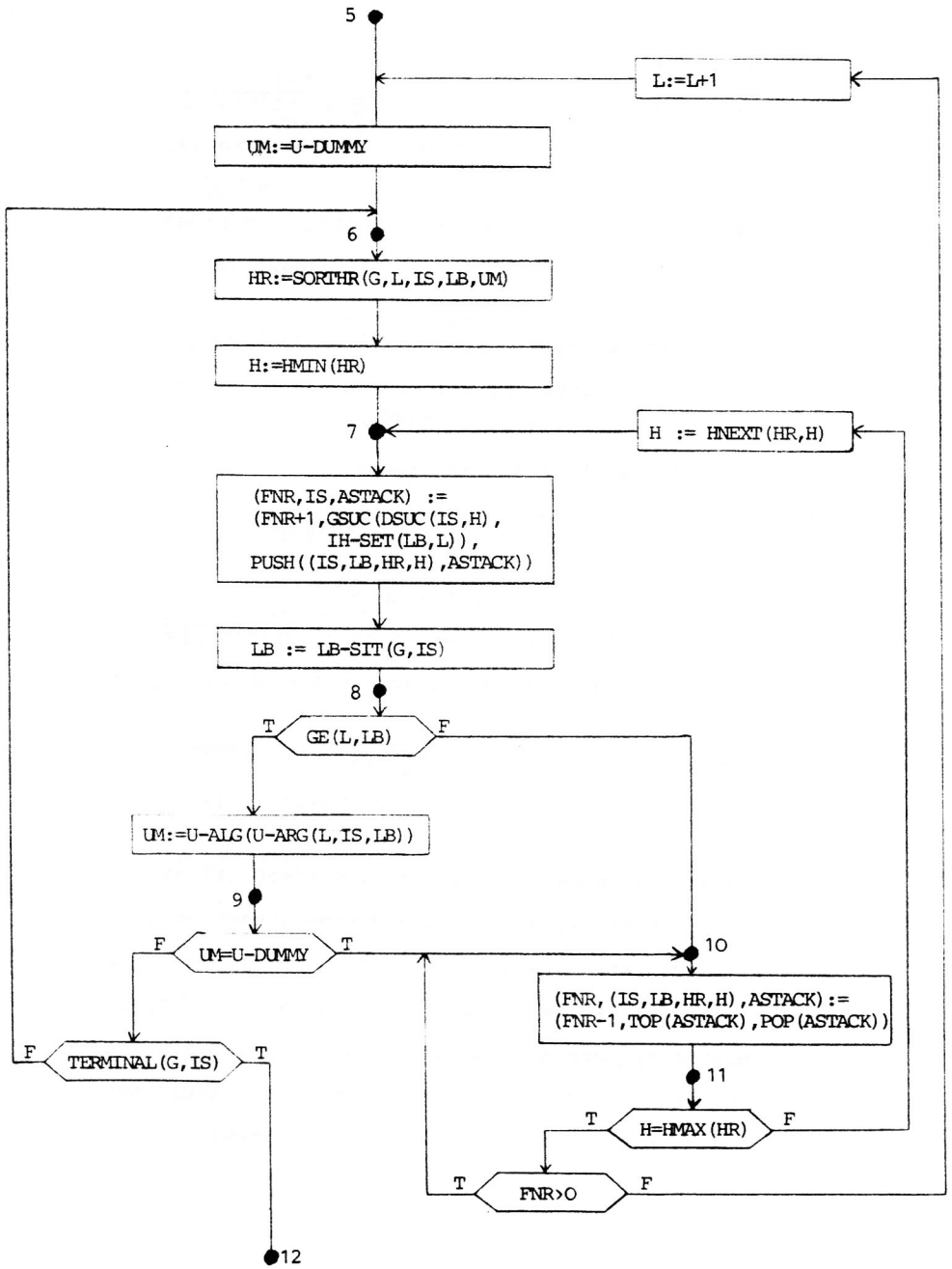


Abb. 5.1 -c-

5.2 Nachweis der Eigenschaften von PA

Ist Q ein endliches PSP, so liefert PA(Q) ein Q realisierendes minimales Programm P, wenn Q überhaupt realisierbar ist, sonst liefert PA(Q) die leere Menge. Diese Aussage wird in Satz 5.1 bewiesen.

Satz 5.1

Die Prädikate IN und OUT seien definiert durch:

$$IN(Q) = \underline{true} \quad <=> \quad Q \in PSP^{fin}_{I,S}$$

$$OUT(Q,P,L) = \underline{true} <=> [P = \emptyset \Rightarrow L=0 \wedge \neg(Q \text{ ist realisierbar})]$$
$$[P \neq \emptyset \Rightarrow P \text{ L-realisiert } Q$$
$$\wedge P \text{ ist minimal bzgl. } Q]$$

Behauptung: PA ist total korrekt bzgl. IN und OUT.

Beweis:

Der Beweis gliedert sich in Teil I (partielle Korrektheit) und Teil II (Terminierung).

Teil I: partielle Korrektheit

Die partielle Korrektheit von PA bzgl. IN und OUT wird mit Hilfe der Methode der induktiven Zusicherungen (IAM - inductive-assertions method [Manna 1974]) bewiesen. Die in Abb. 5.1 a-c eingezeichneten Schnittpunkte 1 - 14 schneiden die Kanten von PA so, daß es keinen zyklischen Teilpfad in PA gibt, der nicht mindestens einen Schnittpunkt hat;

darüberhinaus gibt es einen Schnittpunkt unmittelbar nach dem START- und einen unmittelbar vor dem STOP-Knoten. Bevor die den Schnittpunkten zugeordneten Zusicherungen ASS1,...,ASS14 angegeben werden, werden zur Vereinfachung zunächst einige Prädikate definiert, aus denen die ASSi dann aufgebaut werden können. Die 'Bedeutung' dieser Prädikate ist jeweils in informeller Weise mit angegeben.

1. "G ist IH-Graph von Q"

   IHG(Q,G) $\equiv Q \in PSP^{fin}_{I,S} \wedge G = \text{IH-GRAPH}(Q) \wedge G \neq \varepsilon$

2. "L ist untere Grenze für eindeutige Markierung von IS"

   LBIS(G,L,IS) $\equiv$ IS ist aus G ableitbare IH-Situation
   $\wedge$ ($\forall$ IS'. IS'=(A',R') ist term. Nachfolger von IS.
   ($\forall$ u. u ist eindeutige Markierung für A'
   $\Rightarrow$ UL(A',u) $\geq$ L))

3. "Ist L untere Grenze für IS, dann auch für IS' "

   LBIMPL(G,L,IS,IS') $\equiv$ LBIS(G,L,IS) $\Rightarrow$ LBIS(G,L,IS')

4. "L ist untere Grenze für G"

   LBG(G,L) $\equiv$ LBIS(G,L,IH-INIT(G))

5. "Es existiert ein terminaler Nachfolger von IS"

   ETERM(G,IS) $\equiv \exists$ IS'. (IS' ist aus G ableitbare terminale IH-Situation) $\wedge$ (IS' ist Nachfolger von IS)

6. "LB ist LB-Situation von IS"

   LBS(G,IS,LB) $\equiv$ IS ist aus G ableitbare IH-Situation
   $\wedge$ LB = LB-SIT(G,IS)

7. "u ist eindeutige Markierung für A und respektiert d"

   ULAD(A,u,d) $\equiv$ u ist eindeutige Markierung für A
   $\wedge$ [A = (V,E,a,c) $\Rightarrow$
   $\forall K \in V(\forall K' \in d(K).\ u(K) \neq u(K'))$]

8. "HR ist vollständiger Hypothesenraum für IS"

   CHR(G,L,IS,HR,LB,UM) $\equiv$ HR=SORTHR(G,L,IS,LB,UM)
   $\wedge$ [$\forall H \in FIH_G(IS)$.
   $H \notin HR \Rightarrow$ LBIS(G,L+1,DSUC(IS,H))]

9. "Hypothesen kleiner als H im L-Suchraum nicht mehr relevant"

   NRH(G,L,IS,HR,H) $\equiv \forall H' \in HR.\ H' <_{HR} H \Rightarrow$
   LBIS(G,L+1,DSUC(IS,H'))

10. "UM ist eindeutiges Markierungspaar"

    UMP(IS,UM) $\equiv$ IS = (A,R) $\Rightarrow$ UM ist eindeutiges Markierungspaar für A

11. "Stackinvariante"

    SIV(G,L,FNR,ASTACK) $\equiv$
    [$\forall i \in \{0,\ldots,FNR-1\}$. TOP(POP^i(ASTACK))=(IS,LB,HR,H)
    $\Rightarrow$ LBIS(G,L,IS)
    $\wedge$ LBS(G,IS,LB)
    $\wedge \neg$ TERMINAL(G,IS)

$\wedge$ ETERM(G,IS)

$\wedge$ NRH(G,L,IS,HR,H)

$\wedge$ CHR(G,L,IS,HR)]

$\wedge$ [$\forall i \in \{1,\ldots,FNR-1\}$.

TOP(POP^{i-1}(ASTACK)) = (IS,LB,HR,H)

$\wedge$ TOP(POP^{i}(ASTACK)) = (IS',LB',HR',H')

=> LBIMPL(G,L+1,IS,DSUC(IS',H'))]

$\wedge$ [FNR > 0 =>

(TOP(POP^{FNR-1}(ASTACK)) = (IS,LB,HR,H)

=> LBIMPL(G,L+1,IS,IH-INIT(G)))]

$\wedge$ [POP^{FNR}(ASTACK) = empty]

12. "Stackinvariante bzgl. aktuellem IS"

SIVIS(G,L,FNR,IS,ASTACK) $\equiv$

[FNR=0 => LBIMPL(G,L+1,IS,IH-INIT(G))]

$\wedge$ [FNR>0 => (TOP(ASTACK) = (IS',LB',HR',H')

=> LBIMPL(G,L+1,IS,DSUC(IS',H')))]

13. "Nachbedingung für U-Algorithmus"

UPOSTC(L,IS,LB,UM) $\equiv$

IS=(A,R) $\wedge$ LB = (Lmin,Lbr,Lb,d,IM,l) $\wedge$ UM=(UG,u) =>

[UG=$\emptyset$ => ($\forall$u. ULAD(A,u,d)=> $\sum_{I \in IM}$ max{Lb(I),UL(A,u,I)}>L-Lbr)]

$\wedge$ [UG$\neq\emptyset$ => ULAD(A,u,d) $\wedge$ $\sum_{I \in IM}$ max{Lb(I),UL(A,u,I)} $\leq$ L-Lbr]

14. "Invariante für Programmteil Abb. 5.1 -b-"

INV1(Q,G,L,IS,LB) $\equiv$ IHG(Q,G)

$\wedge$ LBG(G,L)

$\wedge$ LBIS(G,L,IS)

$\wedge$ LBS(G,IS,LB)

$\wedge$ GE(L,LB)

$\wedge$ TERMINAL(G,IS)

15. "Invariante für Programmteil Abb. 5.1 -c-"

INV2(Q,G,L,FNR,IS,LB,ASTACK) $\equiv$

IHG(Q,G)

$\wedge$ LBG(G,L)

$\wedge$ LBIS(G,L,IS)

$\wedge$ LBS(G,IS,LB)

$\wedge$ SIV(G,L,FNR,ASTACK)

$\wedge$ SIVIS(G,L,FNR,IS,ASTACK)

$\wedge$ FNR $\geq$ 0

Mit den so definierten Prädikaten werden die benötigten induktiven Zusicherungen ASSi i.f. angegeben, wobei zur Vereinfachung die Argumente auf der linken Seite nicht explizit mit aufgeführt sind.

ASS1 := IN(Q)

ASS2 := (P,L) = (Ø,O) ∧ ¬(Q ist realisierbar)

ASS3 := INV1(Q,G,L,IS,LB)

ASS4 := INV1(Q,G,L,IS,LB) ∧ UMP(IS,UM) ∧ UPOSTC(L,IS,LB,UM)

ASS5 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ ¬TERMINAL(G,IS) ∧ ETERM(G,IS) ∧ GE(L,LB)

ASS6 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ ¬TERMINAL(G,IS) ∧ ETERM(G,IS) ∧ GE(L,LB)
∧ UMP(IS,UM)

ASS7 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ ¬TERMINAL(G,IS) ∧ ETERM(G,IS)
∧ CHR(G,L,IS,HR,LB,UM)
∧ NRH(G,L,IS,HR,H)

ASS8 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ (FNR > O) ∧ (ASTACK ≠ empty)

ASS9 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ (FNR > O) ∧ (ASTACK ≠ empty)
∧ ETERM(G,IS) ∧ GE(L,LB)
∧ UMP(IS,UM) ∧ UPOSTC(L,IS,LB,UM)

ASS1O:= INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ (FNR > O) ∧ (ASTACK ≠ empty)
∧ LBIS(G,L+1,IS)

ASS11 := INV2(Q,G,L,FNR,IS,LB,ASTACK)
∧ ¬TERMINAL(G,IS) ∧ ETERM(G,IS)
∧ CHR(G,L,IS,HR,LB,UM)
∧ NRH(G,L,IS,HR,H)
∧ LBIS(G,L+1,DSUC(IS,H))

ASS12 := IHG(Q,G) LBG(G,L) TERMINAL(G,IS)
∧ UMP(IS,UM) ∧ (UM ≠ U-DUMMY)
∧ (UM = (UG,u) ∧ IS = (A,R) => UL(A,u) ≤ L)

ASS13 := (P ≠ Ø) ∧ (P L-realisiert Q) ∧ (P ist minimal bzgl.Q)

ASS14 := OUT(Q,P,L)

Als letzter Schritt für den Nachweis der partiellen Korrektheit müssen die Verifikationsbedingungen gezeigt werden. Es gibt in PA 21 Pfade zwischen 2 Schnittpunkten ohne einen dazwischenliegenden Schnittpunkt. Wird ein Pfad von Schnittpunkt i nach Schnittpunkt j mit (i,j) bezeichnet, so gilt:

1. Für die Pfade (2,14),(4,12),(5,6),(9,6),(9,12),(1O,11) und (13,14) ist der Nachweis der Verifikationsbedingungen trivial bzw. folgt aus der Anwendung des Zuweisungsaxioms.

2. Für die Pfade (3,4),(4,4) und (8,9) genügt ebenfalls das Zuweisungsaxiom bei Beachtung der Definitionen 4.6 und 5.1 .

3. Für die übrigen Pfade gibt die folgende Tabelle die jeweiligen Sätze an, aus denen die entsprechenden Verifikationsbedingungen folgen:

Pfad	Verifikationsbedingung folgt aus:
(1, 2)	Satz 3.5 und Def. 3.8
(1, 3)	Sätze 3.7 (6) und 4.6
(1, 5)	Sätze 3.7 (6) und 4.6
(6, 7)	Satz 4.10
(7, 8)	Satz 4.9
(8,10)	Satz 4.6
(9,10)	Satz 4.7
(11, 6)	Satz 4.10
(11, 7)	Satz 4.10
(11,10)	Satz 4.10
(12,13)	Sätze 4.12 und 4.13

Teil II: Terminierung

Mit Hilfe der Methode der wohlbegründeten Mengen (well-founded-sets method [Manna 1974]) wird gezeigt, daß PA über dem Eingabeprädikat IN terminiert. Hierzu sind zunächst einige Vorüberlegungen notwendig.

Sei IN(Q) = true, dim(Q) = $(m,(n_1,\ldots,n_m))$ und $n = \max\{n_1,\ldots,n_m\}$. N bezeichne die Menge der nicht-negativen ganzen Zahlen. Die in diesem Beweis benützte wohlbegründete Menge W ist die Menge aller (n+1)-Tupel über N mit der üblichen lexikographischen Ordnung: $W = (N^{n+1}, <_{n+1})$.

(Bemerkung: W kann also für unterschiedliche Eingaben Q u.U. verschiedene Dimensionen annehmen.)

Die für den Nachweis der Terminierung notwendigen Parameter sind i.w. der Suchraumindex L, die Anzahl der Fronten FNR und die aktuelle Hypothese H. Dabei wird ausgenützt, daß es an einer Front nur endlich viele verschiedene Hypothesen gibt, daß die Anzahl der Fronten und die Größe des Suchraumindex beschränkt sind und daß eine Konstellation mit gleichem Index L, gleicher Frontnummer FNR und gleicher Hypothese H höchstens ein einziges Mal bei Ausführung von PA(Q) auftreten kann.

Sei G = IH-GRAPH(Q), G = (b,V,E,f,g,t), $G \neq \varepsilon$.

$$ubV(G) := |V|$$

$$ubH(G) := 1 + \prod_{K \in V} |f(K)|$$

Damit gilt:

1. ubV(G) ist eine triviale obere Grenze für den Suchraumindex L: für ein minimales Programm P, das Q realisiert, gilt sicherlich: $L_b(P) < ubV(G)$.

2. ubH(G) ist eine triviale obere Grenze für die Mächtigkeit der während der Ausführung von PA(Q) erzeugten Hypothesenräume HR (vgl. Def. 4.5). Für alle diese HR gilt: $|\{H|H \in HR\}| < ubH(G)$.

3. n ist eine obere Grenze für die Werte, die FNR während der Ausführung von PA(Q) annehmen kann; für alle diese FNR gilt: FNR < n .
   Insbesondere gilt an den Schnittpunkten 7 und 11 immer: FNR < n - 1 .

Außerdem werden noch einige weitere Hilfsdefinitionen benötigt. Für H∈HR gebe F(HR,H) an, an wievielter Stelle H in dem durch $<_{HR}$ geordneten Hypothesenraum HR steht, wobei HMAX(HR) die 1. Stelle zugeordnet wird; für j∈{1,...,n} gebe F_j(G,FNR,HR,H,ASTACK) diese Information für die j-te Front der i.w. durch ASTACK repräsentierten Abarbeitungssituation an; d.h.:

$$F(HR,H) = i+1 \iff HR = \{H_1,\ldots,H_r\} \wedge H_1 <_{HR} \cdots <_{HR} H_r \wedge H = H_{r-i}$$

Für j∈{1,...,n} sei:

$$F_j(G,FNR,HR,H,ASTACK) = \begin{cases} F(HR',H') \iff j \leq FNR \wedge TOP(POP^{FNR-j}(ASTACK)) \\ \qquad\qquad = (IS',LB',HR',H') \\ F(HR,H) \iff j = FNR+1 \\ ubH(G) \iff j > FNR+1 \end{cases}$$

$$F'_j(G,FNR,HR,H,ASTACK) = \begin{cases} F_j(G,FNR,HR,H,ASTACK) \iff j \leq FNR+1 \\ 0 \iff j > FNR+1 \end{cases}$$

Nach diesen Vorüberlegungen können die vier Teilschritte der Methode der wohlbegründeten Mengen durchgeführt werden.

Schritt 1: "Schneide alle Schleifen"

Die Schnittpunkte 4,7 und 11 aus Abb. 5.1 werden benutzt. Jeder zyklische Teilpfad von PA geht durch mindestens einen dieser Schnittpunkte.

Schritt 2: "Ordne den Schnittpunkten 'good assertions' zu"

Die gesuchten Zusicherungen werden mit GA4, GA7 und GA11 bezeichnet und sind definiert durch:

GA4 := ASS4 ∧ (L < ubV(G))

GA7 := ASS7 ∧ (L < ubV(G)) ∧ (FNR < n-1) ∧ (F(HR,H) < ubH(G))
∧ (∀ i∈{1,...,n}. F_i(G,FNR,HR,H,ASTACK) ≥ 0)

GA11:= ASS11 ∧ (L < ubV(G)) ∧ (FNR < n-1) ∧ (F(HR,H) < ubH(G))
∧ (∀ i∈{1,...,n}. F'_i(G,FNR,HR,H,ASTACK) ≥ 0)

Daß diese GAi die Bedingungen einer "good inductive assertion" erfüllen, wird leicht durch die in Teil I beschriebene Vorgehensweise gezeigt.

Schritt 3: "Ordne den Schnittpunkten 'good functions' zu"

Die Funktionen werden mit GF4, GF7 und GF11 bezeichnet; bei ihrer Definition werden wie bei den Definitionen der ASSi und der GAi die Argumente nicht explizit mit angegeben; sie ergeben sich implizit - wie oben auch - durch die auf der rechten Seite vorkommenden Variablennamen.

GF4 := (ubV(G)-L,0,0,...,0)

GF7 := (ubV(G)-L,F_1(G,FNR,HR,H,ASTACK),...
...,F_n(G,FNR,HR,H,ASTACK))

GF11 := (ubV(G)-L,F'_1(G,FNR,HR,H,ASTACK),...

...,F'_n(G,FNR,HR,H,ASTACK))

Aus den zusätzlich zu ASSi in GAi hinzugenommenen Zusicherungen folgt direkt, daß diese GFi die Bedingungen einer "good partial function" erfüllen und in die durch $W = (N^{n+1}, <_{n+1})$ gegebene wohlbegründete Menge abbilden.

Schritt 4: "Zeige, daß die Terminierungsbedingungen erfüllt sind"

Es gibt 7 verschiedene Pfade, die für Schritt 4 relevant sind:

PF1 - von 4 nach 4

PF2 - von 7 über 8 und 10 nach 11

PF3 - von 7 über 8,9 und 10 nach 11

PF4 - von 7 über 8,9 und 6 nach 7

PF5 - von 11 über 10 nach 11

PF6 - von 11 nach 7

PF7 - von 11 über 6 nach 7

Zur Vereinfachung sei folgendes vereinbart:
Betrachtet man den Pfad PF1 und geht dieser Pfad vom Schnittpunkt i zum Schnittpunkt j, so bezeichne (G',L',FNR',HR',H',ASTACK') die sich aus (G,L,FNR,HR,H,ASTACK) und dem Pfadeffekt von PF1 ergebenden Werte; darüber hinaus sei:

GFi(G,L,FNR,HR,H,ASTACK) = $(Y,X_1,...,X_n) = w$

GFj(G',L',FNR',HR',H',ASTACK') = $(Y',X'_1,...,X'_n) = w'$

Mit diesen Bezeichnungen ist also die unter GAi und der Pfadbedingung von PF1 zu zeigende Terminierungsbedingung:

$$w >_{n+1} w'$$

zu PF1: Es gilt: G' = G

L' = L + 1

=> w = (ubV(G)-L,0,...,0)

$>_{n+1}$(ubV(G)-(L+1),0,...,0) = w'

zu PF2 und PF3:

(G',L',FNR',HR',H',ASTACK') = (G,L,FNR,HR,H,ASTACK)

=> Y = Y'

$\forall$ iε{1,...,FNR+1}. $X_i = X'_i$

Da wegen GA7 FNR + 2 $\leq$ n :

X_{FNR+2} = ubH(G) > 0 = X'_{FNR+2}

=> w $>_{n+1}$ w'

zu PF4: G' = G

L' = L

FNR' = FNR + 1

ASTACK' = PUSH((IS,LB,HR,H),ASTACK)

=> Y = Y'

$\forall$ iε{1,...,FNR+1}. $X_i = X'_i$

X_{FNR+2} = ubH(G)

> F(HR',H') [wegen GA7]

= $X'_{FNR'+1}$

= X'_{FNR+2}

=> $w >_{n+1} w'$

zu PF5:

G' = G

L' = L

FNR' = FNR - 1

ASTACK' = POP(ASTACK)

(IS',LB',HR',H') = TOP(ASTACK)

=> Y = Y'

$\forall$ iε{1,...,FNR}. $X_i = X'_i$

X_{FNR+1} = F(HR,H)

= F(HR,HMAX(HR)) [nach Pfadbedingung]

= 1 [Def. von F]

> 0

= $F'_{FNR'+2}$(G',L',FNR',HR',H',ASTACK')

= $X'_{FNR'+2}$

= X'_{FNR+1}

=> $w >_{n+1} w'$

zu PF6:

G' = G

L' = L

FNR' = FNR

HR' = HR

H' = HNEXT(HR,H)

=> Y = Y'

$\forall$ iε{1,...,FNR}. $X_i = X'_i$

X_{FNR+1} = F(HR,H)

> F(HR,H) + 1

= F(HR,HNEXT(HR,H))

= F(HR',H') = X'_{FNR+1}

=> $w >_{n+1} w'$

zu PF7:

G' = G

L' = L + 1

=> Y = ubV(G) - L

> ubV(G) - (L+1)

= ubV(G') - L'

= Y'

=> $w >_{n+1} w'$

Durch den Nachweis aller Terminierungsbedingungen ist hiermit der Beweis von Teil II und damit auch der des Satzes 5.1 erfolgreich abgeschlossen.

