Formale Spezifikations- und Verifikationstechniken

Prof. Dr. K. Madlener

23. Februar 2006

Studiengang „Informatik“, „Angewandte Informatik“ und „WIWI-Inf.“ SS04

Prof. Dr. Madlener
Universität Kaiserslautern

Vorlesung:
Di 08.15-09.45 13/222 Fr 08.15-09.45 13-222
Ausweichtermin: Mo 08.15-09.45 36/265 erster Termin 14.11.

- Informationen `http://www-madlener.informatik.uni-kl.de/ag-madlener/teaching/ws2005-2006/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.


E. Börger, R. Stärk
Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, 2003.

Zielsetzung - Inhalt

Allgemeine Zielsetzung:
Methoden zur Spezifikation, Verifikation und Implementierung

Inhaltsübersicht

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

Inhalt der Vorlesung

Rolle formaler Spezifikationen

Abstract State Machines: ASM- Spezifikationsmethode

- Grundlagen
- Sequentielle Algorithmen
- ASM-Spezifikationen

Verteilte ASM: Concurrency, reactivity, time

- Grundlagen:Ordnungen, CPO's, Beweistechniken
- Induktion
- DASM
- Reaktive und Zeitabhängige Systeme

Verfeinerung

- Vortrag Börger ASM-Buch

Algebraische Spezifikation

Algebraische Spezifikation - Gleichheitskalküle

- Grundlagen
- Einführung
- Algebren
- Algebraische Grundlagen
- Signatur - Terme
- Striktheit - Positionen-Teilterme
- Interpretationen: sig-Algebren
- Kanonische Homomorphismen
- Gleichungsspezifikationen
- Substitution
- Lose Semantik
- Zusammenhänge zwischen $\models, =_E, \vdash_E$
- Satz von Birkhoff

Algebraische Spezifikation: Initiale Semantik

Algebraische Spezifikation: Operationalisierung

Reduktionssysteme

- Abstrakte Reduktionssysteme
- Prinzip der noetherschen Induktion
- Wichtige Zusammenhänge
- Hinreichende Bedingung für Konfluenz
- Äquivalenzrelationen und Reduktionsrelationen
- Transformation mit dem Inferenzsystem
- Konstruktion der Beweisordnung

Termersetzungssysteme

- Grundlagen
- Kritische Paare, Unifikation
- Lokale Konfluenz
- Konfluenz ohne Terminierung
- Knuth-Bendix Vervollständigung

Berechenbarkeit und Implementierung

Gleichungskalkül und Berechenbarkeit

- Implementierungen
- Primitiv Rekursive Funktionen
- Rekursive und Partiell Rekursive Funktionen
- Partiell Rekursive Funktionen und Register Maschinen
- Berechenbare Algebren

Reduktionsstrategien

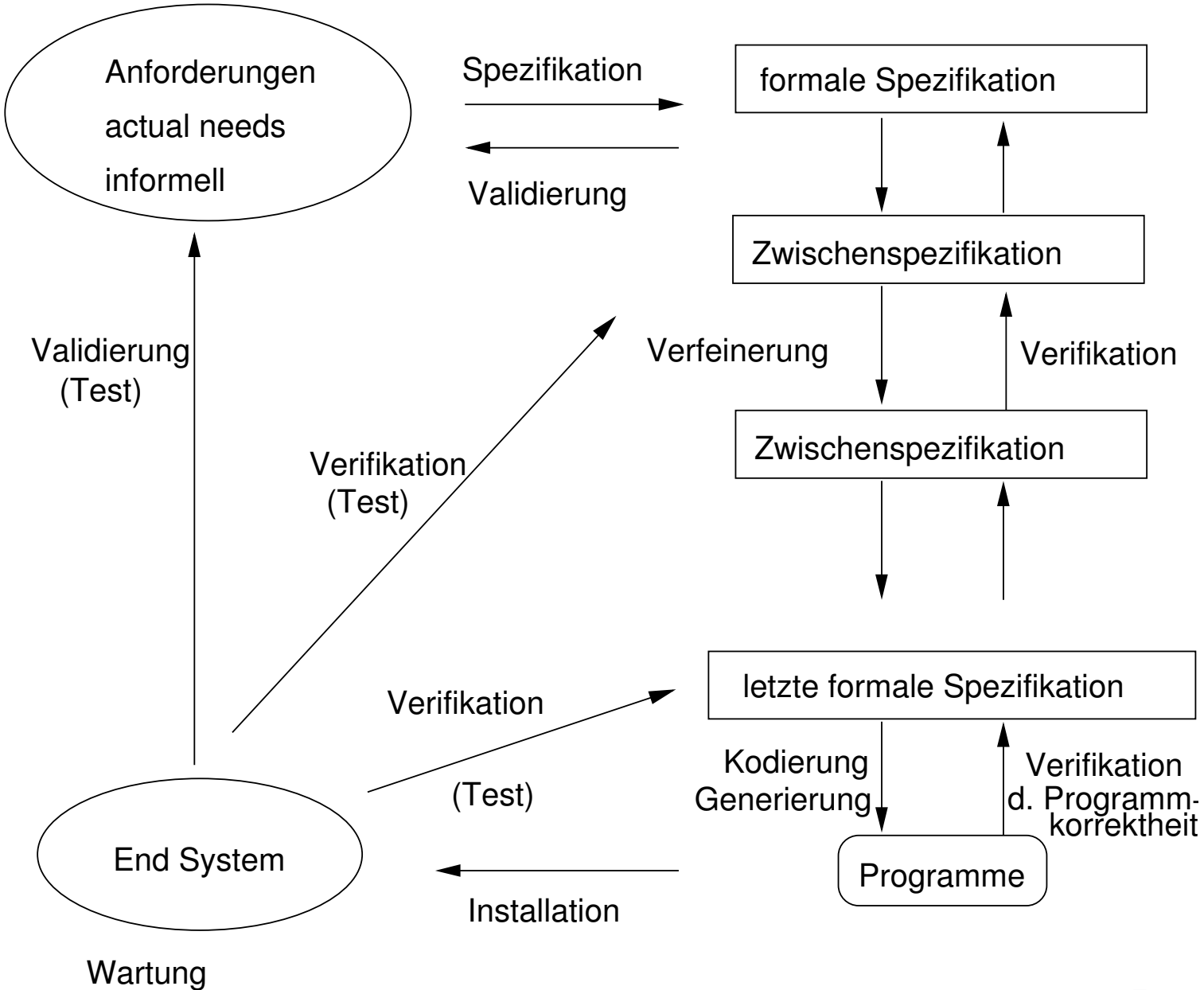
- Allgemeines
- Orthogonale Systeme
- Strategien und Ableitungslängen
- Sequentielle Orthogonale TES: Call by Need

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 $\equiv$ 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.


Anforderungen
actual needs
informell
Spezifikation
Validierung
formale Spezifikation
Zwischenspezifikation
Zwischenspezifikation
Validierung
(Test)
Verfeinerung
Verifikation
Verifikation
(Test)
Verifikation
(Test)
letzte formale Spezifikation
Kodierung
Generierung
Verifikation
d. Programm-
korrektheit
Programme
End System
Installation
Wartung

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		Eigenschaften
eindeutigen Modells		(Formeln Axiome)
aus vorhandenen		
Datenstrukturen und		Modelle
Konstruktionsregeln		algebraische Spezifikation
Korrektheitsbegriff		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: $\forall X\ t_1 = t_2 \qquad t_1, t_2$ 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. $t' \in$ 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 $\{\text{true}, \text{false}\}$
  $\mathbb{N}$ 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. $[\ ] \frown [\text{true}, \text{false}, \text{true}] = [\text{true}, \text{false}, \text{true}]$
  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

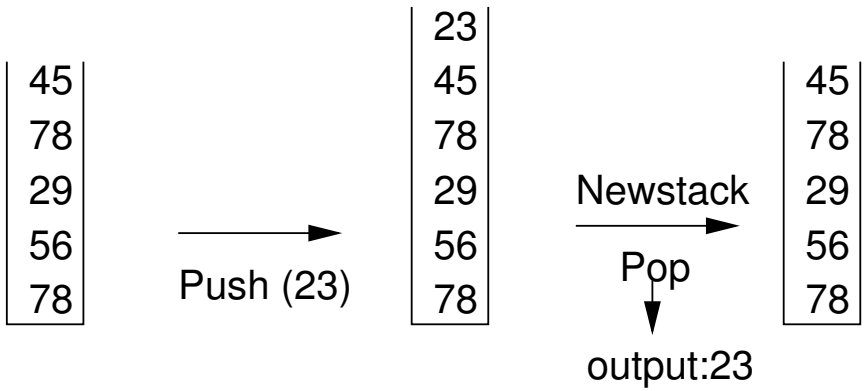
Format:

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$(s, n) \triangleq \text{len}\, s \leq n$
  END

