Formale Spezifikations- und Verifikationstechniken

Prof. Dr. K. Madlener

26. April 2004

Studiengang „Informatik“, „Angewandte Informatik“ und „WIWI-Inf.“ SS04
Prof. Dr. Madlener
Universität Kaiserslautern

Vorlesung:
Mo 08.30-10.00 34/420 Mi 08.30-10.00 34/420

- Informationen `http://www-madlener.informatik.uni-kl.de/ag-madlener/teaching/ss2004/fsvt/fsvt.html`
- Bewertungsverfahren:
  Übungen (Leistungsnachweis) + Abschlussklausur (Credits)
- Erste Abschlussklausur: (Schriftlich oder mündlich)
- Übungen (Termine und Anmeldung): Siehe WWW-Seite

Literatur

M. O'Donnell.
Computing in Systems described by Equations, LNCS 58, 1977.
Equational Logic as a Programming language.

J. Avenhaus.
Reduktionssysteme, (Skript), Springer 1995.

Cohen et.al.
The Specification of Complex Systems.

Bergstra et.al.
Algebraic Specification.

Barendregt.
Functional Programming and Cambda Calculus. Handbook of TCS, 321-363, 1990.

Literatur

Gehani et.al.
Software Specification Techniques.

Huet.
Confluent Reductions: Abstract Properties and Applications to TRS, JACM, 27, 1980.

Nivat, Reynolds.
Algebraic Methods in Semantics.

Loeckx, Ehrich, Wolf.
Specification of Abstract Data Types, Wyley-Teubner, 1996.

J.W. Klop.
Term Rewriting System. Handbook of Logic, INCS, Vol. 2, Abransky, Gabbay, Maibaum.

Literatur

Ehrig, Mahr.
Fundamentals of Algebraic Specification.

Peyton-Jones.
The Implementation of Functional Programming Language.

Plasmeister, Eekelen.
Functional Programming and Parallel Graph Rewriting.

Astesiano, Kreowski, Krieg-Brückner.
Algebraic Foundations of Systems Specification (IFIP).

N. Nissanke.
Formal Specification Techniques and Applications (Z, VDM, algebraisch), Springer 1999.

Literatur

Turner, McCluskey.
The construction of formal specifications. (Modell basiert (VDM) + algebraisch (OBJ)).

Goguen, Malcom.
Algebraic Semantics of Imperative Programs.

H. Dörr.
Efficient Graph Rewriting and its Implementation.

B. Potter, J. Sinclair, D. Till.
An introduction to Formal Specification and Z. Prentice Hall, 1996.

Literatur

J. Woodcok, J. Davis.
Using Z: Specification, Refinement and Proof, Prentice Hall 1996.

J.R. Abrial.
The B-Book; Assigning Programs to Meanings. Cambridge U. Press, 1996.

Zielsetzung - Inhalt

Allgemeine Zielsetzung:
Methoden zur Spezifikation, Verifikation und Implementierung

Inhaltsübersicht

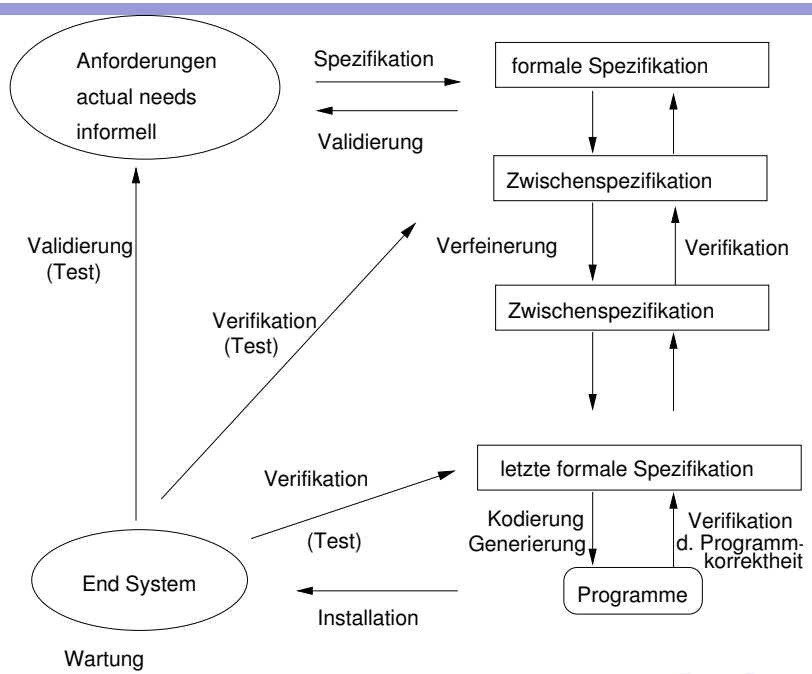
- Die Rolle formaler Spezifikationen
- Algebraische Spezifikation, Gleichheitssysteme
- Reduktionssysteme, Termersetzungssysteme
- Gleichheitskalküle und Programmierung
- Verwandte Kalküle: λ-Kalkül, Kombinatorenkalkül
- Implementierung, Reduktionsstrategien, Graphersetzung