In den Aussagen des folgenden Satzes werden die Eigenschaften von PA zusammengefaßt, die nach den in Abschnitt 2.4 aufgestellten Zielsetzungen ein PSA erfüllen sollte.

Satz 5.2

PA erzeugt aus $PSP^{fin}_{I,S}$

(1) fehlerfreie

(2) minimale

Beispielsrealisierungen und ist bzgl. PROGRAMS

(3) korrekt

(4) vollständig

(5) lösungsminimal.

Beweis:

(1) und (2) wurden in Satz 5.1 gezeigt, (3) und (5) sind leichte Folgerungen daraus; zu (4) folgende Überlegungen. P sei ein Programm, $C_1, C_2, C_3, \ldots$ eine Aufzählung der terminierenden vollständigen Berechnungsfolgen von P, $b \in \{\underline{s}, \underline{a}, \underline{ss}, \underline{as}\}$. Zu zeigen ist, daß ein endliches Präfix von $C_1, C_2, \ldots$ für PA genügt, um ein Programm P' zu erzeugen, das b-getreue Extension von P ist, also jede terminierende Berechnungsfolge von P auch terminierende Berechnungsfolge von P' ist.

Betrachtet man für $i \in \mathbb{N}$ die PSP $Q_i = GPSP(b, \{C_1, \ldots, C_i\})$ (vgl. Def. 2.18), die IH-Graphen $G_i = IH\text{-}GRAPH(Q_i)$ und die Programme $P_i = PA(Q_i)$, so kann man die Folge $P_1, P_2, P_3, \ldots$

als eine immer bessere Annäherung an das Programm P (bzw. P') ansehen. Ab einem bestimmten Index r unterscheiden sich die Programme $P_{r'}$ $(r' \geq r)$ nicht mehr von P_r, das somit identisch ist mit dem Zielprogramm P'.

Da die auf STORES definierte Äquivalenzrelation R

$$(s,s') \in R \iff \bar{\gamma}(s, \underline{true}) = \bar{\gamma}(s', \underline{true})$$

eine endliche Zerlegung von STORES induziert, ist die bei der Bestimmung der Stufengraphdarstellung der Instruktionenmatrix der Q_i (vgl. Def. 3.5) erzeugte Anzahl der Knoten einer jeden Stufe beschränkt. Bezeichnet für $i \geq 1$, $j \geq 0$ X_i^j die Menge der Knoten der Stufe j im IH-Graphen G_i:

$$X_i^j = \{K \mid K \text{ ist Knoten von } G_i \wedge LEVEL(K) = j\}$$

so gilt also:

$$\forall j \in \mathbb{N}.\ \exists Y_j \in \mathbb{N}.\ \forall i \in \mathbb{N}.\ |X_i^j| \leq Y_j$$

Zwei Fälle sind zu unterscheiden:

1. Gibt es eine obere Schranke n für die Länge der Berechnungsfolgen von P, so werden ab einem r die Indizes aller weiteren Berechnungsfolgen redundant sein (im Sinne von Def. 3.8) und in den G_i nicht mehr auftreten. Daher gilt:

   $$\exists r \in \mathbb{N}.\ \forall i \geq r.\ \forall j \in \{0, \ldots, n\}.\ X_i^j = X_r^j$$

   Ab einem r' unterscheiden sich die Knoten der G_i $(i \geq r')$ auch nicht mehr in den Markierungen oder in den Übergängen zwischen ihnen, und da es keine Knoten K mit LEVEL(K) > n

geben kann, folgt:

$$\exists\, r' \in \mathbb{N}\,.\ \forall\, i \geq r'.\ \ G_i = G_{r'}$$

und damit auch:

$$\forall\, i \geq r'.\ PA(Q_i) = PA(Q_{r'})$$

Dieses wiederum impliziert, daß $PA(Q_{r'})$ alle Berechnungsfolgen von P ausführen kann.

2. Gibt es keine obere Schranke für die Länge der Berechnungsfolgen von P, so führt eine analoge Überlegung zum Ziel. Dann gibt es nämlich für jedes $n \in \mathbb{N}$ ein $r_n \in \mathbb{N}$, sodaß alle G_i $(i \geq r_n)$ sich in den ersten n Stufen lediglich durch die Größe der Indexmengen der einzelnen Knoten von G_{r_n} unterscheiden, ansonsten aber identisch sind mit G_{r_n}, was Anzahl und Markierung der Knoten bis zur Stufe n und Kanten sowie Kantenmarkierungen zwischen diesen angeht. Wählt man nun n genügend groß, so führt die Bearbeitung des Teilgraphen von G_r [sei $r := r_n$], der alle Knoten bis zu Stufe n umfaßt, durch den PSA PA zu einem IH-Anfangsgraphen A_r und einer LABEL-Hypothese u_r, die für jedes $i \geq r$ bei der weiteren Bearbeitung von G_i durch PA zu einer aus G_i ableitbaren terminalen IH-Situation (A_i, R_i) und einer LABEL-Hypothese u_i erweitert werden, ohne daß dadurch neue Programmknoten oder neue Übergänge erzeugt werden, d.h.

$$\forall\, i \geq r.\ \text{U-GRAPH}(A_i, u_i) = \text{U-GRAPH}(A_r, u_r).$$

Daraus folgt aber

$$\forall\, i \geq r.\ \ PA(Q_i) = PA(Q_r)$$

analog zu Fall 1.

Es muß allerdings noch eine triviale Forderung an den benutzten U-Algorithmus U-ALG gestellt werden, und zwar die, daß die Markierung von Programmknoten eines Programms P in irgendeiner From standardisiert ist (z.B. derart, daß bei l I-Knoten in P nur die LABEL 1,...,l für diese I-Knoten verwandt werden [vgl. Ausführungen zu Def. 4.6]) und die Wahl der LABEL für einen IH-Anfangsgraphen gemäß dieser Standardisierung erfolgt.

Aussage (4) folgt aus der Formalisierung der angegebenen Überlegungen.

5.3 Beispiel einer PA-Berechnung

Ein Beispiel soll die gesamte Vorgehensweise von PA im Zusammenhang verdeutlichen. Soweit möglich, wird dabei auf die bereits in den Beispielen der vorigen Kapitel angegebenen Werte zurückgegriffen werden. Die auftretenden Graphen werden wie üblich skizziert. Als Eingabe für PA soll hier das PSP $Q_3 = (\underline{ss},\{M_1,M_2,M_3,M_4\})$ aus Beispiel 2.3-3 benutzt werden.

1. INPUT: Q := Q_3

2. G := IH-GRAPH(Q)
   Die Berechnung von G ist bereits in Kapitel 3 ausführlich beschrieben worden. G selbst ist in Beispiel 3.7-2 angegeben.

3. Da G $\neq \varepsilon$ ist, ist Q realisierbar.

4. (L,FNR,IS,LB,ASTACK) := INIT-VALUES(G)
   4.1 IH-INIT(G) ist die IH-Situation IS' = (A',R'), bei der A' nur den Startknoten und R' alle restlichen Knoten von G enthält. Der größte erzwungene Nachfolger IS = (A,R) von IS' gemäß INSTRUCTIONS enthält im Anfangsgraphen A außerdem noch den Knoten [1,{1,2,3}], alle übrigen sind in R.
   IS = (A,R) ist identisch mit IS_2 aus Beispiel 3.10-2.

   4.2 LB := LB-SIT(G,IS)
   Diese LB-Situation ist in Abschnitt 4.2.1 berechnet worden und ist in Beispiel 4.4 wiedergegeben. Danach ist:
   LB = (Lmin,Lbr,Lb,d,IM,l)
   = (9 , 1 ,Lb,d,IM,6) mit:

I	S1(2)	CL(1)	A(1,3)	A(1,4)	ZR(2)	sonst.I
Lb(I)	1	1	2	1	1	0

   d = λK. Ø
   IM = {S1(2),CL(1),A(1,3),A(1,4),ZR(2)}

   4.3 L := 9
   FNR := 0
   ASTACK := <u>empty</u>

5. TERMINAL(G,IS) liefert <u>false</u>, da IS keine terminale IH-Situation ist; es wird mit dem in Abb. 5.1 -c- angegebenen Programmteil von PA fortgefahren.

6. UM := U-DUMMY
   = (Ø,λK.1)

7. HR := SORTHR(G,L,IS,LB,UM)
   Die zu IS gehörige Front F besteht nur aus dem Knoten K = [2,{1,2,3}]; F = (K). Da UG, wenn UM = (UG,u) ist, die leere Menge ist, ist die Vorschlagshypothese VH = (<u>dummy</u>). Die Markierung $f_R(K)$ der möglichen Instruktionshypothesen ist $f_R(K)$ = {CL(1),A(i,1),ZR(i)}.

Da IH-SET(LB,L) = IH-SET(LB,9) = INSTRUCTIONS,
gehören alle Elemente aus $f_R(K)$ zum Hypothesenraum HR.
Für $I \varepsilon \{CL(1), ZR(2)\}$ ist Lb(I) = 1, für die übrigen
$I \varepsilon f_R(K)$ ist Lb(I) = 0. In jedem Fall werden die Hypothesen CL(1),ZR(2) demnach vor allen anderen aufgestellt;
über die weitere Reihenfolge soll nach Abschnitt 4.4.3
eine lineare Ordnung auf INSTRUCTIONS entscheiden.
Da CL(1) zum Widerspruch führt, werde diese Hypothese
hier als erste gewählt, um die Reaktionsweise von PA
in einem solchen "Fehlerfall" zu verdeutlichen:

HR :=
⌈ CL(1) ⌉
| ZR(2) |
| A(1,1) |
| A(1,2) |
| A(1,3) |
| A(1,4) |
| A(1,5) |
| ZR(1) |
| ZR(3) |
| ZR(4) |
⌊ ZR(5) ⌋

8. H := HMIN(HR)
= (CL(1))

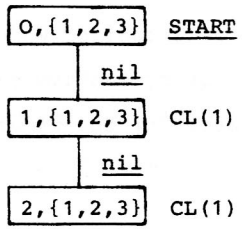
9. (FNR,IS,ASTACK) := (FNR+1,GSUC(DSUC(IS,H),IH-SET(LB,L)),
PUSH((IS,LB,HR,H),ASTACK))

9.1 FNR := 1

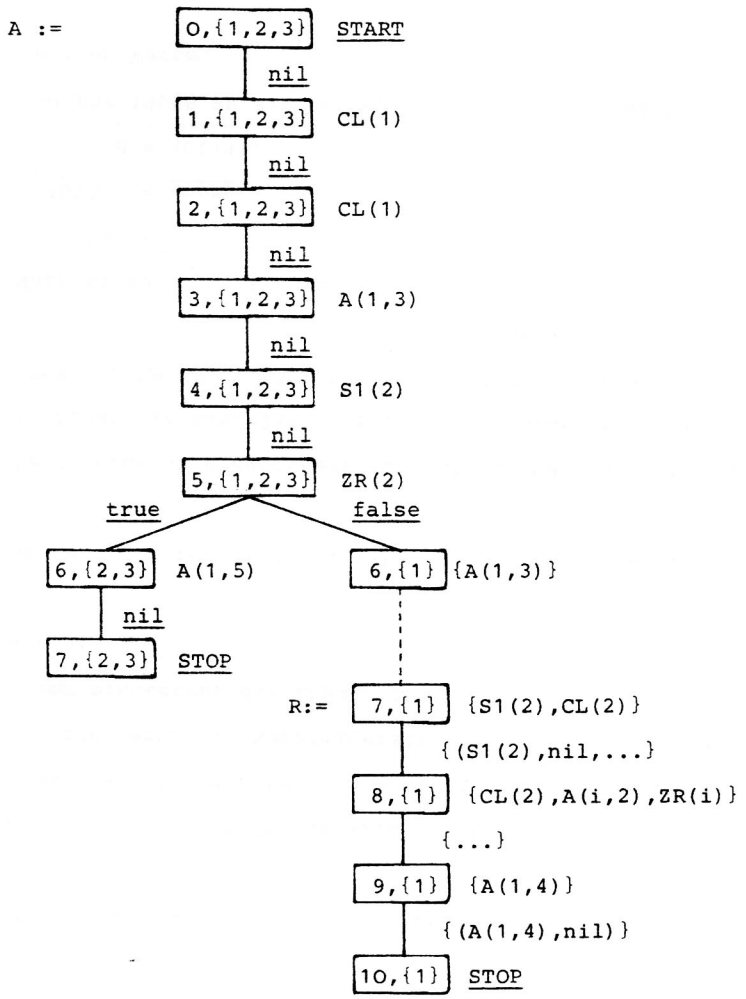
9.2 ASTACK := PUSH((IS,LB,HR,(CL(1))),<u>empty</u>)
mit den in Schritten 4.1, 4.2 und 7 angegebenen
Werten für IS,LB und HR.

9.3 IS := GSUC(DSUC(IS,H),IH-SET(LB,L))
IS' = DSUC(IS,H) = DSUC(IS,(CL(1)))
= (A',R') ist gegeben durch:

A' :=



R' enthält die von [3,{1,2,3}] in G erreichbaren
Knoten. Da IH-SET(LB,L) = INSTRUCTIONS, ist
IS = DSUC(IS',IH-SET(LB,9)) = (A,R) mit:



10. LB := LB-SIT(G,IS)

Zur Berechnung von LB wird zunächst T = IH-TREE(IS) bestimmt. T entsteht aus A und R und der üblichen Modifizierung der Markierungen von A [vgl. Def. 4.1]. Analog zu der auf Satz 4.2 zur Berechnung von LB-Situationen begründeten Vorgehensweise in Beispiel 4.5-2 erhält man:

K	$\{K' \mid K' \overset{+}{\neq} K\}$	d(K)	f(K)	$V_T(M_1)$	$V_T(M_2)$	$V_T(M_3)$	$V_T(M_4)$	$V_T(M_5)$
[1,{1,2,3}]	[2,{1,2,3}]	[2,{1,2,3}]	CL(1)	*				
[2,{1,2,3}]	[1,{1,2,3}]	[1,{1,2,3}]	CL(1)	*				
[3,{1,2,3}]	[6,{1}]	[6,{1}]	A(1,3)		*			
[4,{1,2,3}]	[7,{1}]	ø	S1(2)			*		
[5,{1,2,3}]	[8,{1}]	ø	ZR(2)			*		
[6,{2,3}]	ø	ø	A(1,5)				*	
[6,{1}]	[3,{1,2,3}]	[3,{1,2,3}]	A(1,3)		*			
[7,{1}]	[4,{1,2,3}], [8,{1}]	-	S1(2), CL(2)			*		
[8,{1}]	[5,{1,2,3}], [7,{1}]	-	CL(2), A(i,2), ZR(i)			*		
[9,{1}]	ø	-	A(1,4)					*

Hieraus lassen sich die Werte (Lmin,Lbr,Lb,d,IM,l) = LB ableiten, wie z.B.

I	S1(2)	CL(1)	A(1,3)	A(1,4)	A(1,5)	ZR(2)	sonst.I
Lb(I)	1	2	2	1	1	1	0

$$Lmin = LB_T(INSTRUCTIONS) = \sum_{i-1}^{5} LB_T(M_i) = 2+2+4+1+1 = 10$$

Bereits an dieser Stelle wird erkannt, daß die zuletzt gewählte Hypothese bei L = 9 nicht zum Erfolg führen kann, da jede terminale Nachfolgersituation von IS zu einem Programm mindestens der Länge Lmin = 10 führt [vgl. Satz 4.6] :

11. GE(LB,L) liefert wegen L = 9 < 10 = Lmin false .

12. Bevor eine neue Hypothese aufgestellt werden kann, müssen zunächst die aktuellen Werte wieder hergestellt werden:
(FNR,(IS,LB,HR,H),ASTACK) := (FNR-1,TOP(ASTACK),
POP(ASTACK))
Nach dieser Zuweisung gelten
FNR = 0
ASTACK = empty
H = (CL(1))
und für IS,LB,HR die in Schritten 4.1, 4.2 und 7. angegebenen Werte.

13. H = HMAX(HR) liefert false, da H = (CL(1)) ≠ (ZR(5)) = HMAX(HR).

14. H := HNEXT(HR,H)
= HNEXT(HR,(CL(1)))
= (ZR(2))

15. (FNR,IS,ASTACK) := (FNR+1,GSUC(DSUC(IS,H),IH-SET(LB,L)),
PUSH((IS,LB,HR,H),ASTACK))

15.1 FNR := 1

15.2 ASTACK := PUSH((IS,LB,HR,(ZR(2))),empty)
mit den in Schritten 4.1, 4.2 und 7. angegebenen Werten

15.3 IS := GSUC(DSUC(IS,(ZR(2))),INSTRUCTIONS)
IS ist identisch mit der in Beispiel 3.11-2 angegebenen IH-Situation IS_4.

16. LB := LB-SIT(G,IS)
Analog zu Schritt 10 erhält man folgende Werte:

K	{K' | K' $\ddagger^{+}$ K}	d(K)	f(K)	$V_T(M_1)$	$V_T(M_2)$	$V_T(M_3)$	$V_T(M_4)$	$V_T(M_5)$
[1,{1,2,3}]	∅	∅	CL(1)	*				
[2,{1,2,3}]	[5,{1,2,3}]	[5,{1,2,3}]	ZR(2)			*		
[3,{1,2,3}]	[6,{1}]	[6,{1}]	A(1,3)		*			
[4,{1,2,3}]	[7,{1}]	∅	S1(2)			*		
[5,{1,2,3}]	[2,{1,2,3}], [8,{1}]	[2,{1,2,3}]	ZR(2)			*		
[6,{2,3}]	∅	∅	A(1,5)				*	
[6,{1}]	[3,{1,2,3}]	[3,{1,2,3}]	A(1,3)		*			
[7,{1}]	[4,{1,2,3}], [8,{1}]	-	S1(2), CL(2)			*		
[8,{1}]	[5,{1,2,3}], [7,{1}]	-	CL(2), A(2,2), ZR(1)			*		
[9,{1}]	∅	-	A(1,4)					*

I	S1(2)	CL(1)	A(1,3)	A(1,4)	A(1,5)	ZR(2)	sonst.I
Lb(I)	1	1	2	1	1	2	0

$$Lmin = LB_T(INSTRUCTIONS) = \sum_{i=1}^{5} LB_T(M_i) = 1+2+4+1+1 = 9$$

$$IM = \{S1(2), CL(1), A(1,3), A(1,4), A(1,5), ZR(2)\}$$

$$Lbr = LB_T(INSTRUCTIONS, IM) = \sum_{i=1}^{5} LB_T(M_i, IM) = 0$$

$$l = \sum_{I \in IM} Lb(I) = 8$$

wobei:

LB = (Lmin,Lbr,Lb,d,IM,l)

17. GE(L,LB) liefert <u>true</u> wegen L = 9 ≥ 9 = Lmin.

18. UM := U-ALG(U-ARG(L,IS,LB))

    = U-ALG(N,A,Lb,d,IM,l)

    mit den in Beispiel 4.12-1 angegebenen Werten für N,...,l. Mit den dort berechneten Werten für UG und u gilt:

    UM = (UG,u)

19. UM = U-DUMMY liefert <u>false</u>, da UG ≠ ∅.

20. TERMINAL(G,IS) liefert ebenfalls <u>false</u>, da die Front von IS nicht leer ist.

21. HR := SORTHR(G,L,IS,LB,UM)

    Analog zu Schritt 10 erhält man Vorschlagshypothese VH = (<u>dummy</u>) und wegen Lb(S1(2)) = 1 > 0 = Lb(CL(2))

$$HR = \begin{bmatrix} S1(2) \\ CL(2) \end{bmatrix}$$

22. H := HMIN(HR)

    H = (S1(2)) wird als nächste Hypothese an der aktuellen Front ([7,{1}]) aufgestellt; diese Hypothese führt nicht zum Widerspruch. Im Gegensatz zu den in Schritt 16 angegebenen Werten gilt nun Lb(S1(2)) = 2, was wegen

$l = \sum_{I \in IM} Lb(I) = 9$ und $L = 9$ zu

IH-SET(LB,L) = IM = {S1(2),CL(1),A(1,3),A(1,4),A(1,5),ZR(2)}

führt, wobei LB die neue LB-Situation bezeichne.

Wegen $f_R([8,\{1\}]) \cap IM = \{ZR(2)\}$ kommt als einzige Hypothese für ([8,{1}]) nur (ZR(2)) in Betracht. So erhält man die terminale IH-Situation IS = IS_5, die in Beispiel 3.11-3 angegeben ist; die zugehörige LB-Situation LB wurde in Beispiel 4.7 bestimmt. UM=U-ALG(U-ARG(L,IS,LB)) wurde in Beispiel 4.12.-2 berechnet; mit diesem Wert (UG,u) für UM geht es bei Schnittpunkt 9 in Abb. 5.1 -c- wie folgt weiter:

23. UM = U-DUMMY liefert <u>false</u> wegen UG ≠ ∅.

24. TERMINAL(G,IS) liefert <u>true</u>, da IS terminale IH-Situation ist.

25. P := T-ALG(T-ARG(Q,UM))

= T-ALG(<u>ss</u>,UG)

Da für b = <u>ss</u> die Tripel von UG bereits Programmtransitionen sind, ist P gleich UG.