Beschränkter Keller

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

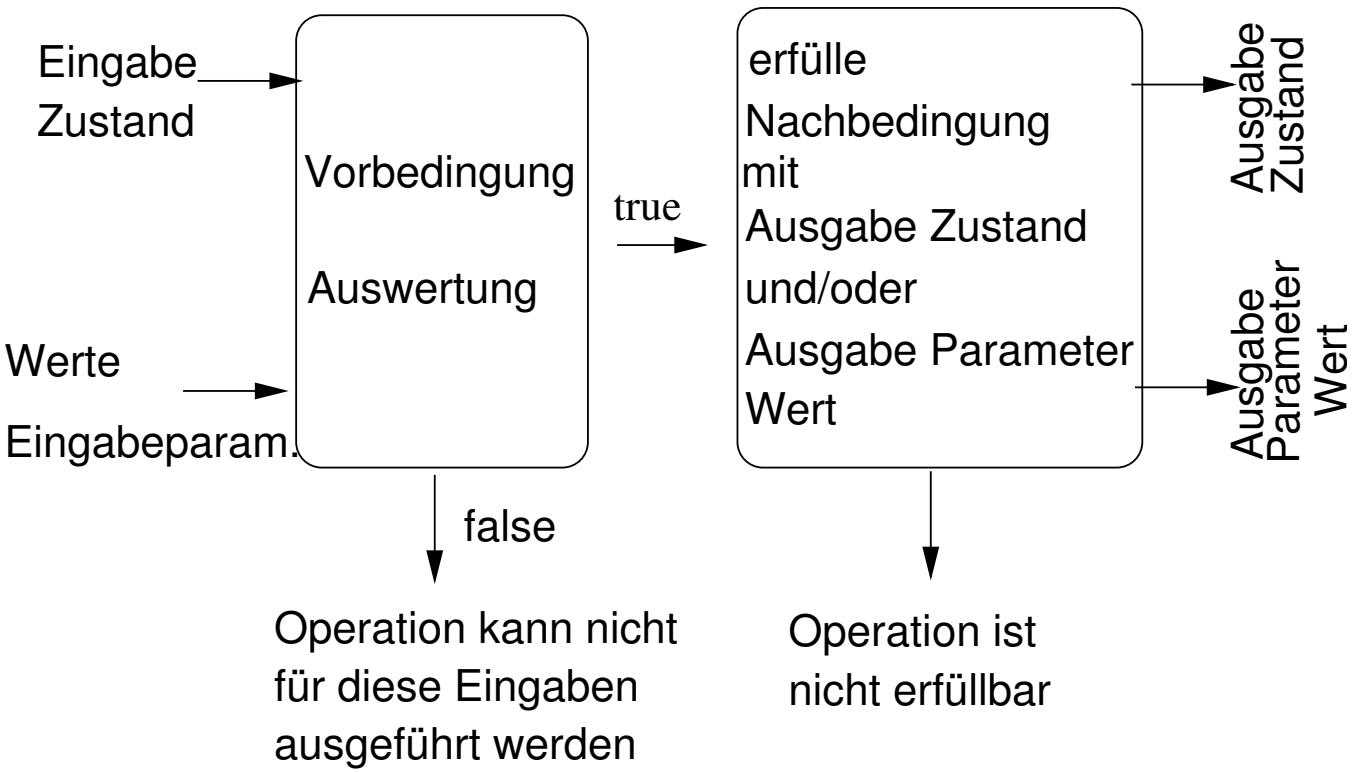
Full( )b : $\mathbb{B}$
ext rd s : Contents
rd n : Max_Stack_Size
pre true
post $b \Leftrightarrow (\text{len}\, s = n)$

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

Pop( )c : $\mathbb{N}$
ext wr s : Contens
pre $\text{len}\, s > 0$
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 : $\longrightarrow$ stack "Konstante der Sorte stack, leerer Keller"
push : stack nat $\longrightarrow$ stack
pop : stack $\longrightarrow$ stack
top : stack $\longrightarrow$ nat
is_empty? : stack $\longrightarrow$ bool
stack_error : $\longrightarrow$ stack
nat_error : $\longrightarrow$ 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 $\longrightarrow$ 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

Einführung in ASM: Grundlagen

Anpassungsfähige Spezifikationstechnik.

Modellierung in der richtigen Abstraktionsebene.

Natürliche und leicht verständliche Semantik.

Material: Siehe http://www.di.unipi.it/AsmBook/

Theoretische Fundierung: ASM Thesen

Abstrakte Zustandsmaschinen als Berechnungsmodelle

Turing Maschinen (RAM, part.rek. Fkt,..) dienen als Berechnungsmodell, d.h. präzisierung der berechenbaren Funktionen. Prinzipiell lässt sich jede algorithmische Lösung durch eine geeignete TM simulieren.

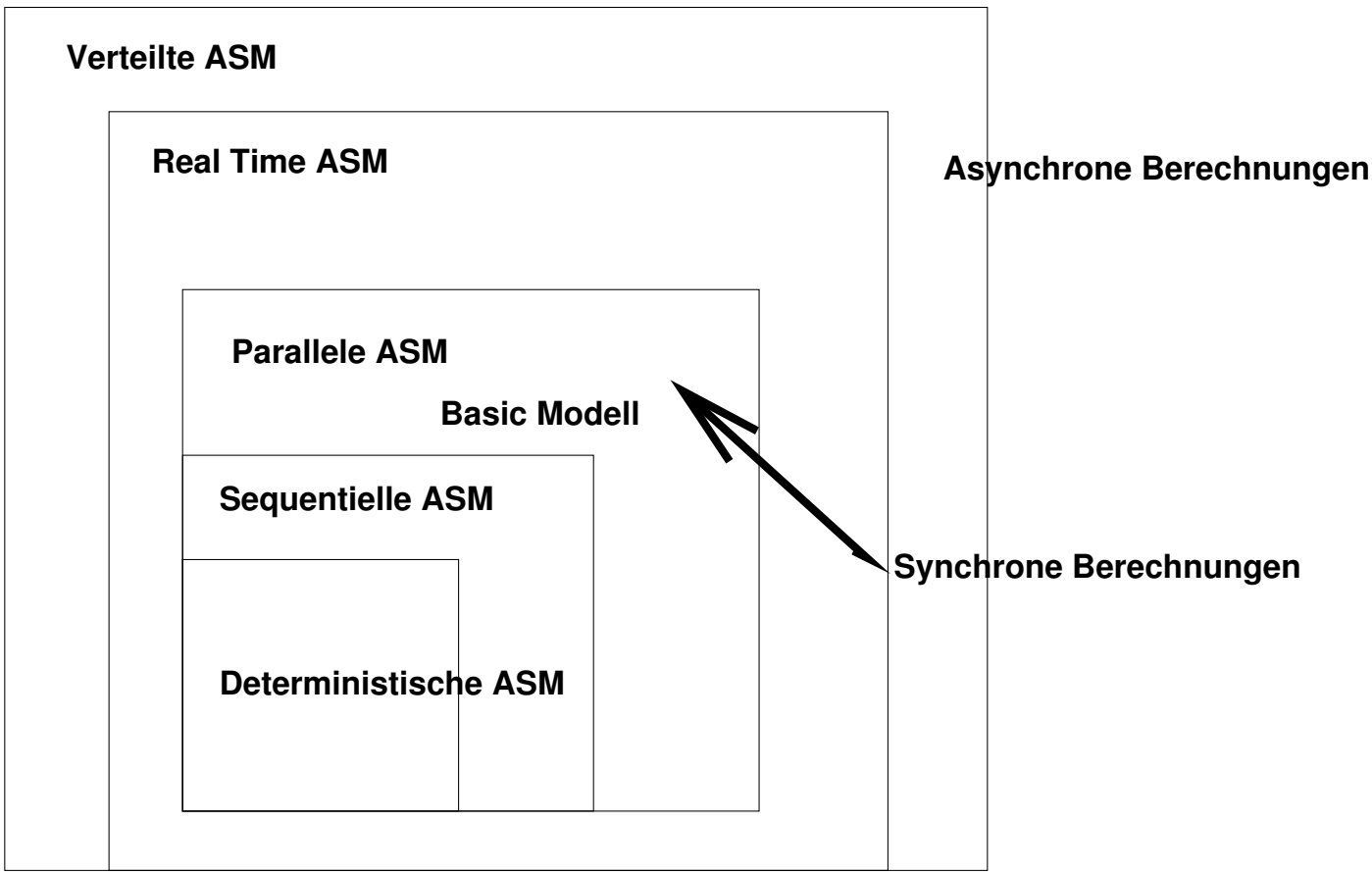
Problem: Simulation nicht einfach, da oft verschiedene Abstraktionsstufen und unterschiedliche Granularität.

Frage: Kann man TM so verallgemeinern, dass jeder Algorithmus, unabhängig von seiner Abstraktionsstufe, von einer verallgemeinerten Maschine natürlich und treu simuliert werden kann.
Wie sehen die Zustände und Instruktionen solcher Maschinen aus?

Einfach: If Bedingung Then Aktion

ASM These

ASM Thesis The concept of abstract state machine provides a universal computation model with the ability to simulate arbitrary algorithms on their natural levels of abstraction. Yuri Gurevich



Sequentielle ASM These

- Das Modell der sequentiellen ASM's ist universell für alle sequentiellen Algorithmen.
- Jeder sequentielle Algorithmus kann, unabhängig von seiner Abstraktionsstufe, schrittweise von einer sequentiellen ASM simuliert werden.

Um diese These zu untermauern benötigt man eine Präzisierung der sequentiellen Algorithmen und der sequentiellen ASM's.

$\rightsquigarrow$ Postulate für Sequentialität

Sequentialitäts-Postulate

- Sequentielle Zeit:
  Berechnungen sind linear geordnet.
- Abstrakte Zustände:
  Jede Art von statischer mathematischer Realität kann durch eine Struktur der Logik erster Stufe dargestellt werden. (Tarski)
- Bounded Exploration:
  Jeder Berechnungsschritt hängt nur von einer endlichen (vom Algorithmus abhängigen) beschränkten Zustandsinformation ab.

Y. Gurevich Sequential Abstract State Machines Capture Sequential Algorithms, ACM Transactions on Computational Logic, 1, 2000, 77-111.

Sequentielle Zeit

Sei A ein sequentieller Algorithmus. Zu A gehören:

- Eine Menge (Zustandsmenge) $S(A)$ von Zuständen von A.
- Eine Teilmenge $I(A)$ von $S(A)$ deren Elemente initiale Zustände von A heißen.
- Eine Abbildung $\tau_A : S(A) \to S(A)$, die Einschrittfunktion von A.

Ein Lauf (oder eine Berechnung) von A ist eine endliche oder unendliche Folge von Zustäden von A

$$X_0, X_1, X_2, \ldots$$

wobei X_0 ein initialer Zustand ist und $\tau_A(X_i) = X_{i+1}$ für alle i gilt.

Logische Zeit und nicht physikalische Zeit.

Abstrakte Zustände

Definition 3.1. *Algorithmen A and B sind äquivalent falls $S(A) = S(B)$, $I(A) = I(B)$ und $\tau_A = \tau_B$. Insbesondere haben äquivalente Algorithmen die gleichen Läufe.*

Sei A ein sequentieller Algorithmus:

- Zustände von A sind Strukturen erster Stufe.
- Alle Zustände von A haben gleiches Vokabular (Signatur).
- Die Einschrittfunktion verändert nicht die Basismenge $B(X)$ eines Zustands.
- $S(A)$ und $I(A)$ sind abgeschlossen gegen Isomorphismen und jeder Isomorphismus von Zustand X auf Zustand Y ist auch ein Isomorphismus von Zustand $\tau_A(X)$ auf $\tau_A(Y)$.

