OBSCURE: Eine Spezifikations- und Verifikationsumgebung
A 89/06
Fachbereich 14 - Informatik
Universität des Saarlandes
D - 6600 Saarbrücken

Dezember 1989
OBSCURE: Eine Spezifikations- und Verifikationsumgebung

Jacques Loeckx
FB 14 - Informatik
Universität des Saarlandes
D - 6600 Saarbrücken 11
e-mail: loeckx@fb10vax.informatik.uni-saarland.dbp.de

Das Ziel des OBSCURE-Projekts ist die Entwicklung einer Spezifikations- und Verifikationsumgebung. OBSCURE beruht auf einfachen und mathematisch soliden Grundlagen; dabei wird ein gewisser Verlust an Abstraktion im Vergleich zu "klassischen" Systemen in Kauf genommen. Das Projekt wird von der Deutschen Forschungsgemeinschaft unterstützt.

1 Einleitung

Programme werden üblicherweise modularisiert auf der Basis der Kontrollstrukturen. Eine solche Modularisierung führt in natürlicher Weise zu dem Begriff "Prozedur". Bei einer Modularisierung auf der Basis der Datenstrukturen werden die Operationen und die Datenstrukturen zu einer Einheit ("module", "cluster") zusammengeschnürt. Diese Modularisierung führt zum Begriff "abstrakter Datentyp".

konstruktive Spezifikationen verwandt mit, aber weniger abstrakt als axiomatische Spezifikationen.

Eine Spezifikationssprache ermöglicht es, Spezifikationen zusammenzufügen, d.h. aus "kleinen" Spezifikationen "größere" zu bauen. Spezifikationssprachen ermöglichen deshalb den modularen Aufbau von Spezifikationen. Beispiele solcher Spezifikationssprachen sind Clear [BG80], PLUSS [Bi88], OBJ3 [GW88], Larch [GWH85], die Spezifikationssprache von OBSCUR [LL88].


2 Die algorithmische Spezifikationsmethode

Die algorithmische Spezifikationsmethode ist eine konstruktive Spezifikationsmethode. Sie ist präzise beschrieben in [Lo87]. Wir begnügen uns hier mit ein paar allgemeinen Bemerkungen und einem Beispiel.

Eine algorithmische Spezifikation führt eine Sorte und die zugehörigen Operationen ein. Die Trägermenge ist definiert als die Termgsprache, die von den — explizit angegebenen — Konstruktoren erzeugt ist. Die Operationen, die keine Konstruktoren sind, werden definiert durch rekursive Programme (siehe, z.B. [LS87], Abschnitt 3.3).

Eine Spezifikation von Listen von Objekten der Sorte element ist:

```
create sorts list
    opns constructor ε : → list
        constructor ... : list × element → list
        _⊙_ : list × list → list
    programs
        s ⊙ t ← case t of
            ε : s
            t'.ε : (s ⊙ t').ε
        esac
endcreate
```

Die Trägermenge besteht aus Worten wie

ε
ε.ε
ε.ε.ε'

wobei ε und ε' Objekte der Sorte element sind. Die Anwendung der Operation "·" auf die Argumente ε.ε1.ε2 und ε3 führt einfach zu dem Wort ε.ε1.ε2.ε3. Die Anwendung der Operation "⊙" wird illustriert durch:
\[(ε.ε_1) ⊗ ε = ε.ε_1\]
\[(ε.ε_1) ⊗ (ε.ε_2) = ε.ε_1.ε_2\]

3 Die Spezifikationssprache von Obscure

Eine Spezifikationssprache dient dazu, einzelne Spezifikationen zusammenzusetzen. Wir präzisieren darum zuerst die Semantik einer Spezifikation.

\[\text{FIGUR 1: } a, b, c \text{ und } d \text{ sind Sorten und/oder Operationen.} \]
\[\text{Die importierte Signatur besteht aus } a \text{ und } b, \text{ die exportierte aus } a, c, d; \text{ dabei wird } a \text{ vererbt.}\]

In der klassischen Literatur wird eine Spezifikation interpretiert als eine Algebra. In Obscure wird eine Spezifikation interpretiert als eine Funktion, die "importierte" in "exportierte" Algebren abbildet. Etwas genauer, eine Spezifikation in Obscure wird syntaktisch charakterisiert durch eine importierte und eine exportierte Signatur\(^1\); sie wird semantisch charakterisiert durch eine Funktion, die Algebren der importierten Signatur in Algebren der exportierten Signatur abbildet, und die die folgende Persistenzbedingung erfüllt:

\(^1\)Eine Signatur besteht aus Sorten und den "zugehörigen" Operationen
jede Sorte oder Operation, die in beiden Signaturen auftritt, hat in den entsprechenden Algebren dieselbe Bedeutung.