26. OUTPUT:(P,L)

P ist minimal bzgl. Q und es gilt $L_{\underline{ss}}(P) = 9 = L$.

P ist identisch mit dem in Beispiel 2.2 angegebenen Programm.

5.4 Hinweise für eine Implementierung von PA

Die Berechnung von IH-GRAPH(Q) eines PSP Q ist in Kapitel 3 der besseren Übersicht wegen in vier einzelne Teilschritte aufgegliedert worden, und zwar werden dort nacheinander die Standarddarstellung Q_{ST} = (b,S,MI,MT) von Q, die Stufengraphdarstellungen von MI und Q und schließlich G = IH-GRAPH(Q) bestimmt. Bei einer aktuellen Realisierung von PA könnten diese vier Teilschritte miteinander verknüpft werden. So läßt sich die nicht-reduzierte Stufengraphdarstellung der Instruktionsmatrix aus Q heraus berechnen, ohne daß MI zuvor explizit angegeben wird. Aus der reduzierten Stufengraphdarstellung von MI kann durch Einfügen der zusätzlichen Kantenmarkierungsfunktion und Löschen der redundanten Indizes der Indexmengen in den einzelnen Knoten G in einem Schritt bestimmt werden.

Für die Darstellung von G können die Standardimplementierungen für Bäume benutzt werden; für den Fall b = <u>ss</u> muß dabei noch berücksichtigt werden, daß ein Knoten mehr als eine Eingangskante besitzen kann.

Bei der Darstellung einer IH-Situation IS = (A,R) kann ausgenützt werden, daß die Knoten von A und R Teilmenge der Knoten von G sind und darüberhinaus die Markierungen ebenfalls schon in G enthalten sind. Daher kann IS bereits durch eine zusätzliche Knotenmarkierungsfunktion F_{IS} repräsentiert werden: F_{IS} gibt für einen Knoten K aus G an, ob er in IS

enthalten ist oder nicht (diese Angabe ist nur für b = ss relevant) und, wenn ja, die Markierung a(K) bzw. $f_R(K)$. Darauf aufbauend läßt sich eine Nachfolgesituation IS' = (A',R') ebenfalls einfach darstellen. Alle Knoten aus A haben in A' dieselbe Markierung, die Markierungen der Knoten aus R' sind in denen aus R enthalten.

Bei der Ausführung von PA sind zu jedem Zeitpunkt in ASTACK nur IH-Situationen enthalten, die jeweils in Nachfolgerrelation zueinander stehen. Mit den obigen Ausführungen bietet es sich an, alle diese IH-Situationen durch Verweise auf die Darstellung des IH-Graphen G zu repräsentieren. Satt der vollständigen IH-Situationen sind dann i.w. nur noch deren Frontknoten zu speichern, um bei einem eventuell notwendig werdenden Zurücksetzen der Front nach links die alten Werte wieder herstellen zu können.

Kapitel 6 KOMPLEXITÄTSBERECHNUNGEN

Die Komplexität der Programmsynthese aus den verschiedenen Arten von Beispielsfolgen wird in diesem Kapitel untersucht. In [Biermann 1972] und [Biermann, Baum, Petry 1975] sind zwar einige wenige statistische Angaben über die benötigte Rechenzeit einer implementierten Version der Synthese aus Instruktionsfolgen gemacht und in [Hwa, Wrightson 1973] ist eine Abschätzung der maximalen Suchraumgröße bei der Synthese von Turingmaschinen angegeben, aber weder in den drei zitierten Arbeiten noch z.B. in [Biermann, Krishnaswamy 1976] oder in [Raulefs 1973] sind Komplexitätsberechnungen oder Hinweise darauf zu finden.

In 6.1 wird nach Abschätzungen der Suchraumgröße des PSA PA der Komplexitätsbegriff für Algorithmen und Problemstellungen präzisiert. Die Komplexität einiger Teilprobleme der Programmsynthese wird in 6.2 betrachtet und eine etwas vereinfachte Version davon, das sog. Realisierungsproblem, eingeführt. Nach Definition der Komplexitätsklassen P-TIME und NP-TIME [Cook 1971], [Garey, Johnson 1979] in 6.3 wird in 6.4 die NP-Vollständigkeit des Realisierungsproblems sowie der Programmsynthese aus Beispielsfolgen nachgewiesen. Einige vereinfachende Einschränkungen werden in 6.5 betrachtet, und es wird gezeigt, welche davon ebenfalls NP-vollständig sind. Als Folgerung daraus wird in 6.6 die NP-Vollständigkeit der Synthese von Turingmaschinen aus Beispielsfolgen sowie einer stark vereinfachten Version dieses Problems nachgewiesen.

.1 Komplexität von Algorithmen und Problemstellungen

.1.1 Suchraumgröße von PA

Anhand von einigen Beispielen sollen Abschätzungen über die Größe des Suchraumes von PA vorgenommen werden. Zwei verschiedene Ansätze zur Abschätzung werden angegeben.

1. Es wird von dem Modell einer Aufzählung $E = (P_1, P_2, \ldots)$ aller Programme ausgegangen, wobei die Länge von P_i immer kleiner oder gleich der Länge aller P_{i+j} $(j \geq 1)$ ist. Für jedes PSP Q läßt sich sicherlich leicht eine obere Grenze Lmax angeben, sodaß Q, falls überhaupt realisierbar, zumindest Lmax-realisierbar ist; z.B. Lmax = Summe der Länge der Beispielsfolgen von Q. Die als minimale Realisierung in Frage kommenden Programme wären demnach alle die P_i, für die $i \in \{1, \ldots, r\}$ gilt, wobei die Länge von P_r gleich Lmax, die Länge von P_{r+1} aber gleich Lmax + 1 wäre. Ein Algorithmus, der PROGRAMS gemäß E aufzählt und bei jedem Programm überprüft, ob es Q realisiert, bis ein derartiges P gefunden worden ist, würde zwar ein minimales Programm liefern, müßte im ungünstigsten Fall aber tatsächlich alle r Programme überprüfen. Die folgenden Überlegungen dienen zur Abschätzung der Größe von r.

   Die Menge PROGRAMS wird wesentlich mitbestimmt durch die Menge der Zuweisungen ASSIGNMENTS = $\{A_1, \ldots, A_p\}$ und durch die Menge der Tests TESTS = $\{T_1, \ldots, T_q\}$; insbesondere ist die Anzahl der möglichen Programme mit l Knoten $(l \in \mathbb{N})$ ab-

hängig von den Mächtigkeiten dieser Mengen, also p und q.

Der Einfachheit halber werden nur die Fälle $b \in \{\underline{s}, \underline{ss}\}$ der Programmsynthese berücksichtigt; die Länge eines Programms P ist also die Summe aus Anzahl der Zuweisungs- und Anzahl der Testknoten von P.

Sei zunächst q = 0. Jedes Programm P der Länge l kann dann nur aus einer linearen Folgen von l Zuweisungsknoten bestehen, wobei jeder der l Knoten mit einem $A_i \in$ ASSIGNMENTS markiert ist. Daher gibt es

$$X = p^l$$

verschiedene Programme der Länge l und daher

$$\sum_{i=0}^{l} p^i = \frac{p^{l+1}-1}{p-1}$$

Programme der Länge kleiner oder gleich l, wobei mögliche Umbenennungen der LABEL unberücksichtigt sind. Für p=30 wie in den Beispielen der Kapitel 2 - 5 (vgl. Beispiel 2.1) gilt

$$\sum_{i=0}^{l} 30^i = \frac{30^{l+1}-1}{30-1}$$

und für l = 6 ergäbe dies mit 754 137 931 bereits eine unverhältnismäßig große Anzahl von Programmen, die selbst bei kleinsten PSP durchsucht werden müßte.

Ist q > 0, so können auch Verzweigungsknoten auftreten. Auf eine genaue Berechnung der Anzahl der Programme soll verzichtet werden, jedoch wird untersucht, welche Rolle

die Größen p und q in einer solchen Berechnung spielen.

Für $l \geq j$ sei ein Programmschema vom Typ T(l,j) im Prinzip aufgebaut wie ein Programm mit eindeutigem START- und STOP-Knoten, jedoch ohne Markierungen der Kanten und der übrigen Knoten, die sich in genau j Knoten mit einem Nachfolger und l-j Knoten mit zwei Nachfolgern aufteilen. $X_{l,j}$ gebe die Anzahl der verschiedenen Programmschemata vom Typ T(l,j) an. Demnach gibt es

$$\sum_{j=0}^{l} X_{l,j}$$

verschiedene Programmschema mit l Knoten (START- und STOP-Knoten werden wie bei Programmen nicht mitgezählt). Für jedes Programmschema vom Typ T(l,j) gibt es

$$p^j q^{l-j}$$

verschiedene Markierungen der Knoten (ohne Berücksichtigung der Markierung eines Knoten mit nur einem Nachfolger durch eine Testinstruktion) und

$$2^{l-j}$$

verschiedene Markierungen der Kanten, um ein Programm gemäß ASSIGNMENTS und TESTS zu erhalten. Es gibt also

$$\sum_{j=0}^{l} X_{l,j} \cdot p^j \cdot q^{l-j} \cdot 2^{l-j}$$

verschiedene Programme mit l Knoten und daher

$$\sum_{i=0}^{l} \sum_{j=0}^{i} X_{i,j} \cdot p^j \cdot q^{i-j} \cdot 2^{i-j}$$

verschiedene Programme mit l oder weniger als l Knoten.

Selbst wenn man alle auftretenden $X_{i,j}$ in der letzten Formel durch die Konstante 1 ersetzt - obwohl es i.a. sehr viel mehr als nur ein Programmschema vom Typ T(i,j) gibt -, wird deutlich, welchen Einfluß die Parameter p und q auf die Anzahl der Programme mit einer bestimmten Anzahl von Knoten haben. Für die Werte p = 30 und q = 5 aus Beispiel 2.1 ergeben sich dabei schon für l = 6 Werte der Größenordnung 10^{10}.

2. Der zweite Ansatz zur Abschätzung der Suchraumgröße von PA geht nicht von einer von dem vorliegenden PSP Q unabhängigen Aufzählung von PROGRAMS aus, sondern von der Standarddarstellung Q_{ST} = (b,S,MI,MT).

$$X = \prod_{i=1}^{m} \prod_{j=0}^{n_i} |MT_{i,j}|$$

gibt allgemein die Anzahl der verschiedenen vollständigen Instruktionshypothesen wieder, die gemäß MT möglich sind. Speziell für das PSP Q_3, dessen Standarddarstellung in Beispiel 3.1 - 3 angegeben ist, gilt

$$X_3 = \prod_{i=1}^{4} \prod_{j=0}^{n_i} |MT_{i,j}|$$

$$= 10 \cdot 9^2 \cdot 7^7 \cdot 4 \cdot 3^2 \cdot 2^{10}$$

$$= 7^7 \cdot 5 \cdot 3^6 \cdot 2^{13}$$

Geht man von der sicherlich zutreffenden, aber groben Abschätzung von

$$Lmax = \sum_{i=1}^{m} (n_i - 1)$$

als obere Grenze für die Anzahl der Programmknoten eines minimalen, Q realisierenden Programmes P aus und nimmt man l als größtes mögliches LABEL, wenn P als ein Programm der Länge l angesehen wird, so wären unter diesen Annehmen maximal

$$Y = \sum_{l=1}^{Lmax} l^{lmax}$$

verschiedene vollständige LABEL-Hypothesen zu berücksichtigen, die Anzahl der Kombinationen von vollständigen Instruktions- und LABEL-Hypothesen wäre somit

$$X \cdot Y = \left(\prod_{i=1}^{m} \prod_{i=1}^{n_i} |MT_{i,j}| \right) \cdot \left(\sum_{l=1}^{Lmax} l^{lmax} \right)$$

Selbst auf das relativ einfache PSP Q_3 bezogen werden mit

$$Lmax_3 = 8 + 5 + 5 + 8 = 26$$

$$Y_3 = \sum_{l=1}^{26} l^{26}$$

derartige Größenordnungen erreicht, sodaß eine solch rein kombinatorische Vorgehensweise völlig unrealistisch wird.

Ein Vergleich der hier unter 1. und 2. genannten Abschätzungen mit der in Abschnitt 5.3 angegebenen Beispielsberechnungen von PA bei Eingabe Q_3 deutet die durch PA geleistete Suchraumreduktion an.

6.1.2 Größenkriterien für Programmsyntheseprobleme

Bei der Abschätzung der Suchraumgröße von PA im vorigen Abschnitt wurde deutlich, daß diese von einer Reihe von Parametern abhängt. Im einzelnen können für ein PSP $Q = (b,M)$, wobei $M = \{M_1,\ldots,M_m\}$ und $dim(Q) = (m,(n_1,\ldots,n_m))$, als Kriterien für die Schwierigkeit von Q unterschieden werden:

(1) m - die Anzahl der Beispielsfolgen

(2) $n_1,\ldots,n_m$ - die Längen der einzelnen Beispielsfolgen gemessen an der Anzahl der Elemente

(3) $\bar{n}_1,\ldots,\bar{n}_m$ - die Längen der einzelnen Beispielsfolgen unter Berücksichtigung der Längen der darin enthaltenen Speicherbelegungsbeschreibungen

(4) p - die Anzahl der Q zugrunde liegenden Zuweisungsinstruktionen: |ASSIGNMENTS| = p

(5) q - die Anzahl der Q zugrunde liegenden Testinstruktionen: |TESTS| = q

(6) r - die Mächtigkeit des Alphabets zur Beschreibung der Speicherbelegungen: $|\Sigma| = r$

(7) b - die Problemspezifizierungsmarke von Q

Zur Vereinfachung sei folgendes vereinbart. Als Größenkriterium der Länge der Beispielsfolgen kann auch lediglich das Maximum betrachtet werden; dann ist (2) und (3) zu ersetzen durch

(2') $n - n = \max(n_1,\ldots,n_m)$

(3') $\bar{n} - \bar{n} = \max(\bar{n}_1,\ldots,\bar{n}_m)$

Es ist zu beachten, daß hier lediglich von der Anzahl der Zuweisungen und Tests die Rede ist, nicht jedoch davon, wie kompliziert oder einfach die einzelnen Instruktionen selbst sind. Es macht also z.B. für die folgenden Ausführungen keinen Unterschied, ob eine Zuweisungsinstruktion $A = \lambda x.\ x+1$ sehr einfach oder $A' = \lambda x.\ 3x^8 + 5x^2 + 27x$ etwas aufwendiger zu berechnen ist; entsprechendes gilt für die Testinstruktionen.

Auf die genaue Spezifizierung der Größe von Q soll hier zunächst verzichtet werden; dies wird später in Abschnitt 6.3.1 im Zusammenhang mit einer weiteren Formalisierung geschehen. Oft reicht es aus, z.B. mit den oben genannten Werten m,n,p,q,r als 5-Tupel zu operieren; ist ein einzelner numerischer Wert erwünscht, so bezeichne SIZE(Q) den Wert $m\cdot n + p + q + r$.

6.1.3 Komplexität eines Algorithmus

Unter der Komplexität eines Algorithmus A wird in dieser Arbeit immer die Zeitkomplexität von A verstanden, also der Zeitaufwand, den A für eine Eingabe x benötigt, ausgedrückt als eine Funktion der Größe von x. Der Grenzwert einer solchen Funktion für den Fall, daß die Größe der Eingaben wächst, wird als asymptotische Zeitkomplexität bezeichnet und stellt

ein wichtiges Beurteilungskriterium für die Bewertung von A dar, da insbesondere dadurch die maximale Größe festgelegt wird, bis zu welcher die Eingaben von A mit realistischem Zeitaufwand bearbeitet werden können. Dabei kommt es i.a. nicht so sehr auf die genaue Spezifikation der Zeitkomplexität an, sondern auf die Größenordnung.

Die Größenordnung einer Funktion g(n) wird wie üblich mit O(f(n)) bezeichnet, wenn es eine Konstante c gibt, sodaß für fast alle n $g(n) \leq c\cdot f(n)$ gilt.

Nicht näher spezifiziert werden muß der Begriff "Zeitaufwand" eines Algorithmus A. Es genügt anzunehmen, daß dies die Anzahl der von A ausgeführten elementaren Berechnungsschritte ist.

Beispiel 6.1

1. Ist A ein Algorithmus zur Matrizenmultiplikation, so ist der Zeitaufwand von A bei Eingabe (M,N) die Anzahl der von A bei der Multiplikation von M und N benötigten arithmetischen Operationen.
2. Ist A eine (deterministische) Turingmaschine, so gibt
   $T(n) = \max\{k \mid$ es gibt ein Eingabewort der Länge n, für das A nach k Lese/Schreiboperationen anhält$\}$
   die Zeitkomplexität von A wieder.

6.1.4 Komplexität einer Problemstellung

Für eine bestimmte Problemstellung P gibt es i.a. eine ganze Reihe verschiedener Lösungsalgorithmen $A_1,\ldots,A_n$. Unter der Zeitkomplexität von P wird die Zeitkomplexität eines Lösungsalgorithmus A verstanden, dessen Zeitkomplexität minimal ist.

Aus dieser Definition wird deutlich, weshalb es sich als bedeuted schwieriger erweist, die Komplexität T(n) einer Problemstellung P zu bestimmen, als die Komplexität eines einzelnen Algorithmus. Zum einen muß die Existenz eines Algorithmus mit der Komplexität T(n) nachgewiesen werden, zum anderen muß gezeigt werden, daß es keinen Algorithmus für P mit kleinerer Komplexität geben kann. Insbesondere ist es also für den zweiten Schritt nicht ausweichend, etwa nur alle bekannten Lösungsalgorithmen für P zu berücksichtigen. Oft ist aus diesem Grund eine genaue Angabe der Zeitkomplexität von P sehr schwierig; daher dient die zuvor schon eingeführte Notation O(T(n)) wieder zur Angabe der Größenordnung von T(n).

In den Abschnitten 6.2 - 6.5 werden Aussagen über die Komplexität der Programmsynthese aus Beispielsfolgen gemacht. Wie auch schon in diesem Abschnitt ist mit Komplexität einer Problemstellung i.f. immer deren Zeitkomplexität gemeint.

6.2 Teilprobleme der Programmsynthese

6.2.1 Komplexität der Instruktionshypothesengraphberechnung

Der erste Schritt des PSA PA ist die Berechnung des IH-Graphen des Eingabe-PSP Q. Dieser wiederum kann vom Konzept her in die folgenden Teilschritte aufgegliedert werden (vgl. Kap.3):

1. Bestimmung der Standarddarstellung Q_{ST} von Q:
   Q_{ST} = (b,S,MI,MT)
2. Bestimmung der Stufengraphdarstellung von MI
   [2.1 Überprüfen der Realisierbarkeit von Q]
3. Erzeugen der Stufengraphdarstellung von Q
4. Erzeugen von IH-GRAPH(Q)

Bei einer aktuellen Realisierung von PA würden diese einzelnen vier Schritte natürlich nicht unabhängig voneinander durchgeführt, sondern miteinander verknüpft. Dies kann hier jedoch unberücksichtigt bleiben, denn es gilt, daß jeder der Schritte 1. - 4. in polynomialer Zeit ausgeführt werden kann, d.h. für $i\varepsilon\{1,\ldots,4\}$ gibt es ein Polynom P_i, sodaß die Ausführung von Schritt i nicht mehr als $P_i(m,n,p,q,r)$ Zeiteinheiten benötigt, wenn Q als Eingabe vorliegt und m,n,p,q,r die Größenkriterien von Q gemäß Abschnitt 6.1.2 sind. Auf die Frage, was als Zeiteinheit zu verstehen ist, muß hier nicht weiter eingegangen werden, da eine entsprechende Anpassung der Koeffizienten von P_i es ermöglicht, die Zeiteinheiten nahezu beliebig zu wählen.

Auf eine Formalisierung und einen Beweis zur Existenz der Polynome P_i wird verzichtet; er ergibt sich leicht, wenn man für die in Kap. 3 angegebenen einzelnen Berechnungsabschnitte jeweils die polynomiale Zeitbegrenzung zeigt und ausnutzt, daß Summe bzw. Produkt von zwei Polynomen wieder ein Polynom ergeben.
Als Folgerung erhält man sofort:

Satz 6.1

Die Berechnung von IH-GRAPH(Q) nach der in Kapitel 3 angegebenen Vorgehensweise ist für alle Q zeitlich durch ein Polynom P in den Variablen m,n,p,q,r begrenzt, wobei m,n,p,q,r die Größe von Q wiedergeben und Q selbst ein PSP ist.

Beweis: folgt aus den vorhergehenden Überlegungen
(z.B. $P = P_1 + P_2 + P_3 + P_4$) .

Es ist zu beachten, daß Satz 6.1 insofern eine relativ schwache Aussage liefert, da an das Polynom P keine weiteren Anforderungen gestellt werden, etwa was den Grad von P betrifft. Diesbezüglich können weitaus schärfere Aussagen gemacht werden, jedoch soll darauf verzichtet werden, da Satz 6.1 für den i.f. zu führenden Nachweis der NP-Vollständigkeit einzelner Problemklassen eine ausreichende Grundlage bietet.

6.2.2 Das Realisierungsproblem

Der Programmsynthesealgorithmus PA liefert für ein realisierbares PSP Q ein minimales Programm P, das Q realisiert. Die Fragestellung nach einem minimalen l, sodaß Q l-realisierbar ist, kann als gleichwertige Formulierung des PSP angesehen werden, da bei einem minimalen Programm P die Länge von P das gesuchte l darstellt, umgekehrt aber die Tatsache, daß l minimal ist, im Prinzip durch die Konstruktion eines minimalen Programmes P nachgewiesen werden muß.

Eine wesentliche Vereinfachung der Problemstellung scheint zunächst die Reduzierung auf eine Ja/Nein-Entscheidung zu sein, indem ein Index l zusammen mit Q vorgegeben und gefragt wird, ob Q l-realisierbar ist.

Definition 6.1

Das Realisierungsproblem ist das folgende:
Gegeben sei ein PSP Q und ein $l \in \mathbb{N}$, ist Q l-realisierbar?
Ein Algorithmus, der für jede Instanz (Q,l) des Realisierungsproblems entscheidet, ob Q l-realisierbar ist oder nicht, heißt Realisierungsalgorithmus.

Es stellt sich die Frage nach der 'Schwierigkeit' des Realisierungsproblems im Vergleich zu Programmsyntheseproblemen.