Übungen

Zustände: Signaturen, Interpretationen, Terme, Grundterme, Wert ...
Signaturen (Vokabular): Funktionen- und Relationennamen, Stelligkeit ($n \geq 0$)
Annahme: *true*, *false*, *undef* (Konstanten), *Boole* (einstellig) und $=$ sind in jeder Signatur enthalten. Die Interpretation von *true* ist stets verschieden von der von *false*, *undef*. Relationen werden als Funktionen mit Wert in den Interpretationen von *true*, *false* betrachtet. Einstellige Relationen werden als Teilmengen der Basismenge der Interpretation gesehen. Für ein Grundterm t im Vokabular sei $Val(t, X)$ sein Wert im Zustand X. Funktionen werden in dynamischen und statischen aufgeteilt, je nachdem ihre Interpretation sich bei Zustandsübergängen ändern oder nicht ändern können.
Übung: Modelliere Zustände von TM als abstrakten Zustand.
Modelliere Zustand des standard Euklidischen Algorithmus.

Bounded Exploration

- Parallelität Betrachte den folgenden Graph- Erreichbarkeits-Algorithmus der folgenden Schritt iteriert. Es wird angenommen, dass am Anfang nur ein Knoten die unäre Relation R erfüllt.

  do for all x, y with $Edge(x, y) \wedge R(x) \wedge \neg R(y)$ $\quad R(y) := true$

  In jedem Berechnungsschritt wird eine unbeschränkte Anzahl von lokalen Änderungen eines globalen Zustands durchgeführt.
- Unbeschränkte Schrittinformation
  Test auf isolierte Knoten in Graphen:

  if $\forall x \exists y\ Edge(x, y)$ then Output := false else Output := true

  In einem Schritt werden nur beschränkt viele lokale Veränderungen gemacht, es wird jedoch ein unbeschränkter Teil des Zustands in einem Schritt berücksichtigt.
  Wie erfasst man diese Eigenschaften ?$\rightsquigarrow$ Atomare Aktionen

Update Mengen

Betrachte die Struktur X als Speicher: Ist f ein j-stelliger Funktionsname und $\overline{a}$ ein j-Tupel von Basiselementen von X, dann heißt das Paar $(f, \overline{a})$ eine Lokation und $Content_X(f, \overline{a})$ ist der Wert der Interpretation von f für $\overline{a}$ in X.
Ist $(f, \overline{a})$ eine Lokation von X und b ein Element von X, so heißt $(f, \overline{a}, b)$ ein Update von X. Das Update ist trivial falls $b = Content_X(f, \overline{a})$. Um ein Update auszuführen wird der aktuelle Inhalt der Lokation durch b ersetzt. Eine Menge von Updates von X ist Konsistent falls es kein Paar von Updates mit gleicher Lokation und ungleiche Werte in der Menge gibt. Eine Menge Δ von Updates wird ausgeführt in dem man alle Updates in der Menge simultan ausführt falls die Menge konsistent ist, ansonsten wird nichts gemacht. Das Ergebnis wird mit $X + \Delta$ bezeichnet.

Lemma 3.2. *Sind X, Y Strukturen über die gleiche Signatur und mit gleicher Basismenge, dann gibt es eine eindeutige konsistente Menge Δ von nichttrivialen Updates von X mit $Y = X + \Delta$. Sei $\Delta \leftrightharpoons Y - X$.*

Update Mengen von Algorithmen, Erreichbare Elemente

Definition 3.3. *Sei X ein Zustand von Algorithmus A. Nach Definition haben X und $\tau_A(X)$ gleiche Signatur und Basismenge. Setze:*

$$\Delta(A, X) \leftrightharpoons \tau_A(X) - X \quad d.h.\ \tau_A(X) = X + \Delta(A, X)$$

Wie kann man Elemente der Basismenge eines Zustands in der Beschreibung des Algorithmus überhaupt ansprechen?
$\rightsquigarrow$ Über Grundterme der Signatur.

Definition 3.4. *Ein Element a einer Struktur X ist erreichbar wenn $a = Val(t, X)$ für ein Grundterm t im Vokabular von X.*
Eine Lokation $(f, \bar{a})$ von X ist erreichbar wenn jedes Element im Tupel $\bar{a}$ erreichbar ist.
Ein Update $(f, \bar{a}, b)$ von X ist erreichbar wenn $(f, \bar{a})$ und b erreichbar sind.

Bounded Exploration Postulat

Zwei Strukturen X und Y mit gleichem Vokabular Sig stimmen auf einer Menge T von Sig- Termen überein, falls $Val(t, X) = Val(t, Y)$ für alle $t \in T$. Das Vokabular (Signatur) eines Algorithmus ist das Vokabular seiner Zustände.

Sei A ein sequentieller Algorithmus.

- Es gibt eine endliche Menge T von Termen im Vokabular von A, so dass gilt $\Delta(A, X) = \Delta(A, Y)$ für Zustände X, Y von A die auf T übereinstimmen.

Intuition: Algorithmus A untersucht nur den Teil eines Zustands der erreichbar ist über die Termmenge T. Stimmen zwei Zustände auf dieser Termmenge überein, so sind die Update Mengen des Algorithmus für beide Zustände gleich.

Die Menge T ist ein bounded-exploration Zeuge für A.

Beispiel

Betrachte Algorithmus A:

$$\text{if P(f) then f} := \text{S(f)}$$

Zustände mit Interpretationen mit Basismenge $\mathbb{N}$, von P Teilmenge der natürlichen Zahlen, für S die Nachfolgerfunktion und f eine Konstante. Offenbar erfüllt A die Postulate Sequentielle Zeit und Abstrakte Zustände. Man könnte meinen, dass $T_0 = \{f, P(f), S(f)\}$ ein bounded-exploration Zeuge für A ist.
Sei X der kanonischer Zustand von A mit $\text{f} = 0$ und $\text{P}(0)$ gelte.
Setze $a \leftrightharpoons Val(true, X)$ und $b \leftrightharpoons Val(false, X)$, so dass $Val(P(0), X) = Val(true, X) = a$.
Sei Y der Zustand den man aus X erhält durch Reinterpretation von *true* als b und *false* als a, d.h. $Val(true, Y) = b$ und $Val(false, Y) = a$. Der Wert von $P(0)$ wurde nicht verändert:
$Val(P(0), Y) = a$, also $P(0)$ gilt nicht in Y . Somit stimmen X, Y über T_0 überein aber $\Delta(A, X) \neq \emptyset = \Delta(A, Y)$.
Die Menge $T = T_0 \cup \{true\}$ ist ein bounded-exploration Zeuge für A.

Sequentielle Algorithmen

Definition 3.5. *Ein sequentieller Algorithmus ist ein Objekt A welches die drei Postulate erfüllt.*
Insbesondere hat A ein Vokabular und einen bounded-exploration Zeugen T. O.B.d.A ist T Subterm-Abgeschlossen und enthält true, false, undef. Die Terme aus T heißen kritisch und ihre Interpretationen in einem Zustand X heißen kritische Werte in X.

Lemma 3.6. *Ist $(f, a_1, ..., a_j, a_0)$ ein Update in $\Delta(A, X)$, dann sind alle Elemente $a_0, a_1, ..., a_j$ kritische Werte in X.*

Beweis: Übung.
Die Menge der kritischen Terme hängt nicht von X ab, somit gibt es eine feste obere Schranke für die Größe von $\Delta(A, X)$ und A verändert in jeden Schritt somit eine beschränkte Anzahl von Lokationen. Jedes einzelne Update in $\Delta(A, X)$ ist eine atomare Aktion von A. D.h. $\Delta(A, X)$ besteht aus einer beschränkten Menge atomarer Aktionen von A.

Update Regeln

Definition 3.7. *Eine Update Regel über die Signatur Sig hat die Form*

$$f(t_1, ..., t_j) := t_0$$

wobei f Funktion und die t_i Terme in Sig sind. Um die Regel in der Sig-Struktur X zu feuern berechne die Werte $a_i = Val(t_i, X)$ und führe Update $((f, a_1, ..., a_j), a_0)$ über X aus.
Parallele Update Regel über Sig: Seien R_i Update Regeln über Sig, dann

par
  R_1
  R_2
  .
  .
  .
  R_k
endpar

Bezeichnung: Block (falls leer skip)

feuert durch gleichzeitiges feuern der R_i.

Sequentielle ASM-Programme

Definition 3.8. ***Semantik von Updateregeln.*** *Ist R eine Update Regel $f(t_1, ..., t_j) := t_0$ und $a_i = Val(t_i, X)$ dann setze*

$$\Delta(R, X) \leftrightharpoons \{(f, (a_1, ..., a_j), a_0)\}$$

Ist R eine par-Update Regel mit Bestandteilen $R_1, ...R_k$ so setze

$$\Delta(R, X) \leftrightharpoons \Delta(R1, X) \cup \cdots \cup \Delta(Rk, X).$$

Folgerung 3.9. *Insbesondere gibt es für jeden Zustand X eine Regel R^X die nur kritische Terme verwendet mit* $\Delta(R^X, X) = \Delta(A, X)$.

Beachte: Stimmen X, Y auf den kritischen Termen überein, so gilt $\Delta(R^X, Y) = \Delta(A, Y)$. Sind X, Y Zustände und $\Delta(R^X, Z) = \Delta(A, Z)$ für Zustand Z der isomorph zu Y ist, dann auch $\Delta(R^X, Y) = \Delta(A, Y)$. Betrachte Äquivalenzrelation $E_X(t1, t2) \leftrightharpoons Val(t1, X) = Val(t2, X)$ auf T. X, Y sind T-ähnlich, falls $E_X = E_Y \rightsquigarrow \Delta(R^X, Y) = \Delta(A, Y)$.Übung

Sequentielle ASM-Programme

Definition 3.10. *Ist φ ein Boolescher Term über Sig und R_1, R_2 Regeln über Sig dann ist*

if φ then R_1
else $R2$
endif *eine Regel*

Zur Feuerung der Regel im Zustand X werte φ in X aus. Ist das Ergebnis true, so $\Delta(R, X) = \Delta(R_1, X)$ sonst $\Delta(R, X) = \Delta(R_2, X)$.