Die Rolle formaler Spezifikationen

- Software und Hardware Systeme müssen wohldefinierte Aufgaben (Anforderungen) erfüllen.
  Software Engineering hat als Ziele
  - Definition der Kriterien zur Evaluation von SW-Systemen
  - Methoden und Techniken zur Entwicklung von SW-Systemen, die diese Kriterien erfüllen
  - Charakteristiken von SW-Systemen
  - Entwicklungsprozesse
  - Werkzeuge
- Vereinfachte Sicht SE-Prozess: Konstruktion einer Folge von Beschreibungen der zu produzierenden SW.
  Ziel: Menge von Dokumenten, die ein ausführbares Programm enthält.

Modelle der SW-Entwicklung

- Wasserfallmodell, Spiralmodell,. . .
  Phasen ≡ Aktivitäten + Teilprodukte
  In jeder Stufe des EP
  Beschreibung, die eine Spezifikation der SW ist, d. h. Festlegung was sie leisten muss, aber nicht immer wie es gemacht wird.



Erläuterung

- Erste Spezifikation: globale Spezifikation
  Grundlage für die Entwicklung "Vertrag" zwischen Entwicklern und Auftraggeber
- Zwischenspezifikationen: Grundlage der Kommunikation zwischen Entwicklern.
- Programme

Entwicklungsparadigmen

- strukturiertes Programmieren
- Entwerfen+Programmieren
- Transformationsparadigmen

Eigenschaften von Spezifikationen

Konsistenz Vollständigkeit

- Validierung der globalen Spezifikation bzgl. der Anforderungen.
- Verifikation der Zwischenspezifikationen bzgl. der Vorgänger.
- Verifikation der Programme bzgl. der Spezifikation.
- Verifikation des integrierten Endsystems bzgl. der globalen Spezifikation.
- Aktivitäten: Validierung, Verifikation, Konsistenz, Vollständigkeit
- Werkzeugunterstützung

Anforderungen

Funktionale -	- nicht Funktionale
was	Zeitaspekte
⋮	Robustheit
wie	Stabilität
	Anpassbarkeit
	Ergonomie
	Wartbarkeit

Eigenschaften
Korrektheit: Erfüllt das implementierte System die Anforderungen.
Testen Validieren Verifizieren

Anforderungen

- Globale Spezifikation beschreibt so genau wie möglich, was gemacht werden soll.
- Abstraktion vom wie
  Vorteile
  - apriori: Referenzdokument, kompakter, lesbarer.
  - aposteriori: Folge von Spezifikationen, Verfolgbarkeit der Entwurfsentscheidungen, Wiederverwendung, Wartung.
- Problem: Größe und Komplexität der Systeme.
  Prinzipien, die unterstützt werden sollten
  - Verfeinerungsprinzip: Abstraktionsstufen
  - Strukturierungsmechanismen
    Zerlegungs- und Modularisierungsprinzipien
    Objektorientierung
  - Verifikations- und Validierungskonzepte

Beschreibung der Anforderungen::Spezifikation

- Wahl der Spezifikationstechnik hängt vom System ab, oft sind mehrere Spezifikationstechniken notwendig.(Was—Wie).
  Art der Systeme:
  Rein funktionsorientiert (I/O), Reaktiv, Eingebettet.
- Problem universeller Spezifikationstechniken
  schwer verständlich, Mehrdeutigkeiten, Werkzeuge, Größe ...
  z. B. UML
- Wunsch: Kompakte gut lesbare genaue Spezifikationen

Hier: formale Spezifikationstechniken

Formale Spezifikationen

- Eine Spezifikation, die in einer formalen Spezifikationssprache beschrieben wird, legt alle erlaubten Verhalten des spezifizierten System fest.
- 3 Aspekte: Syntax, Semantik, Inferenzsystem
  - Syntax Was darf geschrieben werden: Text mit Struktur, Eigenschaften oft als Formeln einer Logik.
  - Semantik Welche Modelle sind mit der Spezifikation assoziiert, Modelle der Spezifikation.
  - Inferenzsystem Folgerung (Herleitung) von Eigenschaften des Systems.