6.2.3 Komplexitätsvergleich von Realisierungs- und Programmsyntheseproblem

Um einen solchen Vergleich durchführen zu können, wird die polynomiale Verwandtschaft zwischen Funktionen als Größenordnungsrelation eingeführt [Garey,Johnson 1979] .

Definition 6.2

Zwei Funktionen $f_1(n)$ und $f_2(n)$ sind polynomial verwandt <=> es gibt Polynome $p_1(x)$ und $p_2(x)$, sodaß für alle $n \in \mathbb{N}$ $f_1(n) \leq p_1(f_2(n))$ und $f_2(n) \leq p_2(f_1(n))$.

Der folgende Satz sagt aus, daß sich die Schwierigkeitsgrade von PSP und Realisierungsproblem bezogen auf den benötigten Zeitaufwand höchstens polynomial unterscheiden.

Satz 6.2

(1) Zu jedem PSA A gibt es einen Realisierungsalgorithmus A', sodaß die Zeitkomplexitäten von A und A' polynomial verwandt sind.

(2) Zu jedem Realisierungsalgorithmus A' gibt es einen PSA A, sodaß die Zeitkomplexitäten von A' und A polynomial verwandt sind.

Beweis:

Aussage (1) ist trivial: ein einfacher Algorithmus A' kann z.B. bei Eingabe (Q,l) aus der Anwendung von A auf Q bestehen, gefolgt von der Ausgabe "Ja" (d.h. Q ist l-realisierbar) genau dann, wenn A(Q) ein Programm der Länge k liefert und $k \leq l$ gilt; andernfalls wird "Nein" (Q ist nicht l-realisierbar) ausgegeben.

Zu (2): Sei A' ein Realisierungsalgorithmus mit der Zeitkomplexität F(n). Da A' bei Eingabe (Q,l) zur Entscheidung der Frage, ob Q l-realisierbar ist, im Prinzip ein Programm der Länge l konstruieren muß, kann effektiv ein Algorithmus A" mit folgenden Eigenschaften gefunden werden:

(i) Bei Eingabe (Q,l) liefert A" ein Q realisierendes Programm P der Länge l genau dann, wenn A angewandt auf (Q,l) "Ja" (d.h. Q ist l-realisierbar) ergibt; ansonsten liefert A" die leere Menge.

(ii) Die Zeitkomplexität von A" ist O(F(SIZE(Q))).

Für ein PSP Q kann nach Satz 6.1 in polynomialer Zeit festgestellt werden, ob Q überhaupt realisierbar ist. Dieser Schritt benötigt demnach maximal O(P(SIZE(Q))) Zeiteinheiten, wobei P ein Polynom ist.

Ist Q realisierbar, so ist Q zumindest Lmax-realisierbar, wobei Lmax das Produkt aus Anzahl und maximaler Länge der Beispielsfolgen (oder genauer: Lmax die Summe der Längen der Beispielsfolgen) ist. Da diese beiden Parameter die Größe von Q als Faktoren mitbestimmen, folgt, daß Lmax von der Größenordnung O(SIZE(Q)) ist. Weiterhin gibt es ein $Lmin \in \{0,\ldots,Lmax\}$, sodaß gilt:

(iii) $\forall l \in \{0,\ldots,Lmin-1\}$. Q ist nicht l-realisierbar.

$\forall l \in \{Lmin,\ldots,Lmax\}$. Q ist l-realisierbar.

Ein PSA A kann damit wie folgt arbeiten. Nach dem Überprüfen der Realisierbarkeit von Q besteht A aus einem wiederholten Aufruf von A" mit den Eingaben (Q,l), wobei l die Werte 0,1,2,... durchläuft, bis A"(Q,l) zum ersten Mal nicht die leere Menge liefert; ist dies der Fall, so gibt A eben dieses Ergebnis von A"(Q,l) aus.

A benötigt maximal Lmax = O(SIZE(Q)) Aufrufe von A"; ein Aufruf von A" benötigt O(F(SIZE(Q))) Zeiteinheiten. Somit ist die gesamte Zeitkomplexität von A gegeben durch

$O(P(SIZE(Q))) + O(SIZE(Q)) \cdot O(F(SIZE(Q))) =$

$O(P(SIZE(Q))) + O(SIZE(Q) \cdot F(SIZE(Q)))$ und daher polynomial verwandt mit der Komplexität von A'.

(Bemerkung: Effizienter als das jeweilige Inkrementieren von l um 1 wäre natürlich das sich hier wegen (iii) anbietende binäre Suchen; dabei gäbe es maximal nur $\lceil \log_2 Lmax \rceil + 1$ Aufrufe von A".)

Dies schließt den Beweis von Satz 6.2 ab.

6.3 Komplexitätsklassen

6.3.1 Die Klassen P-TIME und NP-TIME

Es wird mit Hilfe des letzten Satzes gezeigt werden, daß Programmsynthese und Realisierungsprobleme bzgl. zweier wichtiger Komplexitätsklassen denselben Schwierigkeitsgrad besitzen. Die beiden Komplexitätsklassen sind zum einen die Menge aller der Problemstellungen, für die es einen deterministischen Algorithmus polynomialer Zeitkomplexität gibt, zum anderen die Menge aller der Problemstellungen, für die es einen nicht-deterministischen Algorithmus ebenfalls mit polynomialen Zeitaufwand gibt. Da es sich hierbei um Aussagen über die Menge aller Algorithmen handelt, wird als Grundlage bei der Definition dieser Klassen von Turingmaschinen als Algorithmusbegriff ausgegangen. Als Problemstellungen werden Ja/Nein-Entscheidungen zugrunde gelegt, und diese seien als Spracherkennungsprobleme dargestellt [Garey, Johnson 1979].

Definition 6.3

Σ sei ein Alphabet.

(1) P-TIME = $\{L \mid L \subseteq \Sigma^*$ und es existiert ein Polynom P(n) und eine deterministische Turingmaschine M mit der Zeitkomplexität P(n), die L akzeptiert$\}$

(2) NP-TIME= $\{L \mid L \subseteq \Sigma^*$ und es existiert ein Polynom P(n) und eine nicht-deterministische Turingmaschine M

mit der Zeitkomplexität P(n), die L akzeptiert }

(3) Die Elemente von P-TIME (bzw. NP-TIME) werden auch als Problemklassen bezeichnet.

Im Vergleich zu der bisher benutzten Terminologie ergeben sich einige Besonderheiten:

- Nur Ja/Nein-Entscheidungsprobleme werden in Def. 6.3 betrachtet; daher können PSP nicht direkt als Spracherkennungsprobleme behandelt werden; bei Realisierungsproblemen, die ja gerade Entscheidungsprobleme sind, ist dies möglich.
- Programmsynthese- und Realisierungsalgorithmen müßten als Turingmaschinen angegeben werden.

Es wird daher folgendes vereinbart:

1. Die Standardcodierung eines Realisierungsproblems (Q,l), $Q = (b,\{M_1,\ldots,M_m\})$, ist ein Wort w über einem universellen Alphabet Σ', wobei $w = l \,\&\, b \,\&\, M_1\&\ldots\&M_m$ die Konkatenation der einzelnen Beispielsfolgen von Q zusammen mit den Parametern l und b darstellt und das Sonderzeichen '&' sonst nicht in w auftritt. Die Größe von (Q,l) ist die Länge von w und wird bezeichnet durch $|w|$.
   Bemerkung: Ebenso wie bei PA in Kapitel 5 werden hier die Eingabeparameter ASSIGNMENTS und TESTS vernachlässigt.

2. Eine Klasse K von Realisierungsproblemen ist in P-TIME (bzw. NP-TIME), wenn die Menge der Standardcodierungen der Elemente von K in P-TIME (bzw. NP-TIME) ist.

3. Analog zu 1. und 2. ist für andere Ja/Nein-Problemklassen Standardcodierung, Größe und Zugehörigkeit zu P-TIME bzw. NP-TIME definiert.

4. Obwohl P-TIME und NP-TIME über Turingmaschinen definiert wurden, werden diese beiden Begriffe erweitert benutzt werden. Da für eine Reihe von Berechnungsmodellen polynomiale Verwandtschaft bzgl. ihrer Zeitkomplexität zu den entsprechenden Turingmaschinen besteht, werden auch Problemklassen, die nicht unbedingt Ja/Nein-Entscheidungsprobleme darstellen, als zu P-TIME (bzw. NP-TIME) gehörig bezeichnet, wenn es für sie einen deterministischen (bzw. nicht-deterministischen) Algorithmus polynomialer Zeitkomplexität in einem solchen Berechnungsmodell gibt.

Beispiel 6.2

(1) $L = \{Q \mid Q \in PSP^{fin}_{I,S} \wedge Q \text{ ist realisierbar}\}$ ist in P-TIME, da nach Satz 6.1 in polynomialer Zeit für ein PSP Q IH-GRAPH(Q) berechnet werden kann und diese Berechnung nach Satz 3.5 und Def. 3.8 genau dann ε liefert, wenn Q nicht realisierbar ist.

(2) Da L in P-TIME ist, ist L auch in NP-TIME, denn nach Definition dieser beiden Klassen folgt sofort P-TIME $\subseteq$ NP-TIME. Es ist bisher aber noch nicht gelungen, nachzuweisen, daß es sich dabei um eine echte Inklusion handelt oder ob, was allerdings nicht anzunehmen ist, die beiden Klassen identisch sind [Garey, Johnson 1979].

In den beiden folgenden Sätzen wird gezeigt, daß Realisierungs- und Programmsyntheseproblem in NP-TIME sind.

Satz 6.3

L bezeichne die Menge aller endlichen "lösbaren" Realisierungsprobleme, d.h. $L = \{(Q,l) \mid Q \in PSP^{fin}_{I,S} \wedge l \in \mathbb{N} \wedge Q \text{ ist } l\text{-realisierbar}\}$.

Behauptung: L ist in NP-TIME

Beweis:

(Q,l) sei ein Realisierungsproblem. In polynomialer Zeit kann nach Satz 6.1 G = IH-GRAPH(Q) durch das Verfahren von Kapitel 3 berechnet werden. Liegt G vor und ist $G \neq \varepsilon$, so kann in nicht-deterministischer Vorgehensweise analog zu der in Abschnitt 3.3 beschriebenen Hypothesenbildung an jedem Knoten K von G jedes mögliche LABEL- und Instruktionshypothesenpaar (r,I) "paralell" (d.h. nicht-deterministisch) angenommen werden, wobei I aus der IH-Menge von K und $r \in \{1,\ldots,l\}$ ist (LABEL größer als l sind irrelevant). Die Überprüfung, ob eine sich so als U-Graph UG ergebende Tripelmenge die Eindeutigkeitsanforderungen von Def. 3.13 erfüllt, kann in $O(n^2)$ Schritten überprüft werden, wobei n die Anzahl der Knoten von G ist. Ist dies der Fall und darüberhinaus die Anzahl der Knoten von UG kleiner oder gleich l, so ist Q l-realisierbar und es wird "Ja" ausgegeben; andernfalls ist Q nicht l-realisierbar und es erfolgt ebenso wie bei $G = \varepsilon$ die Ausgabe "Nein".

Das hier skizzierte Verfahren arbeitet nicht-deterministisch, aber in polynomialer Zeit, woraus die Behauptung folgt.

Satz 6.4:

$PSP^{fin}_{I,S}$ ist in NP-TIME

Beweis:

Die Aussagen von Satz 6.2 gelten auch für nicht-deterministische Algorithmen; damit folgt die Behauptung aus Satz 6.3 .

In Abschnitt 6.4 werden die Aussagen der Sätze 6.3 und 6.4 über die Zugehörigkeit zu NP-TIME für den Nachweis der NP-Vollständigkeit dieser Problemstellungen benötigt.

6.3.2 NP-Vollständigkeit

Aufgrund der beiden letzten Sätze stellt sich die Frage, ob die Menge der Realisierungs- bzw. Programmsyntheseprobleme nicht nur in NP-TIME, sondern auch in P-TIME ist, d.h. ob es dafür einen deterministischen Algorithmus polynomialer Zeitkomplexität gibt.

Diese Frage wird in dieser Arbeit nicht beantwortet werden, aber es wird gezeigt, daß die Antwort darauf eines der wichtigsten noch offenen Probleme der Komplexitätstheorie lösen

würde, nämlich ob P-TIME = NP-TIME gilt oder nicht. Dies geschieht durch den Nachweis, daß Realisierungs- bzw. Programmsyntheseprobleme zu den schwierigsten Problemklassen in NP-TIME gehören, d.h.: fände man einen deterministischen polynomialen Algorithmus dafür, so könnte man damit für jede andere Problemklasse aus NP-TIME ebenfalls einen deterministischen polynomialen Algorithmus konstruieren. Problemklassen, die dieses Schwierigkeitskriterium erfüllen, heißen NP-vollständig [Cook 1971], [Garey, Johnson 1979].

Definition 6.4

K sei eine Problemklasse aus NP-TIME.

K heißt NP-vollständig <=>

Wenn es einen deterministischen Algorithmus A für K mit Zeitkomplexität T(n), $T(n) \geq n$, gibt, so kann für jede Problemklasse K' aus NP-TIME effektiv ein deterministischer Algorithmus mit Zeitkomplexität T(p(n)) gefunden werden, wobei p ein von K' abhängiges Polynom ist. In diesem Fall heißt K' reduzierbar zu K.

Im folgenden Abschnitt wird der Begriff der Reduzierbarkeit durch eine andere Relation präzisiert.

6.3.3 Polynomiale Transformierbarkeit

Die Relation "polynomial transformierbar" [Garey, Johnson 1979] erweist sich als nützlich für den Nachweis der NP-Vollständigkeit von Problemklassen.

Definition 6.5

K und K' seien zwei Problemklassen. Σ und Σ' seien die Alphabete, in denen die Standardcodierungen von K und K' vorliegen.

K' ist polynomial transformierbar zu K <=>
Es gibt ein Polynom p(n) und einen deterministischen Algorithmus A polynomialer Zeitkomplexität, der eine Abbildung

$$A : \Sigma'^{*} \longrightarrow \Sigma^{*}$$

definiert, sodaß gilt:

$$\forall_{x \in \Sigma'^{*}}. \; (x \in K' \iff A(x) \in K) \wedge (|A(x)| \leq p(|x|))$$

Die Bedingung $A(x) \leq p(|x|)$ in obiger Definition stellt sicher, daß kein Element x von K' einem Element von K zugeordnet wird, dessen Größe die Größe von x um mehr als einen polynomialen Faktor übersteigt. Der Einfachheit halber sei i.f. bei Algorithmen zur polynomialen Transformation immer angenommen, daß das Polynom p sowohl die Zeitkomplexität wie auch eine obere Grenze für die Länge der zugeordneten Wörter angibt.

Satz 6.5

K und K' seien zwei Problemklassen. A_1 sei ein Lösungsalgorithmus für K mit Zeitkomplexität T(n), $T(n) \geq n$; K' sei polynomial transformierbar zu K.

Behauptung: K' ist reduzierbar zu K.

Beweis:
Da K' polynomial transformierbar ist zu K, gibt es einen Algorithmus $A : \Sigma'^* \longrightarrow \Sigma^*$ und ein Polynom p wie in Def. 6.5. Ein Lösungsalgorithmus A_2 für K' kann nun zunächst für jedes $x \in K'$ in der Anwendung von A auf x bestehen; dieser Schritt benötigt höchstens $p(|x|)$ Zeiteinheiten, und es gilt $A(x) \leq p(|x|)$. A_1 liefert bei Eingabe A(x) das gewünschte Ergebnis des gesuchten Algorithmus A_2 bei Eingabe x; also $A_2 = \lambda x.\ A_1(A(x))$. Dieser zweite Teilschritt benötigt $T(|A(x)|)$ und wegen $|A(x)| \leq p(|x|)$ und der Monotonie von T somit höchstens $T(p(|x|))$, der gesamte Algorithmus A_2 maximal $p(|x|) + T(p(|x|))$ Zeiteinheiten. Da T Zeitkomplexitätsfunktion, kann o.B.d.A. $n + T(n) \leq T(2 \cdot n)$ angenommen werden; damit ist $T(2 \cdot p(|x|))$ eine obere Grenze für den von A_2 benötigten Zeitaufwand. Es folgt, daß K' reduzierbar ist zu K.

6.4 Komplexität von Realisierungs- und Programmsyntheseproblem

Um zu zeigen, daß eine Problemklasse K NP-vollständig ist, muß zum einen die Zugehörigkeit von K zu NP-TIME, zum anderen für jedes K' aus NP-TIME die polynomiale Transformierbarkeit zu K nachgewiesen werden. Der erste Punkt ist wegen der Möglichkeit der Nicht-Determiniertheit des betreffenden Algorithmus of relativ leicht zu erfüllen (vgl. Satz 6.3); der zweite Punkt ist insofern schwieriger, da er für alle K' durchzuführen ist. Ist jedoch für irgendein K_o erst einmal die NP-Vollständigkeit nachgewiesen worden, so genügt es, für den Nachweis der NP-Vollständigkeit eines anderen K aus NP-TIME lediglich für K_o die polynomiale Transformierbarkeit zu K zu zeigen: jedes K' aus NP-TIME wäre dann zumindest über den "Umweg" über K_o polynomial transformierbar zu K.

Es wurde gezeigt, daß das sog. Erfüllbarkeitsproblem, ob es nämlich für einen booleschen Ausdruck B eine Variablenbelegung gibt, unter der B true liefert, NP-vollständig ist [Cook 1971]. An anderer Stelle wurde bewiesen, daß das Erfüllbarkeitsproblem polynomial transformierbar ist zu dem sog. Einfärbungsproblem.

Definition 6.6

(1) G = (V,E) sei ein ungerichteter Graph, $k \in \mathbb{N}$.
G ist k-färbbar <=>
Es gibt eine Abbildung $F : V \longrightarrow \{1,\ldots,k\}$, die benachbarten Knoten von G jeweils verschiedene Farben, das sind

die Zahlen aus $\{1,\ldots,k\}$, zuordnet:

$$\forall (v,v') \in E.\ F(v) \neq F(v')$$

(2) Das Einfärbungsproblem ist das folgende:

Gegeben sei ein ungerichteter Graph G und ein $k \in \mathbb{N}$,

ist G k-färbbar?

Das Einfärbungsproblem für ungerichtete Graphen ist in NP-TIME, und es gehört zu den schwierigsten Problemen dieser Klasse, denn es gilt:

Satz 6.6

Das Einfärbungsproblem ist NP-vollständig.

Beweis

durch polynomiale Transformation des Erfüllbarkeitsproblems [Aho, Hopcraft, Ullman 1974].

Mit Hilfe von Satz 6.6 wird die NP-Vollständigkeit von Realisierungs- (6.4.1) und Programmsyntheseproblem (6.4.2) gezeigt.

6.4.1 NP-Vollständigkeit des Realisierungsproblems

Satz 6.7

Das Realisierungsproblem für PSP ist NP-vollständig.

Beweis:

Es wird gezeigt, daß das Einfärbungsproblem für Graphen polynomial transformierbar ist zu dem Realisierungsproblem für PSP, welches nach Satz 6.3 in NP-TIME ist. Mit Satz 6.6 folgt die NP-Vollständigkeit des Realisierungsproblems.

Gegeben sei ein ungerichteter Graph $G = (V,E)$ und ein $k \in \mathbb{N}$. Es wird ein PSP Q konstruiert, das genau dann k-realisierbar ist, wenn G k-färbbar ist.

Sei $V = \{v_1,\ldots,v_s\}$, $E = \{e_1,\ldots,e_t\}$. O.B.d.A. können folgende Annahmen über G gemacht werden:

1. G ist verbunden, d.h. für alle v_i, v_j aus V mit $i \neq j$ gibt es einen Weg in G von v_i nach v_j.

2. Für alle $v \in V$ gilt $(v,v) \notin E$.

3. V sei linear geordnet durch $v_1 < \ldots < v_s$.

4. Da G ungerichteter Graph, ist $(v,v') \in E$ äquivalent zu $(v,v') \in E$. Die Kanten von G seien i.f. jedoch grundsätzlich so angegeben, daß der unter der linearen Ordnung von V kleinere Knoten immer an erster Stelle steht, d.h. $(v,v') \in E \Rightarrow v < v'$.

Q = (b,M) ist ein PSP aus zuweisungsgetreuen Speicherbelegungsfolgen, also b = as. Die Q zugrunde liegende Menge der möglichen Speicherbelegungen STORES sind die Wörter der Länge 1,2 oder 3 über einem Alphabet Σ, das für jeden Knoten v_i von G und für jede Kante e_j von G ein spezielles Symbol a_i bzw. a'_j enthält:

$$\Sigma = \{a_1,\ldots,a_s\} \cup \{a'_1,\ldots,a'_t\}$$

$$\text{STORES} = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$$

Es gibt nur eine einzige Zuweisungsinstruktion, aber s + t verschiedene Testinstruktionen

$$\text{ASSIGNMENTS} = \{A\}$$

$$\text{TESTS} = \{T_1,\ldots,T_s\} \cup \{T'_1,\ldots,T'_t\}$$

die definiert sind durch:

$$A = \lambda w.\ w = aw' \longrightarrow w'$$

$$T_i = \lambda w.\ w = aw' \longrightarrow a = a_i$$

$$T'_j = \lambda w.\ w = aw' \longrightarrow a = a'_j$$

wobei $i \in \{1,\ldots,s\}$, $j \in \{1,\ldots,t\}$, $a \in \Sigma, w' \in \Sigma^*$.
Die Menge der Beispielsfolgen von Q enthält 2·t Speicherbelegungsfolgen:

$$M = \{M_1,\ldots,M_t\} \cup \{M'_1,\ldots,M'_t\}.$$

Jeder Kante e_r von G entsprechen genau zwei dieser Speicherbelegungsfolgen, nämlich M_r und M'_r. Für $r \in \{1,\ldots,t\}$ sei $e_r = (v_i,v_j)$, wobei gemäß obiger Vereinbarung i < j .

Dann sind M_r und M'_r gegeben durch:

$$M_r = (a_i\ a'_r\ a'_r\ ,\ a'_r\ a'_r)$$

$$M'_r = (a_i\ a'_r\ a'_r\ ,\ a'_r\ a'_r\ ,\ a'_r)$$

Damit ist Q vollständig spezifiziert. Die Mächtigkeit von Σ und der Instruktionenmengen von Q ist linear von s und t abhängig. Q enthält 2·t Beispielsfolgen maximaler Länge 3, wobei jede auftretende Speicherbelegung durch ein Wort höchstens der Länge 3 gegeben ist. Die Größe der Standardcodierung von Q ist somit O(t), während auch die für die Konstruktion von Q benötigte Zeit sicherlich nicht mehr als polynomial mit s und t anwächst. Für den Nachweis, daß es sich hier um eine polynomiale Transformation im Sinne von Def. 6.5 handelt, bleibt also noch zu zeigen: G ist k-färbbar genau dann, wenn Q k-realisierbar ist.