Definition 3.11. *Ein sequentielles ASM Programm Π über die Signatur Sig ist eine Regel über Sig. Dementsprechend ist $\Delta(\Pi, X)$ wohldefiniert für jede Sig-Struktur X. Setze $\tau_\Pi(X) \leftrightharpoons X + \Delta(\Pi, X)$.*

Lemma 3.12. ***Hauptergebnis:*** *Für jeden sequentiellen Algorithmus A über Sig gibt es ein sequentielles ASM-programm Π über Sig mit $\Delta(\Pi, X) = \Delta(A, X)$ für alle Zustände X von A.*

Sequentielle ASM-Maschinen

Definition 3.13. *Eine sequentielle abstrakte Zustandsmaschine (seq-ASM) B über die Signatur Σ ist gegeben durch:*

- *Ein sequentielles ASM-Programm Π über Σ.*
- *Eine Menge $S(B)$ von Interpretationen von Σ die abgeschlossen ist gegen Isomorphismen und der Abbildung τ_Π .*
- *Eine Teilmenge $I(B) \subset S(B)$ die abgeschlossen gegen isomorphismen ist.*
- *Eine Abbildung τ_B, die Einschränkung von τ_Π auf $S(B)$.*

Satz 3.14. *Zu jedem sequentiellen Algorithmus A gibt es eine äquivalente sequentielle ASM.*

Beispiel

Beispiel 3.15. *Maximale Interval-Summe.[Gries 1990]. Sei A eine Function von* $\{0, 1, ..., n-1\} \to \mathbb{R}$ *und* $i, j, k \in \{0, 1, ..., n\}$. *Sei für* $i \leq j$ *beliebig* $S(i,j) \rightleftharpoons \sum_{i \leq k < j} A(k)$. *Insbesondere* $S(i,i) = 0$.
Problem: *Berechne* $S \rightleftharpoons max_{i \leq j} S(i,j)$.

Definiere $y(k) \rightleftharpoons max_{i \leq j \leq k} S(i,j)$. Dann $y(0) = 0, y(n) = S$ und

$$y(k+1) = max\{max_{i \leq j \leq k} S(i,j), max_{i \leq k+1} S(i, k+1)\} = max\{y(k), x(k+1)\}$$

wobei $x(k) \rightleftharpoons max_{i \leq k} S(i,k)$, also $x(0) = 0$ und

$$\begin{aligned} x(k+1) &= max\{max_{i \leq k} S(i, k+1), S(k+1, k+1)\} \\ &= max\{max_{i \leq k}(S(i,k) + A(k)), 0\} \\ &= max\{(max_{i \leq k} S(i,k)) + A(k), 0\} \\ &= max\{x(k) + A(k), 0\} \end{aligned}$$

Fortsetzung Beispiel

Wegen $y(k) \geq 0$, gilt

$$y(k+1) = max\{y(k), x(k+1)\} = max\{y(k), x(k) + A(k)\}$$

Annahme: Die nullstelligen dynamischen Funktionen k, x, y seien 0 im Anfangszustand. Der gewünschte Algorithmus ist dann

if $k \neq n$ *then*
  par
    $x := max\{x + A(k), 0\}$
    $y := max\{y, x + A(k)\}$
    $k := k + 1$
else $S := y$

Übung 3.16. *Definiere ASM die Markov's Normal-Algorithmen realisieren.*
Z.B. für $ab \rightarrow A,\ ba \rightarrow B,\ c \rightarrow C$

Simulation

Detailed definition of ASMs

- Part 1: Abstract states and update sets
- Part 2: Mathematical Logic
- Part 3: Transition rules and runs of ASMs
- Part 4: The reserve of ASMs

Part 1

Abstract states and update sets

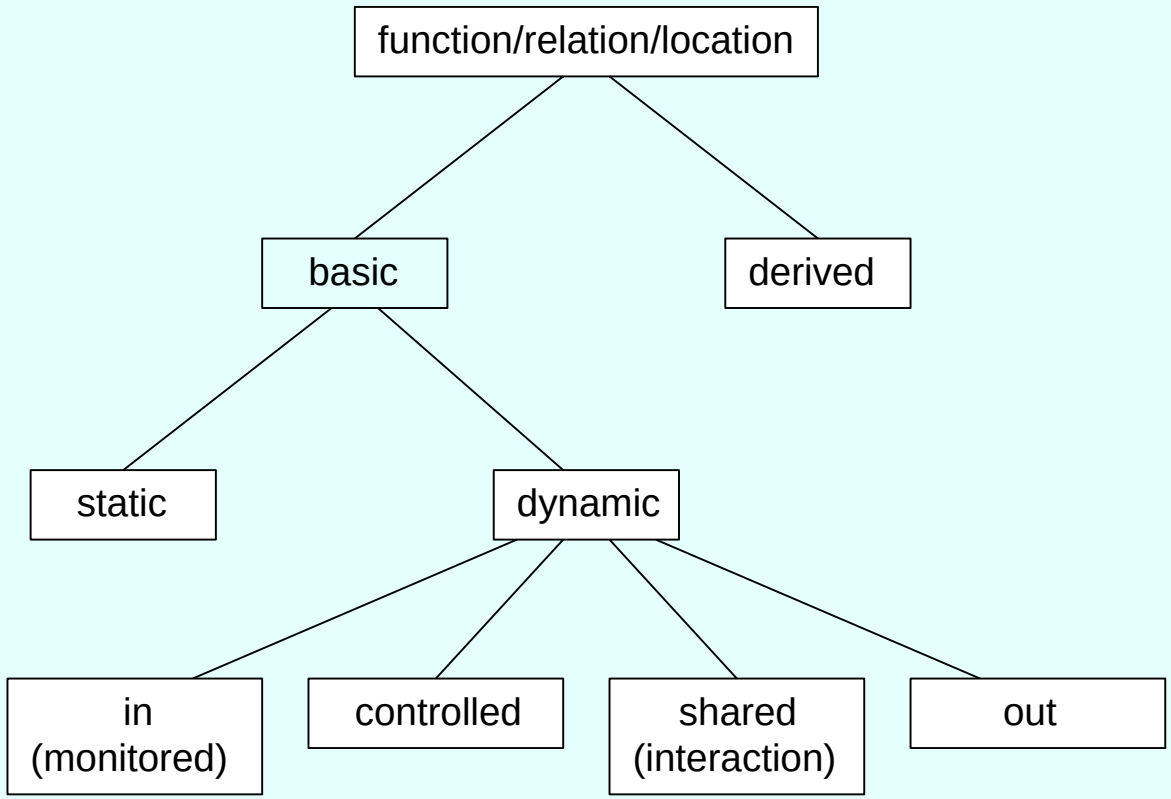
Signatures

Definition. A *signature* Σ is a finite collection of function names.

- Each function name f has an *arity*, a non-negative integer.
- Nullary function names are called *constants*.
- Function names can be *static* or *dynamic*.
- Every ASM signature contains the static constants $undef$, $true$, $false$.

Signatures are also called *vocabularies*.

Classification of functions



States

Definition. A *state* $\mathfrak{A}$ for the signature Σ is a non-empty set X, the *superuniverse* of $\mathfrak{A}$, together with an *interpretation* $f^{\mathfrak{A}}$ of each function name f of Σ.

- If f is an n-ary function name of Σ, then $f^{\mathfrak{A}}\colon X^n \to X$.
- If c is a constant of Σ, then $c^{\mathfrak{A}} \in X$.
- The superuniverse X of the state $\mathfrak{A}$ is denoted by $|\mathfrak{A}|$.

- The superuniverse is also called the *base set* of the state.
- The *elements* of a state are the elements of the superuniverse.

States (continued)

- The interpretations of $undef$, $true$, $false$ are pairwise different.
- The constant $undef$ represents an undetermined object.
- The *domain* of an n-ary function name f in $\mathfrak{A}$ is the set of all n-tuples $(a_1, \ldots, a_n) \in |\mathfrak{A}|^n$ such that $f^{\mathfrak{A}}(a_1, \ldots, a_n) \neq undef^{\mathfrak{A}}$.
- A *relation* is a function that has the values $true$, $false$ or $undef$.
- We write $a \in R$ as an abbreviation for $R(a) = true$.
- The superuniverse can be divided into *subuniverses* represented by unary relations.

Locations

Definition. A *location* of $\mathfrak{A}$ is a pair

$$(f, (a_1, \ldots, a_n))$$

where f is an n-ary function name and $a_1, \ldots, a_n$ are elements of $\mathfrak{A}$.

- The value $f^{\mathfrak{A}}(a_1, \ldots, a_n)$ is the *content* of the location in $\mathfrak{A}$.
- The *elements* of the location are the elements of the set $\{a_1, \ldots, a_n\}$.
- We write $\mathfrak{A}(l)$ for the content of the location l in $\mathfrak{A}$.

Notation. If $l = (f, (a_1, \ldots, a_n))$ is a location of $\mathfrak{A}$ and α is a function defined on $|\mathfrak{A}|$, then $\alpha(l) = (f, (\alpha(a_1), \ldots, \alpha(a_n)))$.

Updates and update sets

Definition. An *update* for $\mathfrak{A}$ is a pair (l, v), where l is a location of $\mathfrak{A}$ and v is an element of $\mathfrak{A}$.

- The update is *trivial*, if $v = \mathfrak{A}(l)$.
- An *update set* is a set of updates.

Definition. An update set U is *consistent*, if it has no clashing updates, i.e., if for any location l and all elements v, w, if $(l, v) \in U$ and $(l, w) \in U$, then $v = w$.

Firing of updates

Definition. The result of *firing* a consistent update set U in a state $\mathfrak{A}$ is a new state $\mathfrak{A} + U$ with the same superuniverse as $\mathfrak{A}$ such that for every location l of $\mathfrak{A}$:

$$(\mathfrak{A} + U)(l) = \begin{cases} v, & \text{if } (l, v) \in U; \\ \mathfrak{A}(l), & \text{if there is no } v \text{ with } (l, v) \in U. \end{cases}$$

The state $\mathfrak{A} + U$ is called the *sequel* of $\mathfrak{A}$ with respect to U.

Homomorphisms and isomorphisms

Let $\mathfrak{A}$ and $\mathfrak{B}$ be two states over the same signature.

Definition. A *homomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a function α from $|\mathfrak{A}|$ into $|\mathfrak{B}|$ such that $\alpha(\mathfrak{A}(l)) = \mathfrak{B}(\alpha(l))$ for each location l of $\mathfrak{A}$.

Definition. An *isomorphism* from $\mathfrak{A}$ to $\mathfrak{B}$ is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ which is a ono-to-one function from $|\mathfrak{A}|$ onto $|\mathfrak{B}|$.