Formale Spezifikationen

- Zwei große Klassen:

Modell orientiert	-	-	Eigenschaften orientiert
(konstruktiv)			(deklarativ)
VDM, Z, B, ASM			Signatur (Funktionen, Prädikate)
Konstruktion eines eindeutigen Modells aus vorhandenen Datenstrukturen und Konstruktionsregeln			Eigenschaften (Formeln Axiome)
Korrektheitsbegriff			Modelle
			algebraische Spezifikation
			AFFIRM, OBJ, ASF,...

- Operationale Spezifikationen: Petri Netze, Prozess Algebren, Automatenbasiert (SDL).

Wozu formale Spezifikationen?

- Begriff der Korrektheit eines Programms ohne formale Spezifikation nicht wohldefiniert.
- Verifikation ohne formale Spezifikation nicht möglich.
- Verfeinerungsbegriff wohldefiniert.

Wunschliste

- Abstand zwischen Spezifikation und Programm nicht zu groß: Generatoren, Transformatoren.
- Nicht zu viele verschiedene Formalismen/Notationen.
- Werkzeugunterstützung.
- Rapid Prototyping.
- Regeln zur Erstellung von Spezifikationen, die bestimmte Eigenschaften garantieren (z. B. Konsistenz + Vollständigkeit).

Formale Spezifikationen

- Vorteile:
  Mathematische (Logik basierte) Behandlung von Korrektheit, Äquivalenz, Vollständigkeit, Konsistenz, Verfeinerung, Komposition usw.,
  Werkzeugunterstützung möglich, Einsatz und Kopplung von unterschiedlichen Werkzeugen.
- Nachteile:

Verfeinerungen

Abstraktionsmechanismen

- Datenabstraktion (Repräsentation)
- Konstrollabstraktion (Reihenfolge)
- Prozedurale Abstraktion (nur I/O Beschreibung)

Verfeinerungsmechanismen

- Wähle Datenrepräsentation (Menge durch Listen)
- Wähle Reihenfolge der Berechnungsschritte
- Entwerfe Algorithmus (Sortieralgorithmus)

Begriff: Implementierungskorrektheit

- Beobachtbare Äquivalenzen
- Verhaltensäquivalenzen

Strukturierung

Probleme: Strukturierungsmechanismen

- Horizontal:
  Zerlegung/Aggregation/Kombination/Erweiterung/
  Parametrisierung/Instanziierung
  (Komponenten)

  Ziel: Vollständigkeit
- Vertikal:
  Realisierung von Verhalten
  Information Hiding/Verfeinerung

  Ziel: Effizienz und Korrektheit

Werkzeugunterstützung

- Syntaktische Unterstützung (Grammatiken, Parser,...)
- Verifikation: Theorembeweisen (Beweisverpflichtungen)
- Prototyping (Ablauffähige Spezifikationen)
- Code Generierung (Aus Spec C Code generieren)
- Testen (Aus Spec Testfälle für Programm)

Wunsch:

Aus Syntax und Semantik der Spezifikationssprache Generierung der Werkzeuge

Beispiel: deklarativ

Eingeschränkte Logik: z. B. Gleichheitslogik
Axiome: $\forall X\ t_1 = t_2$ t_1, t_2 Terme.
Regeln: Gleiches durch Gleiches ersetzen.(Gerichtet).
Terme $\approx$ Namen für Objekte (Bezeichner), Strukturierung, Aufbau der Objekte.
Abstraktion: Terme als Elemente einer Algebra, Termalgebra.

- Axiome induzieren Kongruenz auf Termalgebra
- Unabhängige Teilaufgaben
  - Beschreibung der Eigenschaften
  - Repräsentation der Terme
- Operationalisierung
  - spec, t Term gebe den „Wert“ von t aus, d. h. $t' \in \mathrm{Wert}(spec)$ mit spec $\models t = t'$.
  - $\rightsquigarrow$ Funktionale Programmierung: LISP, CAML,...
    $F(t_1, \ldots, t_n)$ eval( ) $\rightsquigarrow$ Wert.