Der Beweis hierzu beruht auf folgendem: Alle in Q auftretenden Speicherbelegungsübergänge können von der Zuweisungsinstruktion A erzeugt werden, die auch gleichzeitig die einzig mögliche Instruktion ist. Ein Q realisierendes Programm P muß aber mehrere mit A markierte Knoten haben, da es wegen der Folgen in Q Übergänge von einem A-Knoten zum STOP-Knoten bzw. wiederum zu einem A-Knoten geben muß, die nicht durch eine Testinstruktion unterschieden werden können. Jeder Knoten von G kann eindeutig einem A-Knoten von P zugeordnet werden. Wenn G k-färbbar ist, so genügen k verschiedene A-Knoten in P, wobei die Färbung von G die LABEL der A-Knoten liefert; ist andererseits Q k-realisierbar, so enthält P maximal k

A-Knoten, wobei die LABEL dieser Knoten eine Einfärbung von G darstellen.

Um Letzteres zu beweisen, werden die in den Kapiteln 2 - 5 entwickelten Verfahren benutzt. Für die Standarddarstellung Q_{ST} = (as,S,MI,MT) von Q gilt:

(i) Die Speicherbelegungsmatrix S enthält als Zeilen genau die 2 t·Folgen aus M. Diese seien wie folgt indiziert:

$$\forall\, r\varepsilon\{1,\ldots,t\}.\ S_r = M_r$$

$$\forall\, r\varepsilon\{t+1,\ldots,2\cdot t\}.\ s_r = M'_{r-t}$$

Sei $r\varepsilon\{1,\ldots,t\}$. M_r und M'_r entsprechen also der Kante e_r von G. Für $e_r = (v_i,v_j)$ gilt:

(ii) Die Matrix der Instruktionen MI enthält für M_r und M'_r die Zeilen:

$$MI_r = (\{\underline{START}\},\{A\},\{\underline{STOP}\})$$

$$MI_{t+r} = (\{\underline{START}\},\{A\},\{A\},\{\underline{STOP}\})$$

(iii) Die Matrix der Transitionen MT enthält für M_r und M'_r die Zeilen:

$$MT_r = (\{T_i\},\{T'_r\})$$

$$MI_{t+r} = (\{T_j\},\{T'_r\},\{T'_r\})$$

Für die Konstruktion des IH-Graphen IHG von Q läßt sich folgendes beobachten: MT enthält in der ersten Spalte genau s verschiedene Elemente, nämlich $\{T_1\},\ldots,\{T_s\}$, die genau den Knoten von G entsprechen. Dadurch wird eine s-elementige

Zerlegung der Elemente der zweiten Spalte von MI definiert, sodaß IHG s Knoten der Stufe 1 besitzt, die alle mit {A} markiert sind. Auf der zweiten Stufe besitzt IHG 2·t Knoten, deren Indexmengen jeweils einelementig und die entweder mit {STOP} oder mit {A} markiert sind, während auf Stufe drei alle t Knoten die Markierung **{STOP}** besitzen. Es gibt in IHG keine redundanten Indizes, und da alle Knoten mit einer einelementigen Instruktionsmenge markiert sind, gibt es genau eine einzige aus IHG ableitbare terminale IH-Situation IS. Diese ist in Abb. 6.1 skizziert. Der IH-Baum T von IS ergibt sich, wenn man in Abb. 6.1 die Knotenmarkierungen wie A durch {A} ersetzt.

Für einen Knoten v_i aus G bezeichne K_i den Knoten von IHG der Stufe 1, der vom START-Knoten aus über die Kantenmarkierung $\{T_i\}$ erreichbar ist. Dies definiert eine eindeutige Zuordnung, für die insbesondere gilt, wenn $\sharp$ die in T geltende Unterscheidbarkeitsrelation bezeichnet:

(*) $\forall i,j\varepsilon\{1,\ldots,s\}.\ (v_i,v_j)\ \varepsilon\ E \iff K_i \sharp_1 K_j$

Der Beweis zu (*) erfolgt in zwei Schritten:

(i) Falls $(v_i,v_j)\ \varepsilon\ E$, so gibt es für diese Kante $e_r=(v_i,v_j)$ die zwei Beispielsfolgen M_r und M'_r in Q, die in IHG den Indizes r und t+r entsprechen. Index r führt vom START-Knoten aus zum Knoten K_i, Index t+r zu K_j, und wegen $T_i \neq T_j$ gilt $K_i \neq K_j$. K_i und K_j sind beide mit {A} markiert und daher nicht O-unterscheidbar. Der

über Kantenmarkierung $\{T_r'\}$ erreichbare Nachfolger von K_i gemäß Index r ist mit {STOP} markiert, der ebenfalls über $\{T_r'\}$ erreichbare Nachfolger von K_j gemäß Index t+r jedoch mit {A}. Folglich gilt $K_i \neq_1 K_j$.

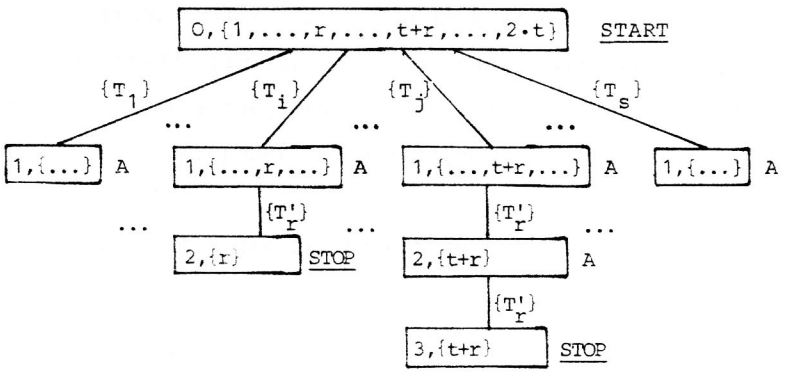


Abb. 6.1 IH-Situation IS

(ii) Falls $K_i \neq_1 K_j$, so muß es wegen $\neg(K_i \neq_0 K_j)$ Nachfolger K_i' und K_j' von K_i und K_j geben, die über dieselbe Kantenmarkierung erreichbar und die O-unterscheidbar sind. Die s Knoten erster Stufe in IHG haben insgesamt 2·t Nachfolger zweiter Stufe, wobei zwischen diesen beiden Stufen jede Kantenmarkierung aus $\{\{T_1'\},\ldots,\{T_t'\}\}$ genau zweimal vorkommt. Von K_i und K_j können aber nach Konstruktion von IHG nur dann gleiche Kantenmarkierungen ausgehen, wenn es eine Kante $(v_i,v_j) \in E$ gibt.

Daher gelten:

1. G sei k-färbbar; F : V —> {1,...,k} sei Einfärbung von G. Die LABEL-Hypothese u, die den Knoten K_i der Stufe eins von IS die Farbe des entsprechenden Knoten von G ($u(K_i) = F(v_i)$ und den mit A markierten Knoten K der zweiten Stufe von IS einen Wert aus {1,...,k}, der lediglich von $F(v_i)$ verschieden sein muß, wenn K Nachfolger von K_i ist, zuordnet, ist eindeutig und liefert einen U-Graph UG mit k Knoten. Nach Satz 4.12 ist Q k-realisierbar.

2. Q sei k-realisierbar; P sei ein Programm, das Q k-realisiert. Nach Satz 3.9 liefert P eine eindeutige LABEL-Hypothese u für IS. O.B.d.A. kann angenommen werden, daß in P nur LABEL aus {1,...,k} vorkommen (vgl. Anmerkung im Anschluß an Def. 4.6). Die durch $F = \lambda v.\ v = v_i \text{ —> } u(K_i)$ definierte Abbildung F : V —> {1,...,k} ist daher wegen (*) und Satz 4.5 eine k-Einfärbung von G.

Dies schließt den Beweis von Satz 6.7 ab.

6.4.2 NP-Vollständigkeit der Programmsynthese aus Beispielsfolgen

Wegen der polynomialen Verwandtschaft der Schwierigkeitsgrade von Realisierungs- und Programmsyntheseproblem folgt aus Satz 6.7 unmittelbar die NP-Vollständigkeit der letzteren Problemklasse.

Satz 6.8

$PSP^{fin}_{I,S}$ ist NP-vollständig.

Beweis: folgt aus den Sätzen 6.2, 6.4 und 6.7.

Ein wesentlicher Punkt der Beweisführung von Satz 6.7 liegt darin, daß die Anzahl der für die Beschreibung der einzelnen Speicherbelegungszustände zur Verfügung stehenden Zeichen und die Anzahl der einem PSP zugrunde liegenden Instruktionen nicht von vorneherein durch die Definition von $PSP^{fin}_{I,S}$ beschränkt ist. Es stellt sich die Frage, ob auch für derartige Einschränkungen von $PSP^{fin}_{I,S}$ Realisierungsproblem und Programmsynthese noch NP-vollständig sind.

6.5 NP-Vollständigkeit eingeschränkter Realisierungs- und Programmsyntheseprobleme

Es werden Einschränkungen bzgl. einzelner Größenkriterien, wie sie in Abschnitt 6.1.2 angegeben worden sind, sowie verschiedene Kombinationen solcher Einschränkungen betrachtet. Zur Vereinfachung der Notation wird folgende Definition eingeführt:

Definition 6.7

Sei $b \in \{s,a,ss,as\} \cup \{-\}$, $m,n,p,q,r \in \mathbb{N} \cup \{-\}$.

(1) L = PSP-E(b,m,n,p,q,r) ist die Menge der Programmsyntheseprobleme mit Einschränkung (b,m,n,p,q,r)

<=>

$L \subseteq PSP^{fin}_{I,S}$ und für alle $Q \in L$ gelten:

(i) falls $b \neq$ '-', dann ist die Problemspezifizierungsmarke von Q gleich b

(ii) falls $m \neq$ '-', dann enthält Q nicht mehr als m Beispielsfolgen

(iii) falls $n \neq$ '-', dann ist keine der Beispielsfolgen von Q länger als n

(iv) falls $p \neq$ '-', dann enthält die Q zugrunde liegende Zuweisungsmenge nicht mehr als p Instruktionen, d.h. |ASSIGNMENTS| $\leq$ p

(v) falls $q \neq$ '-', dann enthält die Q zugrunde liegende Menge von Tests nicht mehr als q Instruktionen, d.h. $|TESTS| \leq q$

(vi) falls $r \neq$ '-', dann enthält das Q zugrunde liegende Alphabet zur Beschreibung der Speicherbelegungen nicht mehr als r Zeichen, d.h. $|\Sigma| \leq r$

(2) Das Realisierungsproblem mit Einschränkung (b,m,n,p,q,r) ist das folgende: Gegeben ein PSP Q aus PSP-E(b,m,n,p,q,r) und ein $l \in \mathbb{N}$, ist Q l-realisierbar?

Wenn in den folgenden Abschnitten 6.5.1 - 6.5.5 die NP-Vollständigkeit des Realisierungsproblems mit verschiedenen Einschränkungen gezeigt wird, so folgt daraus immer sofort die NP-Vollständigkeit der entsprechend eingeschränkten Menge der PSP, denn es gilt:

Satz 6.9 Sei b,m,n,p,q,r wie in Def. 6.7

Behauptung:
Ist das Realisierungsproblem mit Einschränkung (b,m,n,p,q,r) NP-vollständig, so ist auch PSP-E(b,m,n,p,q,r) NP-vollständig.

Beweis: analog zu Satz 6.7 und Satz 6.8 .

Daher wird in den weiteren Sätzen dieses Abschnitts nicht mehr explizit das Programmsyntheseproblem, sondern nur noch das Realisierungsproblem berücksichtigt.

6.5.1 Einschränkung (-,-,3,1,-,-)

Das Realisierungsproblem eingeschränkt auf solche PSP Q, deren Beispielsberechnungen nicht länger als 3 sind und in denen nur eine einzige Zuweisungsinstruktion vorkommt, ist NP-vollständig. Dies folgt aus der Beweisführung zu Satz 6.7, in der zu einem beliebigen Einfärbungsproblem eine korrespondierende Instanz des Realisierungsproblems konstruiert wird, die genau diese Bedingungen erfüllt.

6.5.2 Einschränkung (-,-,-,4,3,4)

Die Einschränkung der Mengen ASSIGNMENTS, TESTS und Σ (vgl. Def. 2.1) ist insofern von größerem Interesse als die zuvor betrachtete, als sie wirklichkeitsnäher in folgendem Sinne ist: Eine vorgegebene Rechenmaschine oder ein bestimmtes Berechnungsmodell werden i.a. durch eine endliche und feste Anzahl von möglichen, ausführbaren Operationen bestimmt. Darüberhinaus stehen üblicherweise nur eine feste, endliche Anzahl von Unterscheidungskriterien zur Verfügung. Bei einer vorgegebenen Rechenmaschine R wäre also auf die hier benützte Terminologie übertragen nach dem ersten Punkt ASSIGNMENTS, nach dem zweiten Punkt TESTS und als Folgerung aus beiden auch Σ beschränkt auf endliche, fest vorgegebene Mengen. Betrachtet man nur PSP, die ein solches R erzeugen kann, so ist die daraus resultierende Einschränkung des

Realisierungsproblems sogar dann noch NP-vollständig, wenn man lediglich 4 Zuweisungs- und 3 Testinstruktionen und 4 Zeichen zur Beschreibung der Speicherbelegungen zuläßt.

Es ist zu beachten, daß diese Aussagen auch bei noch stärkerer Einschränkung bzgl. der angegebenen Parameter zutrifft (vgl. Abschnitt 6.5.5); die wesentliche Aussage des folgenden Satzes liegt jedoch darin, daß eine gleichzeitige Einschränkung dieser drei Parameter überhaupt möglich ist, ohne die NP-Vollständigkeit des Realisierungsproblems zu beseitigen.

Satz 6.10

Das Realisierungsproblem mit Einschränkung (-,-,-,4,3,4) ist NP-vollständig.

Beweis:

Sei L gegeben durch:

$L = \{(Q,l) \mid Q \in PSP\text{-}E(-,-,-,4,3,4) \wedge l \in \mathbb{N} \wedge Q \text{ ist } l\text{-realisierbar}\}$.

Nach Satz 6.3 ist L in NP-TIME. I.f. wird gezeigt, daß das Einfärbungsproblem für Graphen polynomial transformierbar ist zu L, woraus nach Satz 6.6 die Behauptung folgt.

Gegeben sei ein ungerichteter Graph G = (V,E), $V = \{v_1,\ldots,v_s\}$, $E = \{e_1,\ldots,e_t\}$, und ein $k \in \mathbb{N}$. Für G seien die im Beweis zu Satz 6.7 o.B.d.A. gemachten Annahmen 1.- 4. erfüllt. Es wird ein PSP Q konstruiert, das genau dann $((k+1)\cdot s^2 + s + k)$-realisierbar ist, wenn G k-färbbar ist.

Q = (b,M) ist ein PSP aus zuweisungsgetreuen Speicherbelegungsfolgen, also b = as. Die Q zugrunde liegenden Speicherbelegungen STORES sind die Wörter über dem Alphabet $\Sigma = \{a,c,d,z\}$. Es gibt vier Zuweisungsinstruktionen, die alle jeweils ein Zeichen aus Σ an das Ende der vorliegenden Speicherbelegung w anfügen und von denen drei das erste Zeichen von w löschen. Drei Testinstruktionen stehen zur Verfügung; sie liefern true, wenn das erste Zeichen von w gleich 'a','c' bzw. 'z' ist.
Im einzelnen gilt, wobei $x \in \{a,c,z\}$, $y \in \Sigma$ und $w' \in \Sigma^*$:

ASSIGNMENTS = {A,C,D,Z}

A = λw. w = yw' —> w'a

C = λw. w = yw' —> w'c

D = λw. w = yw' —> w'd

Z = λw. wz

TESTS = $\{T_a,T_b,T_z\}$

T_x = λw. w = yw' —> x = y

Ähnlich wie im Beweis zu Satz 6.7 werden für jede Kante e_r von G zwei Beispielsfolgen in M aufgenommen, die mit M_r und M_r' bezeichnet werden. Darüberhinaus enthält M noch drei mit M_A, M_C und M_Z bezeichnete Folgen sowie k Folgen $M_D^1,\ldots,M_D^k$.

Für die Angabe dieser Folgen ist jedoch noch eine Hilfsdefinition notwendig. Dabei wird jedem Paar $(i,j) \in \{1,\ldots,s\}^2$ eineindeutig eine Zahl $f(i,j) \in \{1,\ldots,s^2\}$ zugeordnet. Die Funktion f wird dazu benützt, jeder Kante $e_r = (v_i,v_j)$ von G zwei eindeutige Zahlen f(i,j) bzw. f(j,i) zuordnen zu können. f ist gegeben durch:

$f : \{1,\ldots,s\}^2 \longrightarrow \{1,\ldots,s^2\}$

$f = \lambda i.\ \lambda j.\ (i-1)\cdot s + j$

Offensichtlich gilt:

(i) $f(i,j) = f(i',j') \iff i = i' \wedge j = j'$

(ii) $f(i,j) \neq f(j,i) \iff i \neq j$

Zu jeder Zahl $l \in \{1,\ldots,s^2\}$ gibt es also höchstens eine Kante $e_r = (v_i,v_j)$ in G [aufgrund der getroffenen Vereinbarung ist dies eindeutig, da $i < j$ gefordert wird], sodaß $f(i,j) = l$ oder $f(j,i) = l$, wobei wegen $(v,v) \notin E$ niemals beide Gleichungen gleichzeitig erfüllt sein können.

Die Beispielsfolgen der Menge

$M = \{M_1,\ldots,M_t\} \cup \{M_1',\ldots,M_t'\} \cup \{M_D^1,\ldots,M_D^k\} \cup \{M_A,M_C,M_Z\}$

sind gegeben durch, wobei a^i für $\underbrace{a \ldots a}_{i\text{-mal}}$ steht:

1. $M_A = (a^{s+1}z,a^s za,\ldots,aza^s)$

   M_A ist eine Folge von s+1 Elementen, die alle Wörter der Länge s+2 sind. Die Gesamtlänge von M_A ist somit $O(s^2)$.

2. $M_C = (c^{k+1}z,c^k zc,\ldots,czc^k)$

   M_C ist eine Folge von k+1 Elementen der Länge k+2. Wegen $k \leq s$ ist die Gesamtlänge von M_C $O(s^2)$.

3. $M_Z = (z,z^2,\ldots,z^{s^2+1})$

   M_Z enthält s^2+1 Elemente, deren Länge maximal s^2+1 ist. Die Gesamtlänge von M_Z ist somit $O(s^4)$.

4. Für $i \in \{1,\ldots,k\}$ gilt:

   $M_D^i = (c^i a^{(k-i+1)s^2+1}z,c^{i-1}a^{(k-i+1)s^2+1}zc,\ldots,a^{(k-i+1)s^2+1}zc^i,$

   $a^{(k-i+1)s^2}zc^i d,\ldots,azc^i d^{(k-i+1)s^2})$

   M_D^i enthält $(k-i+1)s^2 + i + 1$ Wörter der Länge $(k-i+1)s^2 + i + 2$, sodaß wegen $k = O(s)$ die Gesamtlänge von M_D^i $O(s^6)$ ist. Weiterhin ist daher die Summe der Längen von $M_D^1,\ldots,M_D^k$ gegeben durch $O(s^7)$.

5. Für jede Kante e_r von G gibt es genau zwei Beispielsfolgen in Q, nämlich M_r und M_r'. Sei $r \in \{1,\ldots,t\}$, $e_r = (v_i,v_j)$, wobei wieder $i < j$. Dann gilt:

   $M_r = (a^i ca^{f(i,j)}z,a^{i-1}ca^{f(i,j)}za,\ldots,ca^{f(i,j)}za^i,$

   $a^{f(i,j)}za^i c,\ldots,za^i ca^{f(i,j)},za^i ca^{f(i,j)}z,\ldots$

   $\ldots,za^i ca^{f(i,j)}z^{f(i,j)})$

   $M_r' = (a^j ca^{f(i,j)}z,a^{j-1}ca^{f(i,j)}za,\ldots,ca^{f(i,j)}za^j,$

   $a^{f(i,j)}za^j c,\ldots,za^j ca^{f(i,j)},za^j ca^{f(i,j)}z,\ldots$

   $\ldots,za^j ca^{f(i,j)}z^{f(i,j)})$

   Da $i,j \in \{1,\ldots,s\}$, enthält M_r Wörter maximal der Länge $2\cdot f(i,j) + j + 2 = O(s^2)$; ebenso ist die Anzahl der Wörter in M_r $O(s^2)$, die Gesamtlänge von M_r und damit auch von M_r' also $O(s^4)$. Da die Anzahl der Kanten t nicht größer als s^2 sein kann, gibt es $O(s^2)$ derartige Folgen M_r sowie M_r'; die Summe der Längen aller dieser ist daher $2\cdot O(s^2)\cdot O(s^4) = O(s^6)$.

Damit ist Q vollständig spezifiziert. Die gesamte Länge der Standardcodierung von Q ergibt sich aus der Summe der unter 1. - 5. gegebenen Längen zu $O(s^7)$, während auch die für die Konstruktion von Q benötigte Zeit sicherlich nicht mehr als polynomial mit s anwächst. Für den Nachweis, daß es sich dabei um eine polynomiale Transformation im Sinne von Def. 6.5 handelt, bleibt also noch zu zeigen: G ist k-färbbar genau dann, wenn Q $((k+1)\cdot s^2 + s + k)$-realisierbar ist.

Der Beweis hierzu erfolgt wie in Satz 6.7 mit Hilfe der Standarddarstellung Q_{ST} und des IH-Graphen IHG von Q. Die Berechnung von $Q_{ST} = (\underline{as},S,MI,MT)$ ist einfach, da jeder Speicherbelegungsübergang in M durch genau eine Zuweisungsinstruktion aus ASSIGNMENTS erzeugt werden kann und für jede in M auftretende Speicherbelegung genau eine Testinstruktion aus TESTS true liefert. Um eine einfache Korrespondenz zwischen den Folgen von M und den Zeilen aus S, MI und MT zu erhalten, wird auf eine fortlaufende Indizierung der Zeilen verzichtet und stattdessen die zuvor benutzte Bezeichnung übernommen, sodaß S_A, MI_A und MT_A der Beispielsfolge M_A entsprechen usw.

(1) S enthält genau die Folgen aus M als Zeilen.