Lemma (Isomorphism). Let α be an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. If U is a consistent update set for $\mathfrak{A}$, then $\alpha(U)$ is a consistent update set for $\mathfrak{B}$ and α is an isomorphism from $\mathfrak{A}+U$ to $\mathfrak{B}+\alpha(U)$.

Composition of update sets

$$U \oplus V = V \cup \{(l, v) \in U \mid \text{there is no } w \text{ with } (l, w) \in V\}$$

Lemma. Let U, V, W be update sets.

- $(U \oplus V) \oplus W = U \oplus (V \oplus W)$
- If U and V are consistent, then $U \oplus V$ is consistent.
- If U and V are consistent, then $\mathfrak{A} + (U \oplus V) = (\mathfrak{A} + U) + V$.

Part 2

Mathematical Logic

Terms

Let Σ be a signature.

Definition. The *terms* of Σ are syntactic expressions generated as follows:

- Variables x, y, z, ... are terms.
- Constants c of Σ are terms.
- If f is an n-ary function name of Σ, $n > 0$, and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

- A term which does not contain variables is called a *ground term*.
- A term is called *static*, if it contains static function names only.
- By $t\frac{s}{x}$ we denote the result of replacing the variable x in term t everywhere by the term s (*substitution* of s for x in t).

Variable assignments

Let $\mathfrak{A}$ be a state.

Definition. A *variable assignment* for $\mathfrak{A}$ is a finite function ζ which assigns elements of $|\mathfrak{A}|$ to a finite number of variables.

- We write $\zeta[x \mapsto a]$ for the variable assignment which coincides with ζ except that it assigns the element a to the variable x:

$$\zeta[x \mapsto a](y) = \begin{cases} a, & \text{if } y = x; \\ \zeta(y), & \text{otherwise.} \end{cases}$$

- Variable assignments are also called *environments*.

Evaluation of terms

Definition. Let $\mathfrak{A}$ be a state of Σ.
Let ζ be a variable assignment for $\mathfrak{A}$.
Let t be a term of Σ such that all variables of t are defined in ζ.
The *value* $[\![t]\!]^{\mathfrak{A}}_{\zeta}$ is defined as follows:

- $[\![x]\!]^{\mathfrak{A}}_{\zeta} = \zeta(x)$
- $[\![c]\!]^{\mathfrak{A}}_{\zeta} = c^{\mathfrak{A}}$
- $[\![f(t_1, \ldots, t_n)]\!]^{\mathfrak{A}}_{\zeta} = f^{\mathfrak{A}}([\![t_1]\!]^{\mathfrak{A}}_{\zeta}, \ldots, [\![t_n]\!]^{\mathfrak{A}}_{\zeta})$

Evaluation of terms (continued)

Lemma (Coincidence). If ζ and η are two variable assignments for t such that $\zeta(x) = \eta(x)$ for all variables x of t, then $[\![t]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\eta}$.

Lemma (Homomorphism). If α is a homomorphism from $\mathfrak{A}$ to $\mathfrak{B}$, then $\alpha([\![t]\!]^{\mathfrak{A}}_{\zeta}) = [\![t]\!]^{\mathfrak{B}}_{\alpha\circ\zeta}$ for each term t.

Lemma (Substitution). Let $a = [\![s]\!]^{\mathfrak{A}}_{\zeta}$.
Then $[\![t\frac{s}{x}]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\zeta[x\mapsto a]}$.

Formulas

Let Σ be a signature.

Definition. The *formulas* of Σ are generated as follows:

- If s and t are terms of Σ, then $s = t$ is a formula.
- If φ is a formula, then $\neg\varphi$ is a formula.
- If φ and ψ are formulas, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$ and $(\varphi \rightarrow \psi)$ are formulas.
- If φ is a formula and x a variable, then $(\forall x\, \varphi)$ and $(\exists x\, \varphi)$ are formulas.

- A formula $s = t$ is called an *equation*.
- The expression $s \neq t$ is an abbreviation for $\neg(s = t)$.

Formulas (continued)

symbol	name	meaning
$\neg$	negation	not
$\wedge$	conjunction	and
$\vee$	disjunction	or (inclusive)
$\longrightarrow$	implication	if-then
$\forall$	universal quantification	for all
$\exists$	existential quantification	there is

Formulas (continued)

$\varphi \wedge \psi \wedge \chi$ stands for $((\varphi \wedge \psi) \wedge \chi)$,

$\varphi \vee \psi \vee \chi$ stands for $((\varphi \vee \psi) \vee \chi)$,

$\varphi \wedge \psi \rightarrow \chi$ stands for $((\varphi \wedge \psi) \rightarrow \chi)$, etc.

- The variable x is *bound* by the quantifier $\forall$ ($\exists$) in $\forall x\, \varphi$ ($\exists x\, \varphi$).
- The *scope* of x in $\forall x\, \varphi$ ($\exists x\, \varphi$) is the formula φ.
- A variable x occurs *free* in a formula, if it is not in the scope of a quantifier $\forall x$ or $\exists x$.
- By $\varphi\frac{t}{x}$ we denote the result of replacing all free occurrences of the variable x in φ by the term t. (Bound variables are renamed.)

Semantics of formulas

$$[\![s = t]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![s]\!]^{\mathfrak{A}}_{\zeta} = [\![t]\!]^{\mathfrak{A}}_{\zeta}; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\neg\varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = false; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \wedge \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = true \text{ and } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = true; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \vee \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = true \text{ or } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = true; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\varphi \rightarrow \psi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = false \text{ or } [\![\psi]\!]^{\mathfrak{A}}_{\zeta} = true; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\forall x\, \varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x \mapsto a]} = true \text{ for every } a \in |\mathfrak{A}|; \\ false, & \text{otherwise.} \end{cases}$$

$$[\![\exists x\, \varphi]\!]^{\mathfrak{A}}_{\zeta} = \begin{cases} true, & \text{if there exists an } a \in |\mathfrak{A}| \text{ with } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x \mapsto a]} = true; \\ false, & \text{otherwise.} \end{cases}$$

Coincidence, Substitution, Isomorphism

Lemma (Coincidence). If ζ and η are two variable assignments for φ such that $\zeta(x) = \eta(x)$ for all free variables x of φ, then $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{A}}_{\eta}$.

Lemma (Substitution). Let t be a term and $a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$. Then $[\![\varphi\frac{t}{x}]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x\mapsto a]}$.

Lemma (Isomorphism). Let α be an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$. Then $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = [\![\varphi]\!]^{\mathfrak{B}}_{\alpha\circ\zeta}$.

Models

Definition. A state $\mathfrak{A}$ is a *model* of φ (written $\mathfrak{A} \models \varphi$), if $[\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = true$ for all variable assignments ζ for φ.

Part 3

Transition rules and runs of ASMs

Transition rules

Skip Rule: **skip**

Meaning: Do nothing

Update Rule: $f(s_1, \ldots, s_n) := t$

Meaning: Update the value of f at $(s_1, \ldots, s_n)$ to t.

Block Rule: P **par** Q

Meaning: P and Q are executed in parallel.

Conditional Rule: **if** φ **then** P **else** Q

Meaning: If φ is true, then execute P, otherwise execute Q.

Let Rule: **let** $x = t$ **in** P

Meaning: Assign the value of t to x and then execute P.

Transition rules (continued)

Forall Rule: **forall** x **with** φ **do** P

Meaning: Execute P in parallel for each x satisfying φ.

Choose Rule: **choose** x **with** φ **do** P

Meaning: Choose an x satisfying φ and then execute P.

Sequence Rule: P **seq** Q

Meaning: P and Q are executed sequentially, first P and then Q.

Call Rule: $r(t_1, \ldots, t_n)$

Meaning: Call transition rule r with parameters $t_1, \ldots, t_n$.

Variations of the syntax

if φ **then**
P
else
Q
endif
	if φ **then** P **else** Q

[**do in-parallel**] P_1 $\vdots$ P_n [**enddo**]	P_1 **par** ... **par** P_n
$\{P_1, \ldots, P_n\}$	P_1 **par** ... **par** P_n

Variations of the syntax (continued)

do forall $x\colon\varphi$ P **enddo**	**forall** x **with** φ **do** P
choose $x\colon\varphi$ P **endchoose**	**choose** x **with** φ **do** P
step P **step** Q	P **seq** Q

Beispiel

Beispiel 3.17. *Sortieren linearer Datenstrukturen*
in-place, one-swap-a-time.
Sei $a : Index \rightarrow Value$

$$\begin{array}{l} choose \quad x, y \in Index : x < y \wedge a(x) > a(y) \\ \quad do \quad in - parallel \\ \qquad a(x) := a(y) \\ \qquad a(y) := a(x) \end{array}$$

Zwei Arten von Nichtdeterminismus:

"Don't-care" Nichtdeterminismus: Random choice

$$\begin{array}{l} choose \quad x \in \{x_1, x_2, ..., x_n\} \ with \ \ \varphi(x) \ \ do \\ \qquad\qquad R(x) \end{array}$$

"Don't-know" Indeterminismus

Extern kontrollierte Aktionen und Ereignisse (z.B. input Aktionen)

$$monitored \quad f : X \rightarrow Y$$

Free and bound variables

Definition. An occurrence of a variable x is *free* in a transition rule, if it is not in the scope of a **let** x, **forall** x or **choose** x.

let $x = t$ **in** P
scope of x

forall x **with** φ **do** P
scope of x

choose x **with** φ **do** P
scope of x

Rule declarations

Definition. A *rule declaration* for a rule name r of arity n is an expression

$$r(x_1, \ldots, x_n) = P$$

where

- P is a transition rule and
- the free variables of P are contained in the list $x_1, \ldots, x_n$.

Remark: Recursive rule declarations are allowed.

Abstract State Machines

Definition. An *abstract state machine* M consists of

- a signature Σ,
- a set of initial states for Σ,
- a set of rule declarations,
- a distinguished rule name of arity zero called the *main rule name* of the machine.

Semantics of transition rules

The semantics of transition rules is defined in a calculus by rules:

$$\frac{\mathit{Premise}_1 \ \cdots \ \mathit{Premise}_n}{\mathit{Conclusion}} \ \mathit{Condition}$$

The predicate

$$\mathsf{yields}(P, \mathfrak{A}, \zeta, U)$$

means:

The transition rule P yields the update set U in state $\mathfrak{A}$ under the variable assignment ζ.