Beispiel: Modellbasiert konstruktiv: VDM

Eindeutigkeit, Standard (Notationen), implementierungsunabhängig, formal manipulierbar, abstrakt, strukturiert, expressiv, Konsistenz
Beispiel: Model (zustands)-basierte Spezifikationstechnik VDM

- Mengenlehre basiert, PL 1-Stufe, Vor- Nachbedingungen.
  Primitive Typen: $\mathbb{B}$ Boolean $\{true, false\}$
  $\mathbb{N}$ natural $\{0, 1, 2, 3, \dots\}$
  $\mathbb{Z}, \mathbb{R}$
- Mengen: $\mathbb{B}$-Set: Mengen von $\mathbb{B}$-'s.
- Mengenoperationen: $\in$: Element, Element-Set $\rightarrow \mathbb{B}, \cup, \cap, \setminus$
- Folgen: $\mathbb{Z}^*$: Folgen ganzer Zahlen.
- Folgenoperationen: $\frown$: Folgen, Folgen $\rightarrow$ Folgen. „Konkatenation"
  z.B. $[\ ] \frown [true, false, true] = [true, false, true]$
  len: Folgen $\rightarrow \mathbb{N}$, hd: Folgen $\rightsquigarrow$ Elem (partiell).
  tl: Folgen $\rightsquigarrow$ Folgen, elem: Folgen $\rightarrow$ Elem-Set.

Operationen in VDM

VDM-SL: System Zustand, Operationsspezifikation

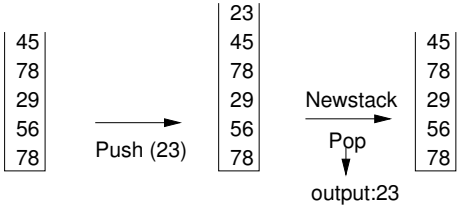
Format:

Operation-Identifier (Inputparameters) Output Parameters
Pre-Condition
Post-Condition

z. B.
$Int\_SQR(x : \mathbb{N})z : \mathbb{N}$
pre $x \geq 1$
post $(z^2 \leq x) \wedge (x < (z+1)^2)$

Beispiel VDM: Beschränkter Keller

- Operationen: · Init · Push · Pop · Empty · Full



Contens $= \mathbb{N}^*$ Max_Stack_Size $= \mathbb{N}$

- STATE STACK OF
  s : Contents
  n : Max_Stack_Size
  inv : mk-STACK$(s, n) \triangleq$ len $s \leq n$
  END

Beschränkter Keller

Init(size : $\mathbb{N}$)
ext wr s : Contents
wr n : Max_Stack_Size
pre true
post $s = [\ ] \wedge n =$ size

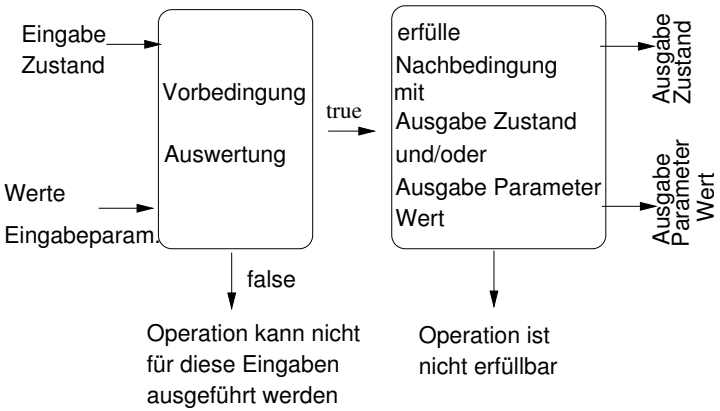
Full( )$b : \mathbb{B}$
ext rd s : Contents
rd n : Max_Stack_Size
pre true
post $b \Leftrightarrow$ (len $s = n$)

Push($c : \mathbb{N}$)
ext wr s : Contens
rd n : Max_Stack_Size
pre len $s < n$
post $s = [c] \frown \overleftarrow{s}$

Pop( )$c : \mathbb{N}$
ext wr s : Contens
pre len $s > 0$
post $\overleftarrow{s} = [c] \frown s$

$\rightsquigarrow$ Proof-Obligations

Allgemeine Form VDM-Operationen



Allgemeine Form VDM-Operationen

Proof Obligations:
Für jede zulässige Eingabe gibt es eine zulässige Ausgabe.

$$\forall s_i, i \cdot (\text{pre-op}(i, s_i) \Rightarrow \exists s_o, o \cdot \text{post-op}(i, s_i, o, s_o))$$

Falls Zustandsinvarianten vorhanden:

$$\forall s_i, i \cdot (\text{inv}(s_i) \wedge \text{pre-op}(i, s_i) \Rightarrow \exists s_o, o \cdot (\text{inv}(s_o) \wedge \text{post-op}(i, s_i, o, s_o)))$$

bzw.

$$\forall s_i, i, s_o, o \cdot (\text{inv}(s_i) \wedge \text{pre-op}(i, s_i) \wedge \text{post-op}(i, s_i, o, s_o) \Rightarrow \text{inv}(s_o))$$

Siehe z. B.Turner, McCluskey The Construction of Formal Specifications oder Jones C.B. Systematic SW Development using VDM Prentice Hall.