FIGUR 2: Illustration von vier Konstrukten der Spezifikationssprache von OBSCURE. Der Konstrukt "+" "vereinigt" Spezifikationen, "o" drückt die "Komposition" (im Sinne von schrittweiser Verfeinerung) aus.
Im Spezifikationsbeispiel von Abschnitt 2 besteht die importierte Signatur aus der Sorte `element`, die exportierte Signatur aus den Sorten `element` und `list` sowie aus den Operationen "e", "." und "\(\otimes\). Sei A eine importierte Algebra, d.h. eine Algebra mit einer (nicht näher definierten) Trägermenge der Sorte `element` und F die Funktion, die die Semantik des Spezifikationsbeispiels darstellt; die Persistenzbedingung drückt aus, daß die Objekte der Sorte `element` in der importierten Algebra A und in der exportierten Algebra F(A) identisch sind.

Schematisch kann eine Spezifikation als ein Kästchen dargestellt werden — wie illustriert in Figur 1.


Ein Beispiel einer Spezifikation in OBSCURE wird im nun folgenden Abschnitt skizziert.

4 Ein Spezifikationsbeispiel

Es ist nicht möglich, eine kurze und sinnvolle Beispiele anzugeben, die die verschiedenen Konstrukte der Spezifikationssprache illustriert. Statt dessen kommentieren wir ein Beispiel, das die Philosophie des Spezifizierens in OBSCURE illustriert.

Das Ziel ist eine Spezifikation des Dateisystems von UNIX\(^2\) oder, etwas genauer, der UNIX-Kommandos:

\[
\text{cd} \, p, \text{pwd}, \text{ls} \, p, \text{mkdir} \, p, \text{rmrdir} \, p
\]

Dabei ist \( p \) ein Ausdruck; sechs Beispiele für einen solchen Ausdruck sind:

\[
\text{usr}, \text{name1}/\text{name2}, \ldots, /, /\text{ulw}, \text{dir}/\ldots/
\]

\(^2\)UNIX is a trademark of Bell Laboratories
Wir entwickeln nun die Spezifikation durch schrittweise Verfeinerung als 

eine Reihe von Spezifikationen, Modulen genannt. Der erste Modul führt — 

nach einem Test auf syntaktische Korrektheit — ein Kommando aus und hat 
sinnvollerweise den Namen **SYNTACTICALLY-CORRECT-COMMAND**: 

```plaintext
module SYNTACTICALLY-CORRECT-COMMAND is 
create 
  opns Parse-and-execute: 
    command x environment 
    \rightarrow display x environment 
    Issyncorrect: command \rightarrow boolean 
programs 
    Parse-and-execute (c, e) \Leftarrow 
      if Issyncorrect(c) 
      then Execute(c, e) 
      else ("SYNTERROR", e) 
    fi 
    Issyncorrect (c) \Leftarrow case c of 
    cd(p) : Isscorrect (p) 
    pwd : true 
    ls(p) : Isscorrect (p) 
    mkdir(p) : Isscorrect(p) 
    rmdir(p) : Isscorrect(p) esac 
endcreate 
forget opns Issyncorrect 
endmodule 
```

Diese Spezifikation kreiert zwei Operationen, Parse-and-execute und Issyncorre-
cnt, aber keine Sorten. Da eine der Operationen "vergessen" wird, enthält die 
exportierte Signatur nur die zusätzliche Operation Parse-and-execute. Diese 
Operation führt ein Kommando aus, nachdem sie — mit Hilfe der Operation 
Issyncorrect — geprüft hat, daß der Ausdruck p syntaktisch korrekt ist. Et-
was genauer: Die Operation Parse-and-execute hat zwei Argumente der Sorte 
command bzw. environment. Das erste Argument ist das auszuführende Kom-
mando; das zweite stellt die "Umgebung" dar, die im wesentlichen aus dem 
"working directory" innerhalb der "Dateistuktur" besteht. Der Wert der 
Operation Parse-and-execute ist ein Paar. Das erste Element des Paares hat 
die Sorte display und stellt die Ausgabe auf dem Bildschirm dar; falls z.B. 
der Ausdruck p syntaktisch nicht korrekt ist, erscheint die Fehlermeldung 
"SYNTERROR" auf dem Bildschirm. Das zweite Argument stellt die neue 
Umgebung dar. Zur Definition der Operationen Parse-and-execute und Issyncor-
rect wurden verschiedene Sorten und Operationen importiert: die Sorten 
command, environment, display und boolean sowie die Operationen Execute,
"-", Isscorrect, cd, pwd, ls, mkdir, rmdir und true. Entsprechend der Philoso-
phie der schrittweisen Verfeinerung werden nun diese Sorten und Operationen mittels weiterer Spezifikationen eingeführt.