Semantics of transition rules (continued)

$$\frac{}{\mathsf{yields}(\textbf{skip}, \mathfrak{A}, \zeta, \emptyset)}$$

$$\frac{}{\mathsf{yields}(f(s_1, \ldots, s_n) := t, \mathfrak{A}, \zeta, \{(l, v)\})} \quad \text{where } l = (f, ([\![s_1]\!]^{\mathfrak{A}}_{\zeta}, \ldots, [\![s_n]\!]^{\mathfrak{A}}_{\zeta})) \text{ and } v = [\![t]\!]^{\mathfrak{A}}_{\zeta}$$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta, U) \quad \mathsf{yields}(Q, \mathfrak{A}, \zeta, V)}{\mathsf{yields}(P \textbf{ par } Q, \mathfrak{A}, \zeta, U \cup V)}$$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta, U)}{\mathsf{yields}(\textbf{if } \varphi \textbf{ then } P \textbf{ else } Q, \mathfrak{A}, \zeta, U)} \quad \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{true}$$

$$\frac{\mathsf{yields}(Q, \mathfrak{A}, \zeta, V)}{\mathsf{yields}(\textbf{if } \varphi \textbf{ then } P \textbf{ else } Q, \mathfrak{A}, \zeta, V)} \quad \text{if } [\![\varphi]\!]^{\mathfrak{A}}_{\zeta} = \mathit{false}$$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\mathsf{yields}(\textbf{let } x = t \textbf{ in } P, \mathfrak{A}, \zeta, U)} \quad \text{where } a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$$

$$\frac{\mathsf{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U_a) \quad \text{for each } a \in I}{\mathsf{yields}(\textbf{forall } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, \bigcup_{a \in I} U_a)} \quad \text{where } I = \mathit{range}(x, \varphi, \mathfrak{A}, \zeta)$$

Semantics of transition rules (continued)

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\text{yields}(\textbf{choose } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, U)} \quad \text{if } a \in range(x, \varphi, \mathfrak{A}, \zeta)$$

$$\frac{}{\text{yields}(\textbf{choose } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, \emptyset)} \quad \text{if } range(x, \varphi, \mathfrak{A}, \zeta) = \emptyset$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A} + U, \zeta, V)}{\text{yields}(P \textbf{ seq } Q, \mathfrak{A}, \zeta, U \oplus V)} \quad \text{if } U \text{ is consistent}$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U)}{\text{yields}(P \textbf{ seq } Q, \mathfrak{A}, \zeta, U)} \quad \text{if } U \text{ is inconsistent}$$

$$\frac{\text{yields}(P\frac{t_1 \cdots t_n}{x_1 \cdots x_n}, \mathfrak{A}, \zeta, U)}{\text{yields}(r(t_1, \ldots, t_n), \mathfrak{A}, \zeta, U)} \quad \text{where } r(x_1, \ldots, x_n) = P \text{ is a rule declaration of } M$$

$$range(x, \varphi, \mathfrak{A}, \zeta) = \{a \in |\mathfrak{A}| : [\![\varphi]\!]^{\mathfrak{A}}_{\zeta[x \mapsto a]} = true\}$$

Coincidence, Substitution, Isomorphisms

Lemma (Coincidence). If $\zeta(x) = \eta(x)$ for all free variables x of a transition rule P and P yields U in $\mathfrak{A}$ under ζ, then P yields U in $\mathfrak{A}$ under η.

Lemma (Substitution). Let t be a static term and $a = [\![t]\!]^{\mathfrak{A}}_{\zeta}$. Then the rule $P\frac{t}{x}$ yields the update set U in state $\mathfrak{A}$ under ζ iff P yields U in $\mathfrak{A}$ under $\zeta[x \mapsto a]$.

Lemma (Isomorphism). If α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{B}$ and P yields U in $\mathfrak{A}$ under ζ, then P yields $\alpha(U)$ in $\mathfrak{B}$ under $\alpha \circ \zeta$.

Move of an ASM

Definition. A machine M can make a *move* from state $\mathfrak{A}$ to $\mathfrak{B}$ (written $\mathfrak{A} \overset{M}{\Longrightarrow} \mathfrak{B}$), if the main rule of M yields a consistent update set U in state $\mathfrak{A}$ and $\mathfrak{B} = \mathfrak{A} + U$.

- The updates in U are called *internal updates*.
- $\mathfrak{B}$ is called the *next internal state*.

If α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{A}'$, the following diagram commutes:

$$\begin{array}{ccc} \mathfrak{A} & \overset{M}{\Longrightarrow} & \mathfrak{B} \\ \alpha \downarrow & & \downarrow \alpha \\ \mathfrak{A}' & \overset{M}{\Longrightarrow} & \mathfrak{B}' \end{array}$$

Run of an ASM

Let M be an ASM with signature Σ.

A *run* of M is a finite or infinite sequence $\mathfrak{A}_0, \mathfrak{A}_1, \ldots$ of states for Σ such that

- $\mathfrak{A}_0$ is an initial state of M
- for each n,
  - either M can make a move from $\mathfrak{A}_n$ into the next internal state $\mathfrak{A}'_n$ and the environment produces a consistent set of external or shared updates U such that $\mathfrak{A}_{n+1} = \mathfrak{A}'_n + U$,
  - or M cannot make a move in state $\mathfrak{A}_n$ and $\mathfrak{A}_n$ is the last state in the run.

- In *internal* runs, the environment makes no moves.
- In *interactive* runs, the environment produces updates.

Beispiel

Beispiel 3.18. *Minimaler aufspannender Baum* ***Prim's Algorithmus***

Zwei getrennte Phasen: initial, run

Signatur: Gewichteter Graph (zush, ohne loops) gegeben durch Mengen NODE, EDGE, . . . Funktionen
$weight : EDGE \to REAL, frontier : EDGE \to Bool, tree : EDGE \to Bool$

```
if mode = initial then
    choose p : NODE
        Selected(p) := true
        forall e : EDGE : p ∈ endpoints(e)
            frontier(e) := true
    mode := run
```

Beispiel: Prim's Algorithmus (Fort.)

$$
\begin{array}{l}
\textit{if}\ \ \textit{mode} = \textit{run}\ \ \textit{then} \\
\quad \textit{choose}\ \ e : \textit{EDGE} : \textit{frontier}(e) \wedge \\
\qquad ((\forall f \in \textit{EDGE}) :\ \textit{frontier}(f) \Rightarrow\ \ \textit{weight}(f) \geq \textit{weight}(e)) \\
\quad\ \ \textit{tree}(e) := \textit{true} \\
\quad\ \ \textit{choose}\ \ p :\ \ \textit{NODE} : p \in \textit{endpoints}(e) \wedge \neg \textit{Selected}(p) \\
\quad\quad \textit{Selected}(p) := \textit{true} \\
\quad\quad \textit{forall}\ \ f : \textit{EDGE} : p \in \textit{endpoints}(f) \\
\quad\quad\ \ \textit{frontier}(f) := \neg \textit{frontier}(f) \\
\quad \textit{ifnone}\ \ \textit{mode} := \textit{done}
\end{array}
$$

Wie zeigt man Korrektheit, Terminierung?

Übung 3.19. *Man gebe eine ASM-Maschine an die Kruskal's Algorithmus realisiert.*

Part 4

The reserve of ASMs

Importing new elements from the reserve

Import rule: **import** x **do** P

Meaning: Choose an element x from the reserve, delete it from the reserve and execute P.

let $x = new(X)$ **in** P abbreviates

import x **do**
$X(x) := true$
P

The reserve of a state

- New dynamic relation $Reserve$.
- $Reserve$ is updated by the system, not by rules.
- $Res(\mathfrak{A}) = \{a \in |\mathfrak{A}| : Reserve^{\mathfrak{A}}(a) = true\}$
- The reserve elements of a state are not allowed to be in the domain and range of any basic function of the state.

Definition. A state $\mathfrak{A}$ satisfies the *reserve condition* with respect to an environment ζ, if the following two conditions hold for each element $a \in Res(\mathfrak{A}) \setminus ran(\zeta)$:

- The element a is not the content of a location of $\mathfrak{A}$.
- If a is an element of a location l of $\mathfrak{A}$ which is not a location for $Reserve$, then the content of l in $\mathfrak{A}$ is $undef$.

Semantics of ASMs with a reserve

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U)}{\text{yields}(\textbf{import } x \textbf{ do } P, \mathfrak{A}, \zeta, V)} \quad \text{if } a \in Res(\mathfrak{A}) \setminus ran(\zeta) \text{ and } V = U \cup \{((Reserve, a), false)\}$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A}, \zeta, V)}{\text{yields}(P \textbf{ par } Q, \mathfrak{A}, \zeta, U \cup V)} \quad \text{if } Res(\mathfrak{A}) \cap El(U) \cap El(V) \subseteq ran(\zeta)$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta[x \mapsto a], U_a) \quad \text{for each } a \in I}{\text{yields}(\textbf{forall } x \textbf{ with } \varphi \textbf{ do } P, \mathfrak{A}, \zeta, \bigcup_{a \in I} U_a)} \quad \text{if } I = range(x, \varphi, \mathfrak{A}, \zeta) \text{ and for } a \neq b \; Res(\mathfrak{A}) \cap El(U_a) \cap El(U_b) \subseteq ran(\zeta)$$

- $El(U)$ is the set of elements that occur in the updates of U.
- The elements of an update (l, v) are the value v and the elements of the location l.

Problem

Problem 1: New elements that are imported in parallel must be different.

import x **do** $parent(x) = root$
import y **do** $parent(y) = root$

Problem 2: Hiding of bound variables.

import x **do**
  $f(x) := 0$
  let $x = 1$ **in**
    import y **do** $f(y) := x$

Syntactic constraint. In the scope of a bound variable the same variable should not be used again as a bound variable (**let**, **forall**, **choose**, **import**).

Preservation of the reserve condition

Lemma (Preservation of the reserve condition).
If a state $\mathfrak{A}$ satisfies the reserve condition wrt. ζ and P yields a consistent update set U in $\mathfrak{A}$ under ζ, then

- the sequel $\mathfrak{A} + U$ satisfies the reserve condition wrt. ζ,
- $Res(\mathfrak{A} + U) \setminus ran(\zeta)$ is contained in $Res(\mathfrak{A}) \setminus El(U)$.