(2) MI und MT erhalten als Zeilen

(2.1) für M_A :

$$MI_A = (\{\underline{START}\},\underbrace{\{A\},\ldots,\{A\}}_{s\text{-mal}},\{\underline{STOP}\})$$

$$MT_A = (\underbrace{\{T_a\},\ldots,\{T_a\}}_{(s+1)\text{-mal}})$$

(2.2) für M_C :

$$MI_C = (\{\underline{START}\},\underbrace{\{C\},\ldots,\{C\}}_{k\text{-mal}},\{\underline{STOP}\})$$

$$MT_C = (\underbrace{\{T_c\},\ldots,\{T_c\}}_{(k+1)\text{-mal}})$$

(2.3) für M_Z :

$$MI_Z = (\{\underline{START}\},\underbrace{\{Z\},\ldots,\{Z\}}_{s^2\text{-mal}},\{\underline{STOP}\})$$

$$MT_Z = (\underbrace{\{T_z\},\ldots,\{T_z\}}_{(s^2+1)\text{-mal}})$$

(2.4) für M_D^i, wobei $i\in\{1,\ldots,k\}$:

$$MI_D^i = (\{\underline{START}\},\underbrace{\{C\},\ldots,\{C\}}_{i\text{-mal}},\underbrace{\{D\},\ldots,\{D\}}_{(k-i+1)\cdot s^2\text{-mal}},\{\underline{STOP}\})$$

$$MT_D^i = (\underbrace{\{T_c\},\ldots,\{T_c\}}_{i\text{-mal}},\underbrace{\{T_a\},\ldots\ldots\ldots,\{T_a\}}_{((k-i+1)\cdot s^2+1)\text{-mal}})$$

(2.5) für M_r und M_r', wobei $r\in\{1,\ldots,t\}$ und $e_r = (v_i,v_j)$:

$$MI_r = (\{\underline{START}\},\underbrace{\{A\},\ldots,\{A\}}_{i\text{-mal}},\{C\},\underbrace{\{D\},\ldots,\{D\}}_{f(i,j)\text{-mal}},\underbrace{\{Z\},\ldots,\{Z\}}_{f(i,j)\text{-mal}},\{\underline{STOP}\})$$

$$MT_r = (\underbrace{\{T_a\},\ldots,\{T_a\}}_{i\text{-mal}},\{T_c\},\underbrace{\{T_a\},\ldots,\{T_a\}}_{f(i,j)\text{-mal}},\underbrace{\{T_z\},\ldots\ldots,\{T_z\}}_{(f(i,j)+1)\text{-mal}})$$

$$MI_r' = (\{\underline{START}\},\underbrace{\{A\},\ldots,\{A\}}_{j\text{-mal}},\{C\},\underbrace{\{D\},\ldots,\{D\}}_{f(i,j)\text{-mal}},\underbrace{\{Z\},\ldots,\{Z\}}_{f(j,i)\text{-mal}},\{\underline{STOP}\})$$

$$MT_r' = (\underbrace{\{T_a\},\ldots,\{T_a\}}_{j\text{-mal}},\{T_c\},\underbrace{\{T_a\},\ldots,\{T_a\}}_{f(i,j)\text{-mal}},\underbrace{\{T_z\},\ldots,\{T_z\}}_{(f(j,i)+1)\text{-mal}})$$

Alle Knoten des aus Q_{ST} zu konstruierenden IH-Graphen IHG von Q sind mit einelementigen Instruktionsmengen markiert; folglich gibt es genau eine einzige aus IHG ableitbare terminale IH-Situation IS. Zur besseren Übersicht sind in dem in Abb. 6.2 skizzierten Teil von IS zunächst nur die Knoten berücksichtigt, die den vier Beispielsfolgen M_A, M_C, M_D^1 und M_Z entsprechen. Über die zwischen diesen Knoten geltenden Unterscheidbarkeitsrelationen lassen sich folgende Aussagen machen:

Alle mit A,C,D und Z markierten Knoten in Abb. 6.2 sind paarweise unterscheidbar. Mit Hilfe der Sätze 4.5 und 4.6 folgt, daß jede eindeutige LABEL-Hypothese u zu einem U-Graph UG mit mindestens s A-Knoten, k C-Knoten, $k \cdot s^2$ D-Knoten und s^2 Z-Knoten führt, daß UG also mindestens $(k+1) \cdot s^2 + s + k$ Zuweisungsknoten enthält.

Berücksichtigt man noch zusätzlich die Beispielsfolgen $M_D^2, \ldots, M_D^k$ in IS, so geht von jedem mit C markierten Knoten K von IS ein mit $\{T_a\}$ markierter Übergang zu einer Folge von $(k-i+1) \cdot s^2$ mit D markierten Knoten, wenn LEVEL(K) gleich i ist. Dieser Teil von IS ist in Abb. 6.3 skizziert. Alle Knoten jeder dieser D-Folgen in Abb. 6.3 sind jeweils paarweise unterscheidbar, und es gibt genau eine einzige Möglichkeit, eine eindeutige LABEL-Hypothese u für den in Abb. 6.2 skizzierten Teil von IS auf die in Abb. 6.3 angegebenen Knoten zu erweitern, ohne zusätzliche LABEL zu benützen. O.B.d.A. kann dabei angenommen werden, daß u den s A-Knoten fortlaufend die LABEL 1,...,s , den k C-Knoten die LABEL 1,...,k usw.

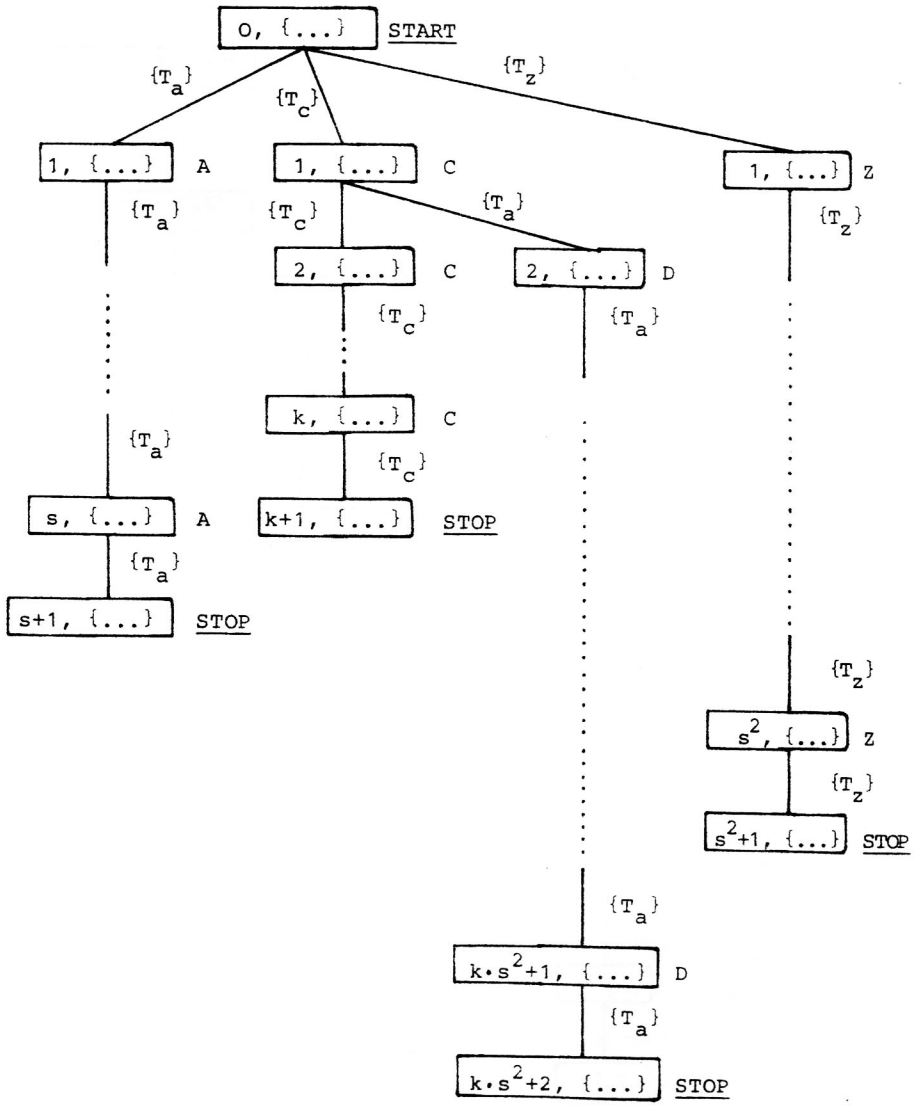


Abb. 6.2 Der Teil von IS für M_A, M_C, M_D^1 und M_Z

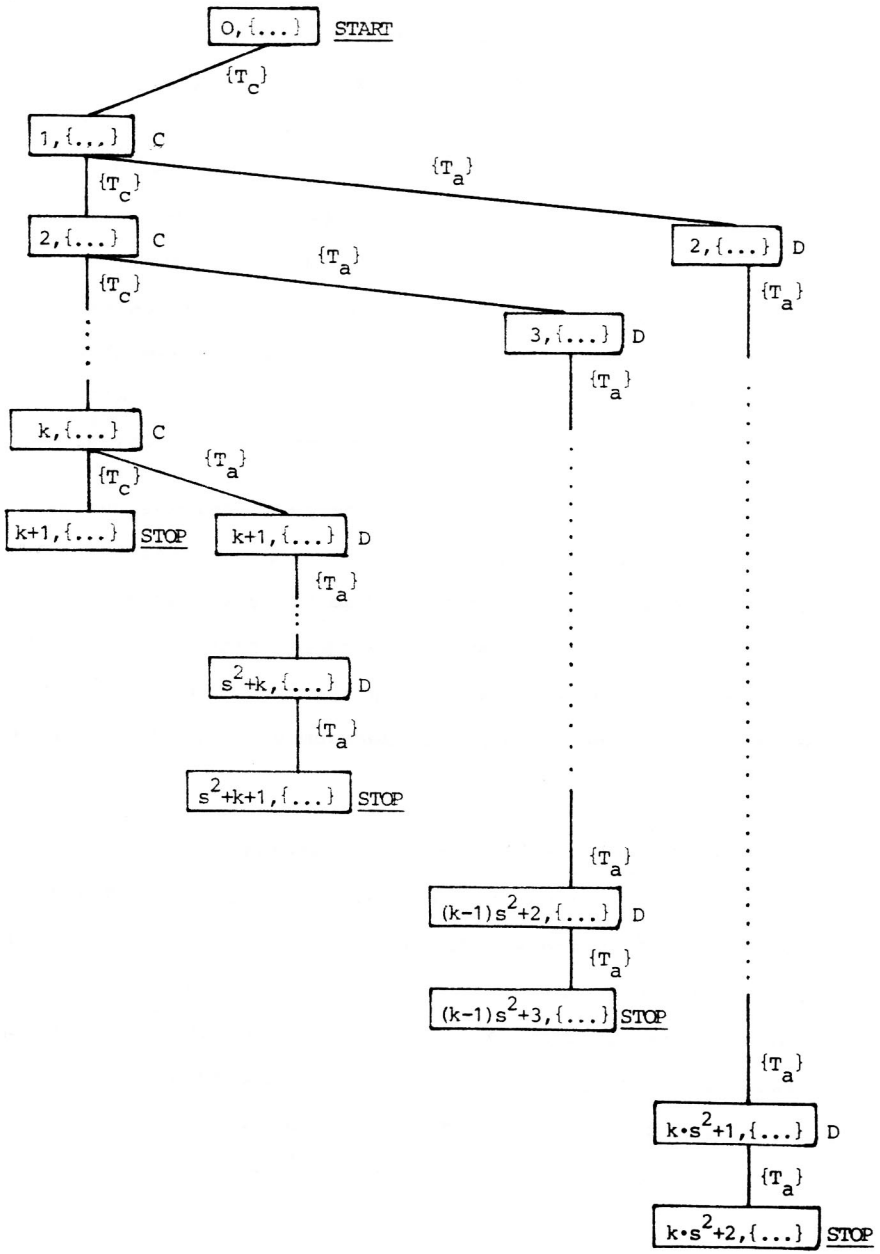


Abb. 6.3 Der Teil von IS für $M_D^1,\ldots,M_D^k$

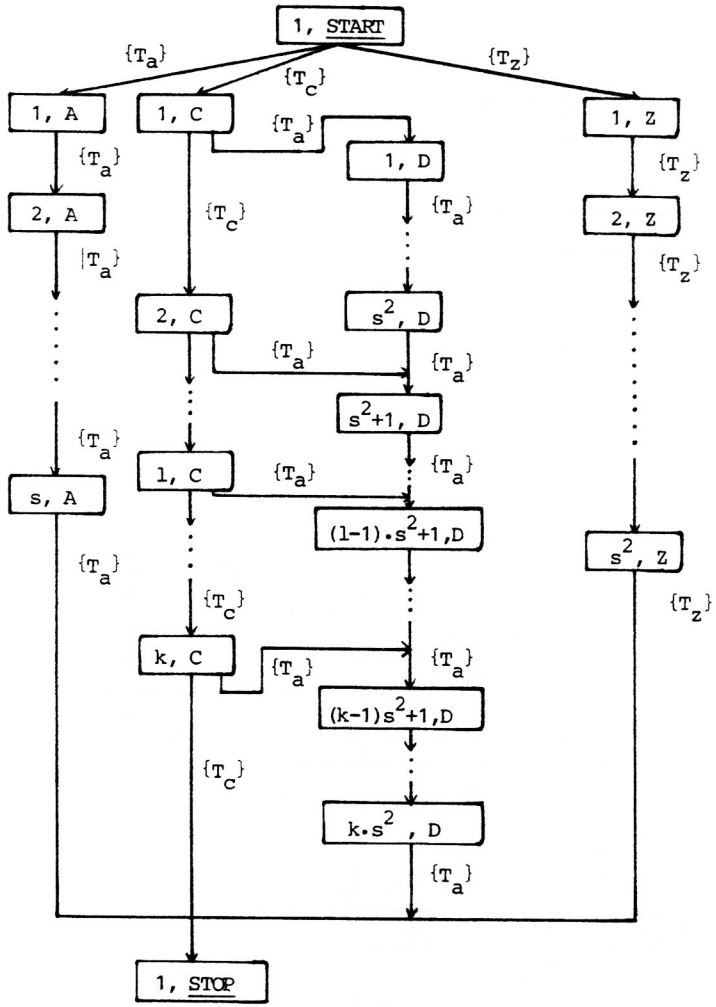


Abb. 6.4 U-Graph UG für M_A, M_C, $M_D^1,\ldots,M_D^k,M_Z$

zuordnet. Der entstehende U-Graph UG ist in Abb. 6.4 angegeben.

Wesentlich ist, daß jede eindeutige LABEL-Hypothese u, die die den Folgen $M_A, M_C, M_Z, M_D^1, \ldots, M_D^k$ entsprechenden Knoten von IS berücksichtigt und nicht mehr als $(k+1)\cdot s^2 + s + k$ Knoten benötigt, zu dem in Abb. 6.4 skizzierten UG führt, wobei höchstens noch eine unwesentliche Umbenennung der LABEL möglich wäre.

Für diese k+3 Beispielsfolgen werden also schon $(k+1)\cdot s^2 + s + k$ Zuweisungsknoten benötigt. Die Erweiterung von u zu einer Markierung für die gesamte IH-Situation IS, ohne neue Programmknoten zu erzeugen, kann daher nur durch zusätzlich in UG eingefügte Übergänge zwischen bereits vorhandenen Knoten geschehen. Aber auch dabei sind die Auswahlmöglichkeiten beschränkt, wie aus den folgenden Ausführungen deutlich wird.

Sei $r \in \{1,\ldots,t\}$, $e_r = (v_i, v_j)$. Die M_r und M_r' entsprechenden Instruktionsfolgen seien I_r und I_r'. Für

$$I_r = (\underline{\text{START}}, \underbrace{A,\ldots,A}_{i\text{-mal}}, C, \underbrace{D,\ldots,D}_{f(i,j)\text{-mal}}, \underbrace{Z,\ldots,Z}_{f(i,j)\text{-mal}}, \underline{\text{STOP}})$$

kommen wegen der Übergänge

$$MT_r = (\underbrace{\{T_a\},\ldots,\{T_a\}}_{i\text{-mal}}, \{T_c\}, \underbrace{\{T_a\},\ldots,\{T_a\}}_{f(i,j)\text{-mal}}, \underbrace{\{T_z\},\ldots,\{T_z\}}_{(f(i,j)+1)\text{-mal}})$$

und den bereits in UG vorhandenen Tripeln als LABEL in Frage:

(i) Für die i A-Knoten von I_r müssen die LABEL 1,...,i verwendet werden, da in MT_r die Übergänge $\{T_a\}$ identisch sind mit den bereits in UG vorhandenen.

(ii) Vom Knoten (i,A) erfolgt ein $\{T_c\}$-Übergang zu einem der k C-Knoten in UG; das LABEL dieses C-Knotens werde in Abhängigkeit von i mit u_i bezeichnet. Der U-Graph enthält demnach das Tripel $((i,A),\{T_c\},(u_i,C))$.

(iii) Von (u_i,C) erfolgt in I_r ein $\{T_a\}$-Übergang zu einem D-Knoten. Ein solcher Übergang ist aber in UG schon vorhanden, nämlich $((u_i,C),\{T_a\},((u_i-1)\cdot s^2+1,D))$, und wegen der Eindeutigkeitsanforderung an UG muß auch für die erste D-Instruktion in I_r als LABEL $(u_i-1)\cdot s^2 + 1$ gewählt werden.

(iv) Für die restlichen f(i,j)-1 D-Instruktionen kommen aus dem gleichen Grund nur die LABEL $(u_i-1)\cdot s^2 + 2, \ldots, (u_i-1)\cdot s^2 + f(i,j)$ in Frage.

(v) Vom Knoten $((u_i-1)\cdot s^2 + f(i,j),D)$ erfolgt in I_r ein T_z-Übergang zu einem Z-Knoten (1,Z), der den ersten einer Folge von genau f(i,j) Z-Knoten gefolgt vom STOP-Knoten darstellt. Da alle dazwischenliegenden Übergänge in MT_r ebenso wie die entsprechenden in UG mit $\{T_z\}$ versehen sind, kommt wieder wegen der Eindeutigkeitsforderung für (1,Z) nur genau der Knoten aus UG in Frage, der f(i,j) Z-Knoten vom STOP-Knoten 'entfernt' ist; 1 muß daher gleich s^2-f(i,j)+1 sein.

Der U-Graph enthält demnach das Tripel

$(((u_i-1)\cdot s^2 + f(i,j),D),\{T_z\},\ (s^2-f(i,j)+1,Z))$.

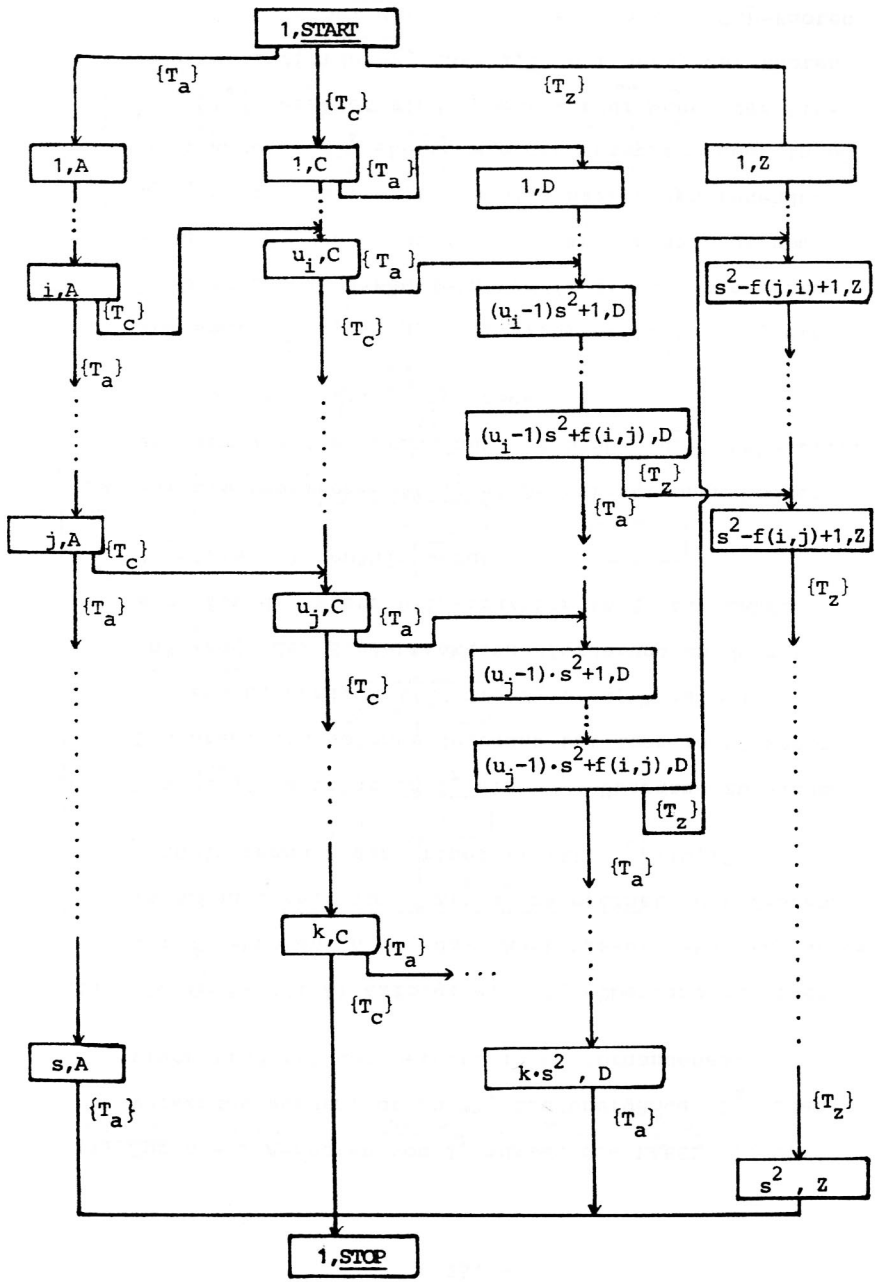
Für I_r besteht daher nur an einem einzigen Punkt Wahlfreiheit bzgl. der LABEL von UG, nämlich bei der Auswahl des C-Knotens, der der C-Instruktion von I_r entspricht. Diese LABEL wurde unter (ii) in Abhängigkeit von i bei $e_r = (v_i,v_j)$ mit u_i bezeichnet. Eine weitere Überlegung zeigt, daß u_i für alle $e = (v_i,v)$ gleich sein muß, da die entsprechenden Instruktionsfolgen alle mit einer Folge von genau i A-Instruktionen beginnen und der anschließende $\{T_c\}$-Übergang von (i,A) zu dem Knoten (u_i,C) eindeutig sein muß.