Keller algebraisch spezifiziert

Bestandteile einer algebraischen Spezifikation: Signatur (Sorten, Operationsnamen mit Stelligkeiten), Axiome (oft nur Gleichungen)

SPEC STACK
USING NATURAL, BOOLEAN "Namen bekannter SPEC's"
SORT stack "Hauptsorte"
OPS init : → stack "Konstante der Sorte stack, leerer Keller"
push : stack nat → stack
pop : stack → stack
top : stack → nat
is_empty? : stack → bool
stack_error : → stack
nat_error : → nat
(Signatur festgelegt)

Axiome für Keller

FORALL s : stack n : nat
AXIOMS
is_empty? (init) = true
is_empty? (push (s, n)) = false
pop (init) = stack_error
pop (push (s, n)) = s
top (init) = nat_error
top (push (s,n)) = n

Terme bzw. Ausdrücke:
top (push (push (init, 2), 3)) "meint" 3
Wie wird der beschränkte Keller algebraisch spezifiziert?
Semantik? Operationalisierung?

Variante: Z - B Methoden: Spezifikation-Entwurf-Programme.

- Abdeckung: Technische Spezifikation (was), Entwurf über Verfeinerung, Architektur (Schichten Architektur), Generierung ausführbarer Codes).
- Beweise: Programm Konstruktion $\equiv$ Beweis Konstruktion. Abstraktion, Instantiierung, Zerlegung.
- Abstrakte Maschinen: Kapselung von Information (Modul, Klassen, ADT).
- Daten und Operationen: SWS besteht aus abstrakten Maschinen. Abstrakte Maschinen „enthält“ Daten und „bietet“ Operationen. Daten können nur über Operationen erreicht werden.

Z - B Methoden: Spezifikation-Entwurf-Programme.

- Datenspezifikation: Mengen, Relationen, Funktionen, Folgen, Bäume. Gesetze (statisch) mit Hilfe von Invarianten.
- Operatorenspezifikation: Nicht ausführbarer „Pseudocode“. Ohne Schleifen:
  Vorbedingung + atomare Aktion
  PL1 verallgemeinerte Substitution
- Verfeinerung ($\rightsquigarrow$ Implementierung).
- Verfeinerung (als Spezifikationstechnik).
- Verfeinerungstechniken:
  Entfernung nicht ausführbarer Teile, Einführung von Kontrollstrukturen (Schleifen). Transformation abstrakter mathematischer Strukturen.

Z - B Methoden: Spezifikation-Entwurf-Programme.

- Verfeinerungsschritte: Verfeinerung wird in mehreren Schritten durchgeführt. Abstrakte Maschinen wird neu aufgebaut. Operationen für Benützer bleiben gleich nur interne Veränderungen. Zwischen Stufen: Misch Code.
- Geschachtelte Architektur: Regel: nicht all zu viele Verfeinerungsschritte, besser Zerlegung.
- Bibliothek: vordefinierte abstrakte Maschinen, Einkapselung klassischer DS.
- Wiederverwendung
- Code Generierung: Letzte abstrakte Maschine kann leicht in imperativer Sprache übersetzt werden.

Z - B Methoden: Spezifikation-Entwurf-Programme.

Wichtig hierbei:

- Notation: Mengenlehre +PL1, Standard Mengenoperationen, kartesische Produkte, Potenzmengen, Mengen Einschränkungen $\{x \mid x \in s \wedge P\}$, P Prädikat.
- Schemata (Schemes) in Z Muster zur Deklaration und Constraint {Zustandsbeschreibungen}.
- Typen.
- Natürliche Sprache: Verbindung Math Obj $\rightarrow$ Objekte der modellierten Welt.
- Siehe Abrial The B-Book, Potter, Sinclair, Till An Introduccion to Formal Specification and Z, Woodcock, Davis Using Z Specification, Refinement, and Proof $\rightsquigarrow$ Literatur