Eine solche Spezifikation ist:

module COMMAND is
  create
    sorts command
    opns constructor cd : expression → command
        constructor pwd : → command
        constructor ls : expression → command
        constructor mkdir : expression → command
        constructor rmdir : expression → command
  endcreate
endmodule

Diese Spezifikation kreiert (und exportiert) die Sorte command und die Operation cd, pwd, ls, mkdir und rmdir, die von der Spezifikation SYNTACTICALLY-CORRECT-COMMAND benötigt (weil importiert) werden.

Eine weitere Spezifikation exportiert die von der Spezifikation SYNTACTICALLY-CORRECT-COMMAND importierte Operation Execute:

module EXECUTE-COMMAND is
  create
    opns E x e c u t e : command × environment → display × environment
        programs Execute (c, e) ≔
            if Iscompatible(c, e)
                then Evaluate (c, e)
                else (Errormessage (c, e), e) fi
  endcreate
endmodule

Diese Spezifikation importiert ihrerseits die Operationen Iscompatible, Evaluate und Errormessage. Dabei ist die Intention dieser Operationen wie folgt: Iscompatible "verwirft" Befehle wie

\[
\text{cd}\ nn
\]

falls nn nicht ein Sohn des "working directory" ist, Evaluate sorgt für die weitere Verarbeitung des Befehls und Errormessage erzeugt die zutreffende Fehlermeldung. Diese Intention muß natürlich noch festgelegt werden und zwar in den Spezifikationen, die diese Operationen kreieren.

Wir spezifizieren nun die Sorte environment:
module ENVIRONMENT is
  create
    sorts environment
    opns constructor < _ , _ , _ > :
      dir-tree × path × path → environment
    Icorrect : environment → boolean
    :
    programs
      Icorrect (e) ⇐ case e of
      < d , p₁ , p₂ > :
        Ispathin(p₁ , d)
        ∧ Ispathin(p₂ , d)
        esac
      :
  endcreate
endmodule

Diese Spezifikation führt u.a. die dreistellige "mixfix" Operation < _ , _ , _ > ein. In einem solchen Tripel < d , p₁ , p₂ > stellt die Komponente d den Dateibaum dar, p₁ der Pfad von der Wurzel zum "working directory" und p₂ der Pfad von der Wurzel zum "home directory".


5 Die OBSCURE-Umgebung

bung fertiggestellt ist, beschränken wir uns hier auf die Beschreibung dieses Teils der OBSCURE-Umgebung.


![Diagram of OBSCURE environment](image)

**Figur 3: Die Spezifikationsumgebung von OBSCURE**

Der **Parser** führt die syntaktische Analyse einer Spezifikation aus und überprüft die Kontextbedingungen. Spezifikationen, die syntaktisch korrekt sind und alle Kontextbedingungen erfüllen, werden in die Moduldatenbank eingetragen.

Der "pretty printer" gibt den Text einer Spezifikation und/oder deren importierte und exportierte Signatur aus.

Die **Moduldatenbank** speichert Spezifikationen, deren importierte und ex-

Der Interpreter berechnet den Wert eines vom Benutzer angegebenen Terms ("rapid prototyping").

Der Übersetzer übersetzt OBSCURE-Spezifikationen in die Programmiersprache C++ oder ML. Er ermöglicht die Einbindung von OBSCURE-Spezifikationen in Programme, die in diesen Programmiersprachen geschrieben sind.


Die Spezifikationsumgebung wurde auf einem Sun 3/60-Rechner entwickelt und auf die Rechner Siemens MX2 und VAX 8700 portiert. Der gesamte Speicherbedarf beträgt etwa 3 MByte.

Nähere Details findet man in [LP89].

6 Schlußbemerkungen

Aus den mit dem OBSCURE-System spezifizierten Beispielen können jetzt schon einige wichtige Folgerungen gezogen werden.

Das schwierigste Problem beim Spezifizieren besteht darin, ein "adäquates" Modell zu finden, d.h. ein Modell, das die Spezifikation einfach und durchsichtig macht (vgl. [Lo89]). Es ist z.B. wichtig, einen Unterschied zu machen zwischen einem leeren "directory" und einer (möglicherweise leeren) Datei, oder zwischen einem Pfad und dem Argument eines Kommandos wie cd oder ls.

Angesichts der großen Anzahl der eingeführten Namen ist die Benutzung einer Spezifikationsumgebung sehr hilfreich; insbesondere verliert man schnell den Überblick über die Operationen und deren Stelligkeit.


Zwei Aspekte von OBSCURE sind noch in Bearbeitung: die Verifikationsmethodologie und der Implementierungsbegriff. Zur Verifikation wurden einige wichtige Grundlagen gelegt [Tr89, Ho90]; die Realisierung der Verifi-


Referenzen


[Bi89] Bidoit, M., PLUSS, a language for the development of modular algebraic specifications, These d'etat, Universite de Paris-Sud, May 1989