Die obigen Ausführungen gelten prinzipiell auch für I_r'. Bei $e_r = (v_i,v_j)$ beginnt I_r' jedoch mit j A-Instruktionen, und im obigen Schritt (v) sind nicht f(i,j) Z-Knoten zu durchlaufen, sondern f(j,i), wobei immer $f(i,j) \neq f(j,i)$ gilt. Bezeichnet man analog zu (ii) mit u_j das LABEL des C-Knotens von UG, der der C-Instruktion aus I_r' entspricht - der U-Graph enthält demnach das Tripel $((j,A),\{T_c\},(u_j,C))$ -, so ist diese Definition mit dem Obigen konsistent, und man erhält eine eindeutige und, da G verbunden ist und daher jeder Knoten von G in mindestens einer Kante auftritt, vollständig definierte Abbildung

$$F_u : V \longrightarrow \{1,\ldots,k\}$$

$$F_u : \lambda v.\ v = v_i \longrightarrow u_i$$

UG' bezeichne den durch die Schritte (i) - (v) erweiterten U-Graphen UG. UG' ist in Abb. 6.5 skizziert.



Es bleibt die Eindeutigkeit von UG' im Sinne von Def. 3.13 zu überprüfen. UG (Abb. 6.4) ist offensichtlich eindeutig. Lediglich in den Schritten (ii) und (v) werden neue Übergänge eingefügt. Für jeden A-Knoten (i,A), $i \in \{1,\ldots,s\}$, wird dabei in (ii) ein eindeutiger Übergang nach (u_i,C) erzeugt. Unabhängig von der Wahl der u_i sind also alle von START-,A-,C- und Z-Knoten ausgehenden Übergänge in UG' eindeutig; abhängig von den u_i und damit der Funktion F_u ist jedoch die Eindeutigkeit der in Schritt (v) eingefügten Übergänge von den D- zu den Z-Knoten. Es gilt:

Die von den D-Knoten ausgehenden Übergänge in UG' sind genau dann eindeutig, wenn F_u Einfärbung von G ist.

(1) F_u sei eine Einfärbung von G.
Widerspruchsannahme: Es gibt einen D-Knoten mit LABEL l, dessen Nachfolger nicht eindeutig durch die zu ihnen führenden Übergangsmarkierungen spezifizierbar sind. Letzteres kann nur dann der Fall sein, wenn es von (l,D) aus zwei $\{T_z\}$-Übergänge zu verschiedenen Z-Knoten gibt. Von (l,D) aus kann aber nur in Schritt (v) der Erweiterungen von UG ein $\{T_z\}$-Übergang eingefügt worden sein. Danach müßte es in E zwei Kanten (v_i,v_j) und $(v_{i'},v_{j'})$ geben, sodaß $l = r \cdot s^2 + f(i,j)$ und $l = r' \cdot s^2 + f(i,j)$, wobei $r = u_i - 1$ oder $r = u_j - 1$ und $r' = u_{i'} - 1$ oder $r' = u_{j'} - 1$ gelten muß. Aus der Definition von f folgt, daß $f(i,j)$ immer kleiner als s^2 ist; die Zerlegung von l in $r \cdot s^2 + f(i,j)$ ist demnach eindeutig und es muß $f(i,j) = f(i',j')$ gelten.

Aus $f(i,j) = f(i',j')$ folgt ebenfalls nach Definition von f $i = i'$ und $j = j'$ und damit $(v_i,v_j) = (v_{i'},v_{j'})$. Das Einfügen der $\{T_z\}$-Übergänge von (l,D) aus kann also nur bzgl. der beiden Beispielsfolgen M_r und M'_r erfolgt sein, wenn $e_r = (v_i,v_j)$. Daher muß u_i die Gleichung $l=(u_i-1)\cdot s^2+f(i,j)$ und u_j die Gleichung $l=(u_j-1)\cdot s^2+f(i,j)$ erfüllen. Diese Zerlegung von l ist jedoch,wie gesagt, wegen $f(i,j) < s^2$ eindeutig, und es folgt $u_i = u_j$ und damit $F_u(v_i) = F_u(v_j)$ im Widerspruch dazu, daß F_u Einfärbung von G ist.

(2) F_u sei keine Einfärbung von G.
Dann gibt es eine Kante $e_r = (v_i,v_j)$ in E, sodaß $F_u(v_i) = F_u(v_j)$ und somit $u_i = u_j$. In Schritt (ii) der Erweiterung von UG erfolgt daher sowohl für I_r als auch für I'_r ein Übergang nach (l,C), wobei $l = u_i = u_j$, gefolgt von einem Übergang nach $(l-1)\cdot s^2 + 1,D)$. Anschließend werden in beiden Fällen genau f(i,j) D-Knoten durchlaufen. Von $((l-1)\cdot s^2 + f(i,j),D)$ gibt es dann gemäß Schritt (v) zwei $\{T_z\}$-Übergänge in UG':

$$(((l-1)\cdot s^2+f(i,j),D),\{T_z\},(s^2-f(i,j)+1,Z)) \in UG'$$

$$(((l-1)\cdot s^2+f(i,j),D),\{T_z\},(s^2-f(j,i)+1,Z)) \in UG'$$

Wären die von diesem K-Knoten ausgehenden Übergänge eindeutig, so müßte $f(i,j) = f(j,i)$ gelten. Dies ist jedoch wegen $i \neq j$ nach Definition von f nicht möglich.

Da bei der Spezifizierung der LABEL-Hypothesen für IS keine die Allgemeinheit beschränkenden Annahmen gemacht wurden, folgt aus dem vorigen, daß es genau dann eine eindeutige

LABEL-Hypothese für IS gibt, die zu einem U-Graph mit nicht mehr als $(k+1)\cdot s^2 + s + k$ Knoten führt, wenn G k-färbbar ist. Da IS einzige aus IH-GRAPH(Q) ableitbare terminale IH-Situation ist, ist Q nach Satz 3.9 und Satz 4.12 genau dann $((k+1)\cdot s^2 + s + k)$-realisierbar, wenn G k-färbbar ist.

Damit ist der Beweis von Satz 6.10 erfolgreich abgeschlossen.

6.5.3 Einschränkung (-,1,-,5,3,4)

Schränkt man nicht nur ASSIGNMENTS, TESTS und Σ ein, sondern darüberhinaus noch die Anzahl der Beispielsfolgen eines PSP Q, so ist das entsprechende Realisierungsproblem immer noch NP-vollständig, und zwar sogar auch dann, wenn man nur eine einzige Beispielsberechnung in Q zuläßt.

Satz 6.11

Das Realisierungsproblem mit Einschränkung (-,1,-,5,3,4) ist NP-vollständig.

Beweis:

Sei L gegeben durch:

$L = \{(Q,1) \mid Q \in PSP\text{-}E(-,1,-,5,3,4) \wedge 1 \in \mathbb{N} \wedge Q \text{ ist 1-realisierbar}\}$.

Nach Satz 6.3 ist L in NP-TIME. Analog zum Beweis von Satz 6.10 wird gezeigt, daß das Einfärbungsproblem für Graphen

polynomial transformierbar ist zu L, woraus nach Satz 6.6 die Behauptung folgt.

Die Beweisidee beruht darauf, die zu einem Einfärbungsproblem (k,G) in Satz 6.10 konstruierten $2\cdot t + k + 3$ Beispielsfolgen zu konkatenieren, sodaß man nur eine einzige Beispielsfolge erhält. Am einfachsten wird dies erreicht durch eine zusätzliche Zuweisungsinstruktion H, die an die Stelle von STOP als Trennmarkierung zwischen zwei Instruktionsfolgen tritt.

Seien k,G,f,Σ,A,C,D,Z,TESTS wie in Satz 6.10;
ASSIGNMENTS = {A,C,D,Z,H}, wobei für $y \in \Sigma$ und $w' \in \Sigma^*$ H gegeben ist durch
$H = \lambda w.\ w = yw' \longrightarrow w'zz$.
Es wird ein PSP Q = (b,M) konstruiert, das genau dann $((k+1)\cdot s^2 + s + k + 1)$-realisierbar ist, wenn G k-färbbar ist. Dies entspricht genau Satz 6.10, wobei der hier noch zusätzlich benötigte Programmknoten mit der neuen Instruktion H markiert ist.

Der Einfachheit halber wird Q als ein PSP aus zuweisungsgetreuen Instruktionsfolgen angegeben, also b = a.
M = {(st,SEQ)} enthält nur eine einzige Beispielsberechnung. Da jedoch die sich aus der Anfangsbelegung st durch sukzessive Anwendung der Instruktionen von SEQ ergebenden Zwischenwerte nur jeweils durch genau die angewandte Instruktion erzeugt werden können, könnte auch ein PSP Q' = (as,M') angegeben werden, das als zuweisungsge-

treue Speicherbelegungsfolge die Anfangsbelegung st gefolgt von diesen Zwischenwerten erhält.
IH-GRAPH(Q) und IH-GRAPH(Q') wären bis auf die Problemspezifizierungsmarke a bzw. as identisch, und die folgenden Aussagen träfen auch für Q' zu.

Da die Instruktionsfolge SEQ im Prinzip eine Konkatenation der zu $M_A, M_C, M_Z, M_D^1, \ldots, M_D^k, M_1, M_1', \ldots, M_t, M_t'$ zugehörigen Instruktionsfolgen ist, werden zunächst die diesen Folgen entsprechenden Teilstücke von SEQ definiert. $\langle m_A \rangle$ entspricht M_A usw.; A^n steht für $\underbrace{A,\ldots,A}_{n\text{-mal}}$.

$$\langle m_A \rangle := A^s$$
$$\langle m_C \rangle := C^k$$
$$\langle m_Z \rangle := Z^{s^2}$$

für $i \varepsilon \{1,\ldots,k\}$:

$$\langle m_D^i \rangle := C^i, D^{(k-i+1)s^2}$$

für $r \varepsilon \{1,\ldots,t\}$: sei $e_r = (v_i, v_j)$

$$\langle m_r \rangle := A^i, C, D^{f(i,j)}, Z^{f(i,j)}$$
$$\langle m_r' \rangle := A^j, C, D^{f(i,j)}, Z^{f(j,i)}$$

Damit ist SEQ gegeben durch:

SEQ = (START, H, $\langle m_A \rangle$, H, $\langle m_C \rangle$, H, $\langle m_Z \rangle$, H, $\langle m_D^1 \rangle$, H, ...
..., $\langle m_D^k \rangle$, H, $\langle m_1 \rangle$, H, $\langle m_1' \rangle$, H, ..., $\langle m_t \rangle$, H,
$\langle m_t' \rangle$, H, STOP)

Die Länge von SEQ kann wie in Satz 6.10 bestimmt werden und errechnet sich zu $O(s^4)$.

Damit die in Satz 6.10 benützte Beweisführung auch hier angewandt werden kann, müssen die für die Zwischenwerte true-liefernden Instruktionen gleich den korrespondierenden Werten dort sein. Dies geschieht durch entsprechende Wahl der Anfangsspeicherbelegung st, die im Prinzip wieder eine Konkatenation der Anfangswerte von M_A, M_C usw. darstellt. Es steht a^n für $\underbrace{a \ldots a}_{n\text{-mal}}$.

$$\langle s_A \rangle := a^{s+1}$$
$$\langle s_C \rangle := c^{k+1}$$
$$\langle s_Z \rangle := z$$

für $i \varepsilon \{1,\ldots,k\}$:

$$\langle s_D^i \rangle := c^i a^{(k-i+1)s^2+1}$$

für $r \varepsilon \{1,\ldots,t\}$: sei $e_r = (v_i, v_j)$

$$\langle s_r \rangle := a^i c a^{f(i,j)} z$$
$$\langle s_r' \rangle := a^j c a^{f(i,j)} z$$

Damit ist st gegeben durch:

$$st = a\langle s_A \rangle\langle s_C \rangle\langle s_Z \rangle\langle s_D^1 \rangle \ldots \langle s_D^k \rangle\langle s_1 \rangle\langle s_1' \rangle \ldots \langle s_t \rangle\langle s_t' \rangle d$$

Das erste Zeichen von st wird wegen der zusätzlichen H-Instruktion benötigt; das Zeichen 'd' am Ende von st sorgt dafür, daß von dem H-Knoten ein mit Ø markierter Übergang zum STOP-Knoten möglich wird, da bei d keine der

drei Testinstruktionen T_a, T_c, T_z true liefert. Die Länge von st ist $O(s^4)$; die Länge der Standardcodierung des Realisierungsproblems $(Q, (k+1)\cdot s^2 + s + k + 1)$ somit ebenfalls $O(s^4) + O(s^4) = O(s^4)$.

(Bemerkung: Die Länge der Standardcodierung des anfangs erwähnten äquivalenten PSP Q' wäre mit $O(s^4)\cdot O(s^4) = O(s^8)$ zwar größer, es läge aber auch dabei eine polynomiale Transformation von G vor.)

IHG = IH-GRAPH(Q) kann wie in Satz 6.10 bestimmt werden. IHG hat auf jeder Stufe genau einen Knoten, jeweils mit einer einelementigen Instruktionenmenge markiert. Es gibt eine einzige aus IH ableitbare terminale IH-Situation IS; aus den in IS geltenden Unterscheidbarkeitsrelationen lassen sich ebenso wie in Satz 6.10 Aussagen über die Mindestanzahl von benötigten A-Knoten usw. ableiten. Es werden demnach mindestens s A-, k C-, s^2 Z-, $k\cdot s^2$ D- und 1 H-Knoten benötigt. Unter der Randbedingung, nicht mehr als $(k+1)\cdot s^2 + s + k + 1$ Knoten zu erzeugen, lassen sich weiterhin Aussagen darüber gewinnen, welchen Knoten von IS gleiche LABEL zugewiesen werden müssen. Wie in Satz 6.10 kann gefolgert werden, daß es genau dann eine eindeutige LABEL-Hypothese u gibt, die nicht mehr als $(k+1)\cdot s^2 + s + k + 1$ Knoten benötigt, wenn G k-färbbar ist. Der entsprechende U-Graph UG unterscheidet sich von UG' (Abb. 6.5) durch einen als direkten Nachfolger des START-Knotens eingefügten H-Knoten, zu dem die Übergänge des jeweils 'letzten' A-,C-,D- und Z-Knotens führen und von dem ein Übergang zum STOP-Knoten ausgeht. UG ist in Abb. 6.6 skizziert.

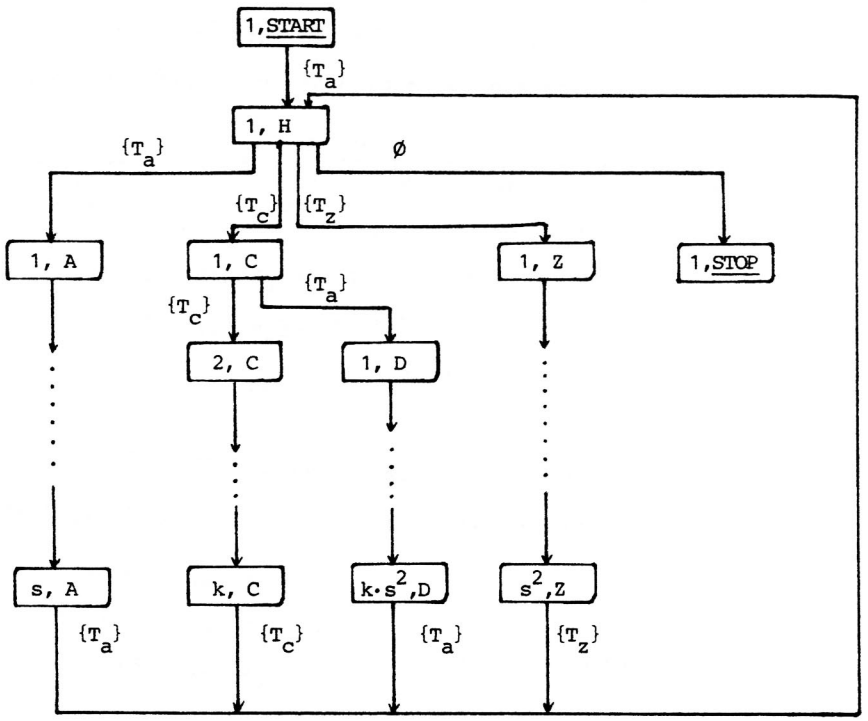


Abb. 6.6 U-Graph UG (zu Satz 6.11)

Die noch zu zeigenden einzelnen Beweisschritte verlaufen analog zu denen von Satz 6.10 .

6.5.4 Einschränkung (b,-,-,-,-,-)

Bisher wurden mit Ausnahme von Satz 6.11 zum Nachweis der NP-Vollständigkeit von Realisierungs- bzw. Programmsyntheseproblem immer nur PSP aus zuweisungsgetreuen Speicherbelegungsfolgen berücksichtigt. Beschränkt man sich jedoch auf irgendeine der drei anderen Arten von Programmsynthesepro-blemen (vgl. Def. 2.13), so gelten die getroffenen Aussagen ganz analog.

6.5.4.1 Programmsynthese aus Speicherbelegungsfolgen

Das Realisierungsproblem eingeschränkt auf zuweisungsgetreue Speicherbelegungsfolgen (also b = as) ist nach der Beweisführung von z.B. Satz 6.7 oder Satz 6.10 NP-vollständig. Aus dem weiter unten in 6.5.5 angegebenen Satz 6.12 folgt ein entsprechendes Resultat für statementgetreue Speicherbelegungsfolgen, also für b = ss. Das Problem der Programmsynthese aus derartigen Beispielsberechnungen ist demnach auch dann noch NP-vollständig, wenn die Speicherbelegungen nach jeder ausgeführten Zuweisung und jedem ausgeführten Test zur Verfügung stehen.

6.5.4.2 Programmsynthese aus Instruktionsfolgen

Liegen für die Programmsynthese nicht die beobachteten Speicherbelegungen, sondern direkt die ausgeführten Instruktionen vor, so ist trotz dieser Vereinfachung die Problemstellung noch NP-vollständig. Für den Fall von zuweisungsgetreuen Instruktionsfolgen (also b = a) folgt dies aus der Beweisführung von Satz 6.11. Aber selbst dann, wenn in den Beispielsfolgen neben allen ausgeführten Instruktionen auch alle durchlaufenen Tests mit angegeben sind (also b = s bei statement-getreuen Instruktionsfolgen), bleibt der Schwierigkeitsgrad der NP-Vollständigkeit erhalten. Zum Beweis dieser letzten Aussage sei auf Satz 6.12 im folgenden Abschnitt verwiesen.

6.5.5 Einschränkung (b,1,-,5,1,2)

Satz 6.12 stellt eine Zusammenfassung und Verschärfung der bisher getroffenen Aussagen zur Komplexität der Programmsynthese dar. Für jede einzelne der vier betrachteten Arten von Beispielsfolgen ist die Problemstellung sogar dann noch NP-vollständig, wenn man sich auf PSP mit nur einer einzigen Beispielsberechnung, fünf zugrunde liegenden Zuweisungs- und einer einzigen Testinstruktion sowie zwei Zeichen zur Beschreibung der Speicherbelegungen beschränkt.

Satz 6.12 Sei $b \in \{s,a,ss,as\}$.

Behauptung:

Das Realisierungsproblem mit Einschränkung (b,1,-,5,1,2) ist NP-vollständig.

Beweis:

Die Beweisidee beruht auf dem schon in den Sätzen 6.10 und 6.11 benützten Verfahren. Es wird gezeigt, daß das Einfärbungsproblem für Graphen polynomial transformierbar ist zu

$$L = \{(Q,1) \mid Q \in PSP\text{-}E(b,1,-,5,1,2) \wedge 1 \in \mathbb{N} \wedge Q \text{ ist 1-realisierbar}\},$$

woraus die Behauptung nach Satz 6.6 folgt.

Sei $k,G = (V,E), V = \{v_1,\ldots,v_s\}$, $E = \{e_1,\ldots, e_t\}$ wie in Satz 6.10. Da für $b \in \{a,as\}$ bereits mit Satz 6.11 eine ähnliche wie die hier zu zeigende Aussage bewiesen wurde (wenn auch mit nicht so starken Einschränkungen bzgl. TESTS und Σ), sei hier speziell der Fall b = s angenommen; die drei anderen Fälle werden im Anschluß daran betrachtet.

Zu G wird ein PSP Q = (s,M) konstruiert, das genau dann $((2\cdot k + 1)s^2 + 2\cdot s + 2\cdot k + 9)$-realisierbar ist, wenn G k-färbbar ist.

M enthält gemäß der vorausgesetzten Einschränkung nur eine einzige Beispielsberechnung, die wegen b = s eine statementgetreue Instruktionsfolge ist. Die Q zugrunde liegenden Mengen ASSIGNMENTS, TESTS und Σ sind wie folgt definiert:

ASSIGNMENTS = {A,C,D,Z,H} mit:

A = $\lambda w.\ w = xw' \longrightarrow w'aa$

C = $\lambda w.\ w = xw' \longrightarrow w'ac$

D = $\lambda w.\ w = xw' \longrightarrow w'ac$

Z = $\lambda w.\ w = xw' \longrightarrow w'cc$

H = $\lambda w.\ w = xw' \longrightarrow w'$

TESTS = {T} mit:

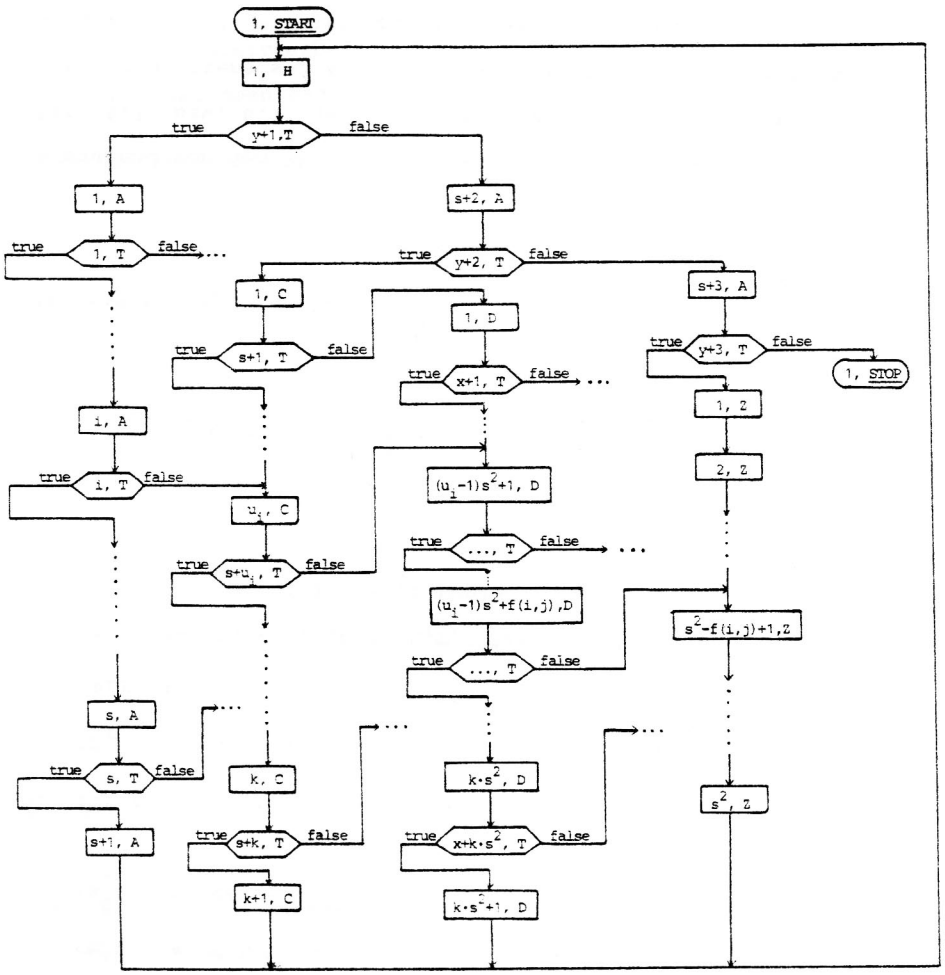
T = $\lambda w.\ w = xw' \longrightarrow x = a$

Σ = {a,c}

wobei $x \in \Sigma$ und $w' \in \Sigma^*$.