Permutation of the reserve

Lemma (Permutation of the reserve). Let $\mathfrak{A}$ be a state that satisfies the reserve condition wrt. ζ. If α is a function from $|\mathfrak{A}|$ to $|\mathfrak{A}|$ that permutes the elements in $Res(\mathfrak{A}) \setminus ran(\zeta)$ and is the identity on non-reserve elements of $\mathfrak{A}$ and on elements in the range of ζ, then α is an isomorphism from $\mathfrak{A}$ to $\mathfrak{A}$.

Independence of the choice of reserve elements

Lemma (Independence).
Let P be a rule of an ASM without **choose**. If

- $\mathfrak{A}$ satisfies the reserve condition wrt. ζ,
- the bound variables of P are not in the domain of ζ,
- P yields U in $\mathfrak{A}$ under ζ,
- P yields U' in $\mathfrak{A}$ under ζ,

then there exists a permutation α of $Res(\mathfrak{A}) \setminus ran(\zeta)$ such that $\alpha(U) = U'$.

Beispiel: Abstrakte Datentypen (ADT)

Beispiel 3.20. *Doppelt verkettete Listen*

ASM-Beispiele

Übung 3.21. *Gebe eine ASM-Spezifikation für die Datenstruktur Beschränkter Keller.*

Verteilte ASM: Concurrency, reactivity, time

Verteilte ASM (DASM) Berechnungsmodell:

Asynchrones Berechnungsmodell
Autonom operierende Agenten

- Eine endliche Menge autonomer ASM-Agenten, jeder mit eigenem Programm.
- Agenten interagieren durch lesen und schreiben von gemeinsamen Lokationen globaler Maschinenzustände.
- Potentielle Konflikte werden durch das zugrundeliegende semantische Modell aufgelöst, nach Maßgabe der Definition partiell geordneter Läufe.

Grundlagen:Ordnungen, CPO's, Beweistechniken

Eigenschaften binärer Relationen

- X Menge
- $\rho \subseteq X \times X$ binäre Relation
- Eigenschaften

(P1)	$x \rho x$	(reflexiv)
(P2)	$(x \rho y \wedge y \rho x) \rightarrow x = y$	(antisymmetrisch)
(P3)	$(x \rho y \wedge y \rho z) \rightarrow x \rho z$	(transitiv)
(P4)	$(x \rho y \vee y \rho x)$	(linear)

Quasi-Ordnungen

- $\lesssim \subseteq X \times X$ Quasi-Ordnung gdw $\lesssim$ reflexiv und transitiv.
- Kern:
$$\approx = \lesssim \cap \lesssim^{-1}$$
- Strikter Anteil : $< = \lesssim \setminus \approx$
- $Y \subseteq X$ links geschlossen (bzgl. $\lesssim$) gdw
$$(\forall y \in Y : (\forall x \in X : x \lesssim y \rightarrow x \in Y))$$
- Schreibweise: Quasi-Ordnung $(X, \lesssim)$

Partial-Ordnungen

- $\leq \subseteq X \times X$ Partial-Ordnung gdw $\leq$ reflexiv, antisymmetrisch und transitiv.
- Kern: Es gilt
$$\mathrm{id}_X = \leq \cap \leq^{-1}$$
- Strikter Anteil: $<= \leq \setminus \mathrm{id}_X$
- Oft: $<$ Partial-Ordnung gdw $<$ irreflexiv, transitiv.
- Schreibweise: Partial-Ordnung $(X, \leq)$

Wohlfundierte Ordnungen

- Partial-Ordnung $\leq \subseteq X \times X$ wohlfundiert gdw

$$(\forall Y \subseteq X : Y \neq \emptyset \rightarrow (\exists y \in Y : y \text{ minimal in } Y \text{ bzgl. } \leq))$$

- Quasi-Ordnung $\lesssim$ wohlfundiert gdw strikter Anteil von $\lesssim$ wohlfundiert.
- Initiales Segment: $Y \subseteq X$, links-geschlossen
- Initiale Sektion von x: $\text{sec}(x) = \{y : y < x\}$

Supremum

- Sei $(X, \leq)$ Partial-Ordnung und $Y \subseteq X$
- $S \subseteq X$ Kette gdw Elemente von S werden linear geordnet durch $\leq$.
- y obere Schranke von Y gdw

$$\forall y' \in Y : y' \leq y$$

- Supremum: y Supremum von Y gdw

$$\forall y' \in X : ((y' \text{ obere Schranke von } Y) \rightarrow y \leq y')$$

- Analog: untere Schranke, Infimum $\inf(Y)$

CPO

- Partial-Ordnung $(D, \sqsubseteq)$ complete partial ordering (CPO) gdw
  - $\exists$ kleinstes Element $\bot$ von D (bzgl. $\sqsubseteq$)
  - Jede Kette S hat ein Supremum $\sup(S)$.

Beispiel

Beispiel 4.1.

- $(\mathcal{P}(X), \subseteq)$ *ist CPO.*
- $(D, \sqsubseteq)$ *ist CPO mit*
  - $D = X \nrightarrow Y$*: Menge aller partieller Funktionen* f *mit* $\mathrm{dom}(f) \subseteq X$ *und* $\mathrm{cod}(f) \subseteq Y$*.*
  - *Sei* $f, g \in X \nrightarrow Y$*.*

$$f \sqsubseteq g \text{ gdw } \mathrm{dom}(f) \subseteq \mathrm{dom}(g) \wedge (\forall x \in \mathrm{dom}(f) : f(x) = g(x))$$

monoton, stetig

- $(D, \sqsubseteq)$, $(E, \sqsubseteq')$ CPOs
- $f : D \to E$ monoton gdw

$$(\forall d, d' \in D : d \sqsubseteq d' \to f(d) \sqsubseteq' f(d'))$$

- $f : D \to E$ stetig gdw f monoton und

$$(\forall S \subseteq D : S \text{ Kette } \to f(\sup(S)) = \sup(f(S)))$$

- $X \subseteq D$ zulässig gdw

$$(\forall S \subseteq X : S \text{ Kette } \to \sup(S) \in X)$$

Fixpunkt

- $(D, \sqsubseteq)$ CPO, $f : D \to D$
- $d \in D$ Fixpunkt von f gdw

$$f(d) = d$$

- $d \in D$ kleinster Fixpunkt von f gdw d Fixpunkt von f und

$$(\forall d' \in D : d' \text{ Fixpunkt } \to d \sqsubseteq d')$$

Fixpunkt-Satz

Fixpunkt-Satz: $(D, \sqsubseteq)$ CPO, $f : D \to D$ stetig, dann hat f einen kleinsten Fixpunkt μf. Es gilt

$$\mu f = \sup\{f^i(\bot) : i \in \mathbb{N}\}$$

Beweis: (Skizze)

- $\sup\{f^i(\bot) : i \in \mathbb{N}\}$ Fixpunkt:

$$\begin{aligned} f(\sup\{f^i(\bot) : i \in \mathbb{N}\}) &= \sup\{f^{i+1}(\bot) : i \in \mathbb{N}\} \\ &\quad \text{(stetig)} \\ &= \sup\{\sup\{f^{i+1}(\bot) : i \in \mathbb{N}\}, \bot\} \\ &= \sup\{f^i(\bot) : i \in \mathbb{N}\} \end{aligned}$$

Fixpunkt-Satz

Fixpunkt-Satz: $(D, \sqsubseteq)$ CPO, $f : D \to D$ stetig, dann hat f einen kleinsten Fixpunkt μf. Es gilt

$$\mu f = \sup\{f^i(\bot) : i \in \mathbb{N}\}$$

Beweis: (Fortsetzung)

- $\sup\{f^i(\bot) : i \in \mathbb{N}\}$ kleinster Fixpunkt:
  1. d' Fixpunkt von f
  2. $\bot \sqsubseteq d'$
  3. f monoton, d' FP: $f(\bot) \sqsubseteq f(d') = d'$
  4. Induktion: $\forall i \in \mathbb{N} : f^i(\bot) \sqsubseteq f^i(d') = d'$
  5. $\sup\{f^i(\bot) : i \in \mathbb{N}\} \sqsubseteq d'$

Induktion über $\mathbb{N}$

Induktionsprinzip:

$$(\forall X \subseteq \mathbb{N} : ((0 \in X \wedge (\forall x \in X : x \in X \rightarrow x + 1 \in X))) \rightarrow X = \mathbb{N})$$

Korrektheit:

1. Ann. nein, also $\exists X \subseteq \mathbb{N} : \mathbb{N} \setminus X \neq \emptyset$
2. Sei y minimal in $\mathbb{N} \setminus X$ (bzgl. $<$).
3. $y \neq 0$
4. $y - 1 \in X \wedge y \notin X$
5. Widerspruch

Induktion über $\mathbb{N}$ (Alternative)

Induktionsprinzip:

$$(\forall X \subseteq \mathbb{N} : (\forall x \in \mathbb{N} : \mathsf{sec}(x) \subseteq X \rightarrow x \in X) \rightarrow X = \mathbb{N})$$

Korrektheit:

1. Ann. nein, also $\exists X \subseteq \mathbb{N} : \mathbb{N} \setminus X \neq \emptyset$
2. Sei y minimal in $\mathbb{N} \setminus X$ (bzgl. $<$).
3. $\mathsf{sec}(y) \subseteq X, y \notin X$
4. Widerspruch

Wohlfundierte Induktion

Induktionsprinzip: Sei $(Z, \leq)$ wohlfundierte Partial-Ordnung.

$$(\forall X \subseteq Z : (\forall x \in Z : \mathrm{sec}(x) \subseteq X \rightarrow x \in X) \rightarrow X = Z)$$

Korrektheit:

1. Ann. nein, also $Z \setminus X \neq \emptyset$
2. Sei z minimal in $Z \setminus X$ (bzgl. $\leq$).
3. $\mathrm{sec}(z) \subseteq X, z \notin X$
4. Widerspruch

FP-Induktion

Induktionsprinzip: Sei $(D, \sqsubseteq)$ CPO, $f : D \to D$ stetig.

$$(\forall X \subseteq D, \text{zulässig} : (\bot \in X \wedge (\forall y : y \in X \to f(y) \in X)) \to \mu f \in X)$$

Korrektheit: Sei $X \subseteq D$ zulässig.