Die in M enthaltene Berechnungsfolge ist analog der in Satz 6.11 angegebenen aufgebaut, jedoch mit dem Unterschied, daß hier auch alle durchlaufenen Tests mit aufgeführt sind. Um das Vorgehen zur Konstruktion von M im Vergleich zu Satz 6.11 zu verdeutlichen, ist zunächst in Abb. 6.7 ein Programm P angegeben, das Q $((2\cdot k+1)\cdot s^2 + 2s + 2k + 9)$ - realisiert, falls G k-färbbar ist. F sei dabei k-Einfärbung von G und es gelte $u_i = F(v_i)$.

P entspricht dem U-Graphen UG aus Abb. 6.6. In P dürfen jedoch nur Programmtransitionen vorkommen, sodaß zwischen den einzelnen A-Knoten (C-Knoten usw.) jeweils ein Testknoten eingefügt wurde, von dem dann sowohl ein true- als auch ein false-Zweig ausgehen können. Da [1,H] in Abb. 6.6 vier Nachfolger hat, hier aber nur eine Testinstruktion zur Verfügung steht, wurden in P zwischen [1,H] einerseits und [1,C],[1,Z] und [1,STOP] andererseits noch zwei weitere A-Knoten zwischen die erforderlichen Testknoten eingefügt,



Es gelte: $e_r = (v_i, v_j)$, $F(v_i) = u_1$, $x = s+k$, $y = s+k+k \cdot s^2$

Abbildung 6.7: Programm P (zu Satz 6.12)

damit auch alle Knoten bei einer Programmausführung erreichbar sind. Zusätzlich zur paarweisen Unterscheidbarkeit der Zuweisungsknoten ist die Unterscheidbarkeit der Tests zu beachten; aus diesem Grunde wurden noch zusätzlich die Knoten $[s+1,A]$, $[k+1,C]$ und $[k \cdot s^2+1,D]$ in P aufgenommen.

Entsprechend diesen Ausführungen ergeben sich bei der Anfangsspeicherbelegung st und der Instruktionsfolge SEQ, wenn $M = \{(st,SEQ)\}$ gilt, einige Modifizierungen im Vergleich zu Satz 6.11. Ähnlich wie dort werden zunächst die st und SEQ konstituierenden Teilstücke angegeben, wobei die Hilfsfunktion f wie zuvor definiert ist und für $X,Y \in \{A,C,D,Z\}$ und $x \in \{a,c\}$ gilt:

- X^n steht für $\underbrace{X,\ldots,X}_{n\text{-mal}}$
- $(X,Y)^n$ steht für $\underbrace{X,Y,\ldots,X,Y}_{\text{je n-mal X und Y}}$
- x^n steht für $\underbrace{x \ldots x}_{n\text{-mal}}$

1. $\langle m_A \rangle := (T,A)^{s+1},H$

   $\langle s_A \rangle := a^{s+2}$

2. $\langle m_C \rangle := T,A,(T,C)^{k+1},H$

   $\langle s_C \rangle := ca^{k+2}$

3. $\langle m_Z \rangle := T,A,T,A,T,Z^{s^2},H$

   $\langle s_Z \rangle := cca^{s^2+1}$

4. für $i \in \{1,\ldots,k\}$:

   $\langle m_D^i \rangle := T,A,T,(C,T)^i,(D,T)^{(k-i+1)s^2+1},H$

   $\langle s_D^i \rangle := ca^i ca^{(k-i+1)s^2+1}$

5. für $r \in \{1,\ldots,t\}$: sei $e_r = (v_i,v_j)$

   $\langle m_r \rangle := T,(A,T)^i,C,T,(D,T)^{f(i,j)},Z^{f(i,j)},H$

   $\langle s_r \rangle := a^i cca^{f(i,j)-1} c^{f(i,j)+1}$

   $\langle m_r' \rangle := T,(A,T)^j,C,T,(D,T)^{f(i,j)},Z^{f(j,i)},H$

   $\langle s_r' \rangle := a^j cca^{f(i,j)-1} c^{f(i,j)+1}$

Damit sind SEQ und st gegeben durch:

$SEQ = (\underline{START},H,\langle m_A \rangle,\langle m_C \rangle,\langle m_Z \rangle,\langle m_D^1 \rangle,\ldots\langle m_D^k \rangle,$
$\langle m_1 \rangle,\langle m_1' \rangle,\ldots,\langle m_t \rangle,\langle m_t' \rangle,T,A,T,A,T,\underline{STOP})$

$st = a\langle s_A \rangle\langle s_C \rangle\langle s_Z \rangle\langle s_D^1 \rangle\ldots\langle s_D^k \rangle$
$\langle s_1 \rangle\langle s_1' \rangle\ldots\langle s_t \rangle\langle s_t' \rangle ccc$

Ausgehend von dem PSP Q = ($\underline{s}$,{(st,SEQ)}) kann analog zu Satz 6.10 bzw. 6.11 gezeigt werden, daß zur Realisierung von Q mindestens s+3 A-Knoten, k+1 C-Knoten, $k \cdot s^2+1$ D-Knoten, s^2 Z-Knoten, 1 H-Knoten und $k \cdot s^2+s+k+3$ T-Knoten benötigt werden. Wie in Satz 6.11 kann gefolgert werden, daß diese Anzahl von Knoten genau dann ausreicht, um Q zu realisieren,

wenn G k-färbbar ist. Eine Einfärbung von G korrespondiert dabei wieder zu den LABEL der C-Knoten, die den C-Instruktionen in den Teilen $\langle m_r \rangle$ bzw. $\langle m_r' \rangle$ von SEQ entsprechen (vgl. Programm P in Abb. 6.7). Da die Größe der Standardcodierung von $(Q,(2 \cdot k+1) \cdot s^2 + 2 \cdot s + 2 \cdot k + 9)$ polynomial begrenzt ist durch s und damit durch die Größe der Standardcodierung des durch G und k gegebenen Einfärbungsproblems, handelt es sich also bei der vorgegebenen Konstruktion von Q um eine polynomiale Transformation im Sinne von Def. 6.5, woraus für b = $\underline{s}$ die Behauptung folgt.

Betrachtet man die Folge SEQ', die als erstes Element st gefolgt von den sich durch sukzessive Anwendung der Instruktionen von SEQ daraus ergebenden Speicherbelegungen enthält, so gilt für alle direkt aufeinanderfolgenden Elemente s,s' aus SEQ', daß es nur genau ein I ε INSTRUCTIONS mit I(s) = s' gibt. Für das PSP Q' = ($\underline{ss}$,{SEQ'}) ist daher der IH-Graph bis auf die Spezifizierung $\underline{ss}$ praktisch identisch mit dem IH-Graph von Q, womit die Behauptung auch für statement-getreue Speicherbelegungsfolgen (b = $\underline{ss}$) gezeigt ist, da man anstelle von Q für gegebenes G und k ebenso Q' konstruieren könnte und die obigen Überlegungen auch für Q' gelten.

Entfernt man in SEQ jedes Auftreten von T, so erhält man ein PSP Q"= ($\underline{a}$,{(st,SEQ")}), das genau dann $((k+1) \cdot s^2 + s + k + 6)$-realisierbar ist - bei $b \in \{\underline{a},\underline{as}\}$ werden je nur die Zuweisungsknoten berücksichtigt -, wenn G k-färbbar ist; der Beweis hierzu verläuft wie zuvor. Der Übergang von zuweisungs-getreuen Instruktionsfolgen (b = $\underline{a}$) zu zuwei-

sungsgetreuen Speicherbelegungsfolgen (b = as) verläuft genau so wie der oben angegebene Übergang von b = s zu b = ss.

Trotz der starken Einschränkung bzgl. ASSIGNMENTS, TESTS und Σ in Satz 6.12 sind die für den Beweis des Satzes benützten Instruktionen und die gewählte einfache Speicherstruktur ausreichend, um damit alle partiell rekursiven Funktionen berechnen zu können [Shepherdson, Sturgis 1963].

Beschränkt man noch zusätzlich die Länge der Beispielsfolgen eines PSP Q auf maximal n ∈ℕ, so ist das Problem der Programmsynthese nicht mehr NP-vollständig. In diesem Fall ist die Länge des zu synthetisierenden Programms P, das ja minimal sein soll, begrenzt durch eine Konstante L, nämlich dem Produkt aus maximaler Anzahl und maximaler Länge der Beispielsfolgen. Bei fester Anzahl von Zuweisungs- und Testinstruktionen gibt es aber nur endliche viele Programme $P_1,\ldots,P_r$ der Länge kleiner oder gleich L (vgl. Abschnitt 6.1.1). Ein PSA A könnte also für jedes PSP Q in linearer Zeit in Abhängigkeit von der Größe von Q überprüfen, welches der $P_1,\ldots,P_r$ Q realisiert und ein minimales davon auswählen. Dies trifft selbst dann zu, wenn man die Parameter m,n,p,q,r zwar gleichzeitig, aber ansonsten beliebig begrenzt.

Satz 6.13

Für alle m,n,p,q,r ∈ℕ gilt:
Das Realisierungsproblem mit Einschränkung (-,m,n,p,q,r) ist nicht NP-vollständig, sondern sogar in linearer Zeit lösbar.

Beweis: folgt aus den obigen Überlegungen.

Die in Satz 6.12 betrachtete Einschränkung von fünf der sechs Parameter stellt also insofern eine Grenzsituation dar, als die zusätzliche Beschränkung des sechsten Parameters die NP-Vollständigkeit des Realisierungsproblems und damit auch der Programmsynthese aus Beispielsfolgen beseitigen würde.

6.6 Komplexität der Synthese von Turingmaschinen

6.6.1 Synthese von Turingmaschinen aus Beispielsfolgen

Während sich die vorliegende Arbeit mit dem allgemeinen Problem der Programmsynthese aus unterschiedlichen Arten von Beispielsfolgen beschäftigt, geht es in [Biermann 1972] , [Hwa, Wrightson 1973] und [Raulefs 1973] speziell um die Synthese von Turingmaschinen. Als Eingabe dient eine Folge C von Tripeln (x,y,d), wobei x das gelesene und y das geschriebene Zeichen und d die Richtung der Bewegung des Lese/Schreibkopfes wiedergibt. Synthetisiert werden soll die endliche Kontrolle K einer Turingmaschine TM, sodaß TM die Folge C (bzw. $C_1,\ldots,C_n$ bei mehreren Beispielsfolgen) reproduzieren kann und die Anzahl der Zustände von TM minimal ist. Dieses Problem ist äquivalent zu dem Problem der Synthese eines Mealy-Automaten aus einer Menge von Ein/Ausgabefolgen [Hwa,Wrightson 1973].

6.6.2 Nachweis der NP-Vollständigkeit

In keiner der drei zitierten Arbeiten werden Komplexitätsuntersuchungen vorgenommen. Hier wird nun gezeigt, daß diese Problemstellung zu den schwierigsten Problemen in NP-TIME gehört.

Satz 6.14

(1) Das Problem der Synthese von Turingmaschinen aus Beispielsfolgen ist NP-vollständig.

(2) Die Synthese von Turingmaschinen ist selbst dann noch NP-vollständig, wenn man sich auf eine einzige Beispielsfolge und ein Bandalphabet mit nur fünf verschiedenen Zeichen beschränkt.

(3) Sei X ein Alphabet, $|X| \leq 5$.
Das Ja/Nein-Entscheidungsproblem, ob es für eine einzelne Folge $M \in (X,X)^*, M = (x_1,y_2)\ldots(x_n,y_n)$, und ein $k \in \mathbb{N}$ einen Mealy-Automaten mit höchstens k Zuständen gibt, der bei Eingabe des Wortes $w = x_1\ldots x_n$ das Ausgabewort $w' = y_1\ldots y_n$ liefert, ist NP-vollständig.

Beweis:
Die Zugehörigkeit der drei angesprochenen Problemstellungen zu NP-TIME wird leicht wie in den Sätzen 6.3 und 6.4 gezeigt. Aussage (1) folgt sofort aus (2); (2) selbst wird von (3) impliziert (vgl. 6.1.1). (3) wird i.f. durch polynomiale Transformation des Einfärbungsproblems für Graphen bewiesen.

Das Verfahren aus Satz 6.11 kann in ähnlicher Form auch hier angewandt werden. Zu einem beliebigen Einfärbungsproblem (k,G), $G = (V,E) = (\{v_1,\ldots,v_s\},\{e_1,\ldots,e_t\})$, wird eine Folge $M \in (X,X)^*$ konstruiert, für die es genau dann einen Mealy-Automaten mit $(k+1)\cdot s^2 + s + k + 3$ Zuständen gibt, wenn G k-färbbar ist.

Für G seien o.B.d.A. die in Satz 6.11 gemachten Annahmen erfüllt; f sei wie dort definiert, X = {a,c,d,z,h}. Die Folge M wird aus 2·t + k + s Teilstücken aufgebaut, wobei $(x,y)^n$ für $\underbrace{(x,y)\ldots(x,y)}_{n\text{-mal}}$ steht:

$$M_A := (a,a)^{s+1}(a,h)^2(d,h)(a,h)$$

$$M_C := (c,c)(a,c)^{k+1}(a,h)$$

$$M_Z := (z,z)(a,z)^{s^2}(a,h)$$

für $i \in \{1,\ldots,k\}$:

$$M_D^i := (c,c)(a,c)^i(d,d)(a,d)^{(k-i+1)\cdot s^2}(a,h)$$

für $r \in \{1,\ldots,t\}$: sei $e_r = (v_i,v_j)$, $i < j$

$$M_r := (a,a)^i(c,c)(a,c)(d,d)(a,d)^{f(i,j)}(z,z)(a,z)^{f(i,j)}(a,h)$$

$$M_r' := (a,a)^j(c,c)(a,c)(d,d)(a,d)^{f(i,j)}(z,z)(a,z)^{f(j,i)}(a,h)$$

Mit

$$M_D := M_D^1(z,h)\ldots(z,h)M_D^k$$

$$M_E := M_1(z,h)M_1'(z,h)\ldots(z,h)M_r(z,h)M_r'$$

ist M gegeben durch:

$$M := M_A(z,h)M_A(z,h)M_C(z,h)M_Z(z,h)M_D(z,h)M_E$$

Bezeichnet [a,x] einen Zustand, in dem bei Eingabe a die Ausgabe x erfolgt, so muß es aufgrund der Teilstücke M_A, M_C, M_Z und M_D^1 (vgl. Satz 6.11) in jedem Mealy-Automaten MAT für M

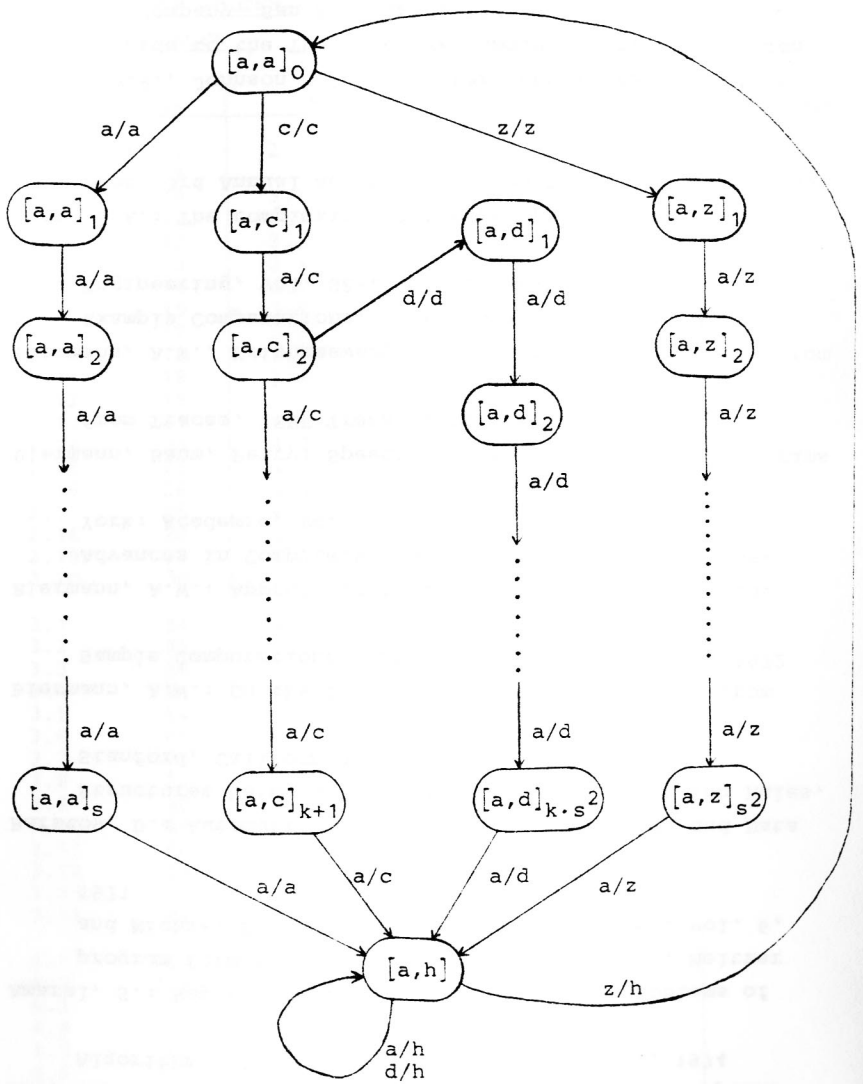


Abb. 6.8 Zustandsgraph eines Mealy-Automaten für M_A, M_C, M_Z, M_D^1

mindestens

$s + 1$	verschiedene	$[a,a]$-Zustände,
$k + 1$	verschiedene	$[a,c]$-Zustände,
s^2	verschiedene	$[a,z]$-Zustände,
$k \cdot s^2$	verschiedene	$[a,d]$-Zustände und
1		$[a,h]$-Zustand

geben. Da alle diese Zustände paarweise verschieden sein müssen, hat MAT mindestens $(k+1) \cdot s^2 + s + k + 3$ Zustände, wobei o.B.d.A. die Übergänge zwischen diesen wie in Abbildung 6.8 skizziert erfolgen und $[a,a]_0$ der Anfangszustand von MAT ist. Analog zu Satz 6.11 und Abb. 6.6 kann gezeigt werden, daß Abb. 6.8 genau dann zu einem (deterministischen) Zustandsgraphen eines Mealy-Automaten für M erweitert werden kann, wenn G k-färbbar ist. Dabei entspricht die jeweilige Zuordnung des Ein/Ausgabepaares (a,c) in M_r bzw. M'_r zu einem der k in Frage kommenden $[a,c]$-Zustände ($[a,c]_{k+1}$ kann nicht gewählt werden), einer k-Einfärbung von G.

Die noch zu zeigenden einzelnen Beweisschritte verlaufen analog zu denen von Satz 6.11.

LITERATURVERZEICHNIS

Aho, Hopcraft,Ullman: The Design and Analysis of Computer Algorithms, Addison-Wesley, Reading, Mass., 1974

Amarel, S.: Representations and modelling in problems of program formation, in: Machine Intelligence, Meltzer and Michie, Ed. New York: American Elsevier, vol. 6, 1971

Barstow, D.: Automatic Construction of Algorithms and Data Structures Using a Knowledge base of Programming Rules, Stanford, California 1977

Biermann, A.W.: On the Inference of Turing Machines from Sample Computations, Artificial Intelligence 3, 1972

Biermann, A.W.: Approaches to automatic programming, in: Advances in Computers, Yovits and Rubinoff, Ed. New York: Academic, vol. 15, 1976

Biermann, Baum, Petry: Speeding up the Synthesis of Programs from Traces, IEEE Trans. Comp., vol. C-24, 1975

Biermann, A.W., Krishnaswamy, R.: Constructing Programs from Example Computations, IEEE Transactions on Software Engineering, vol. SE-2, No. 3, 1976

Cook, S.A.: The complexity of theorem proving procedures, Proc. 3rd Annual ACM Symposium on Theory of Computing, 1971

Garey, M.R., Johnson,D.S.: Computers and Intractability - A Guide to the Theory of NP-Completeness, W.H.Freeman and Company, San Francisco, 1979

Gill, A.: Realization of input-output relations by sequential machines, J.ACM, vol. 13, Jan. 1966

Green, C.C., Barstow, D.: Some rules for the automatic synthesis of programs, Proc. IJCAI (4), 1975

Hwa, J.C.-H., Wrightson, G.: Synthese von Turingsmachinen aus endlichen Mengen von Beispielen, Diplomarbeit, Institut für Informatik I, Universität Karlsruhe, 1973

Lee, R.C.T., Chang, C.L., Waldinger, R.J.: An improved program-synthesizing algorithm and its correctness, C. ACM 17, No. 4, 1974

Manna, Z.: Mathematical Theory of Computation, McGraw-Hill, New York, 1974

Manna, Z., Waldinger, R.J.: Toward automatic program synthesis, C. ACM 14, No. 3, 1971

Raulefs, P.: Automatic Synthesis of Minimal Algorithms from Samples of their Behavior, Interner Bericht INF I-73-2, Inst. für Informatik, Universität Karlsruhe, 1973

Shepherdson, J.C., Sturgis, H.E.: Computability of Recursive Functions, J. ACM 10, 1963

Waldinger, R.J.: Constructing programs automatically using theorem proving, PhD Thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1969

INDEX ZU DEN SEITENZAHLEN DER DEFINITIONEN, SÄTZE, BEISPIELE UND ABBILDUNGEN