$$\begin{array}{lcll}
\mu f \in X & \Leftrightarrow & \sup\{f^i(\bot) : i \in \mathbb{N}\} \in X & \text{(FP-Satz)} \\
& \Leftarrow & \forall i \in \mathbb{N} : f^i(\bot) \in X & (X \text{ zulässig}) \\
& \Leftarrow & \bot \in X \wedge (\forall n \in \mathbb{N} : f^n(\bot) \in X \to f(f^n(\bot)) \in X) & \\
& & & \text{(Induktion } \mathbb{N}) \\
& \Leftarrow & \bot \in X \wedge (\forall y \in X \to f(y) \in X) & \text{(Gen.)}
\end{array}$$

Aufgabe

Übung 4.2. *Sei* $(D, \sqsubseteq)$ *CPO mit*

- $X = Y = \mathbb{N}$
- $D = X \nrightarrow Y$: *Menge aller partieller Funktionen* f *mit* $\mathrm{dom}(f) \subseteq X$ *und* $\mathrm{cod}(f) \subseteq Y$.
- *Sei* $f, g \in X \nrightarrow Y$.

$$f \sqsubseteq g \text{ gdw } \mathrm{dom}(f) \subseteq \mathrm{dom}(g) \wedge (\forall x \in \mathrm{dom}(f) : f(x) = g(x))$$

Betrachte

$$F : \quad D \quad \to \quad \mathbb{N} \times \mathbb{N}$$

$$g \quad \mapsto \quad \begin{cases} \{(0,1)\} & g = \emptyset \\ \{(x, x \cdot g(x-1)) : x - 1 \in \mathrm{dom}(g)\} & \text{sonst} \end{cases}$$

Aufgabe

Zeigen Sie:

1. $\forall g \in D : F(g) \in D$, d.h. $F : D \to D$
2. $F : D \to D$ stetig
3. $\forall n \in \mathbb{N} : \mu F(n) = n!$

Bemerkung:

- μF kann aufgefasst werden als Semantik einer Funktionsdefinition

$$\begin{aligned} &\text{function Fac}(n : \mathbb{N}_\perp) : \mathbb{N}_\perp =_{\text{def}} \\ &\quad \text{if } n = 0 \text{ then } 1 \\ &\quad \text{else } n \cdot \text{Fac}(n-1) \end{aligned}$$

- Stichwort: 'derived functions' in ASM

Aufgabe

Übung 4.3. *Zeigen Sie: Sei $G = (V, E)$ ein unendlicher gerichteter Graph mit*

- *G besitzt endlich viele Wurzeln (Knoten, ohne eingehende Kanten).*
- *Jeder Knoten besitzt endlichen Aus-Grad.*
- *Jeder Knoten ist erreichbar von einer Wurzel.*

Dann existiert ein unendlicher Pfad, der bei einer Wurzel beginnt.

Verteilte ASM

Definition 4.4. *Eine DASM A über eine Signatur (Vokabular) Σ ist gegeben durch:*

- *Ein verteiltes Programm Π_A über Σ.*
- *Eine nicht-leere Menge I_A initialer Zustände*
  Ein initialer Zustand legt eine mögliche Interpretation von Σ über eine potentiell unendlichen Basismenge X fest.

A enthält in seiner Signatur ein dynamisches Relationssymbol AGENT, die als endliche Menge autonom operierender Agenten interpretiert wird.

- *Das Verhalten eines Agenten a in Zustand S von A ist durch $program_S(a)$ festgelegt.*
- *Ein Agent kann terminiert werden durch die Festlegung von $program_S(a) := undef$ (Darstellung eines ungültigen Programms).*

Partiell geordnete Läufe

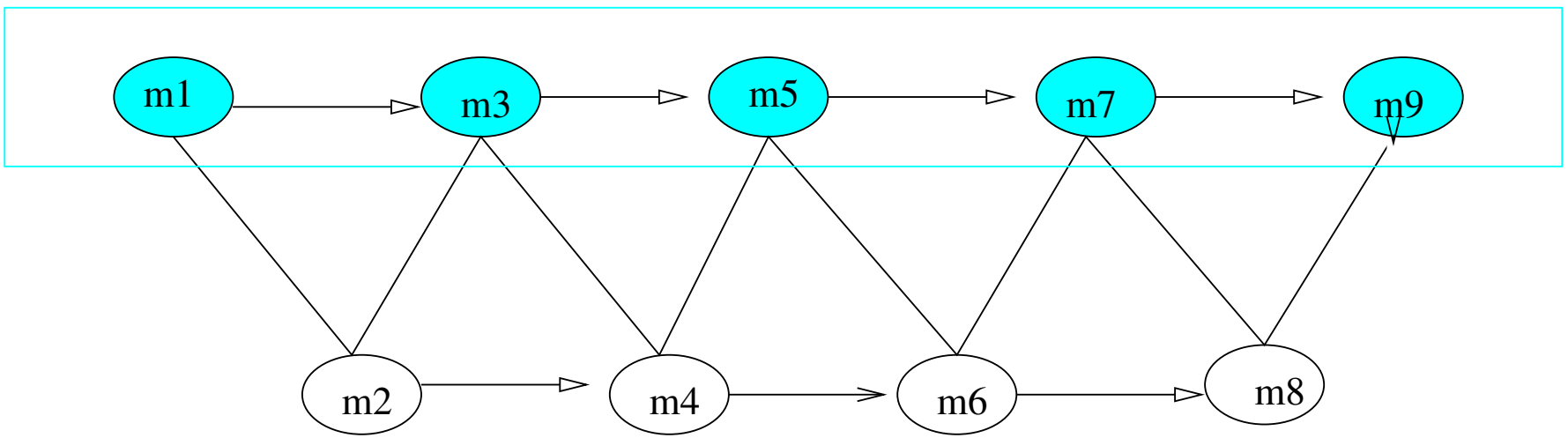
Ein Lauf einer verteilten ASM A ist durch ein Tripel $\varrho \rightleftharpoons (M, \lambda, \sigma)$ mit folgenden Eigenschaften gegeben:

- 1. M ist eine partiell geordnete Menge von Zügen (moves), wobei jeder Zug nur endlich viele Vorgänger hat.
- 2. λ ist eine Funktion auf M, die jedem Zug einen Agenten zuordnet, so dass die Züge eines einzelnen Agenten stets linear geordnet sind.
- 3. σ assoziiert einen Zustand von A mit jedem endlichen initialen Segment Y von M, hierbei ist $\sigma(Y)$ das "Ergebnis der Durchführung aller Züge" in Y. $\sigma(Y)$ ist ein initialer Zustand falls Y leer ist.
- 4. Die Kohärenz Bedingung ist erfüllt:
  Ist max eine Menge maximaler Elemente in einem endlichen initialen Segment X von M und $Y = X \setminus max$, dann ist $\lambda(x)$ ein Agent in $\sigma(Y)$ für $x \in max$ und man erhält $\sigma(X)$ aus $\sigma(Y)$ durch Feuern von $\{\lambda(x) : x \in max\}$ (ihre Programme) in $\sigma(Y)$.

Bemerkung, Beispiel

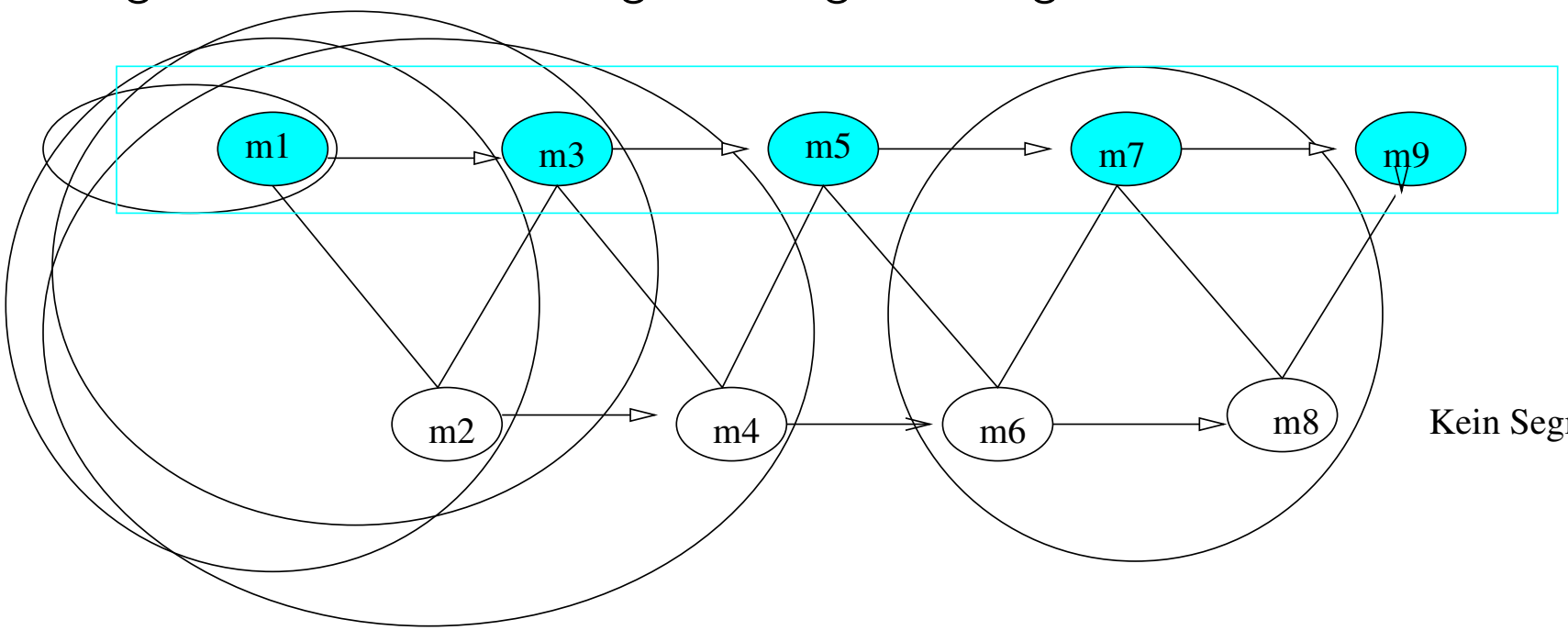
Die Agenten von A modellieren die konkurrierenden Kontrollthreads in der Ausführung von Π_A.
Ein Lauf kann als der gemeinsame Anteil der Historie der gleichen Berechnung aus der Sicht mehrerer Beobachter.

Die Rolle von λ:



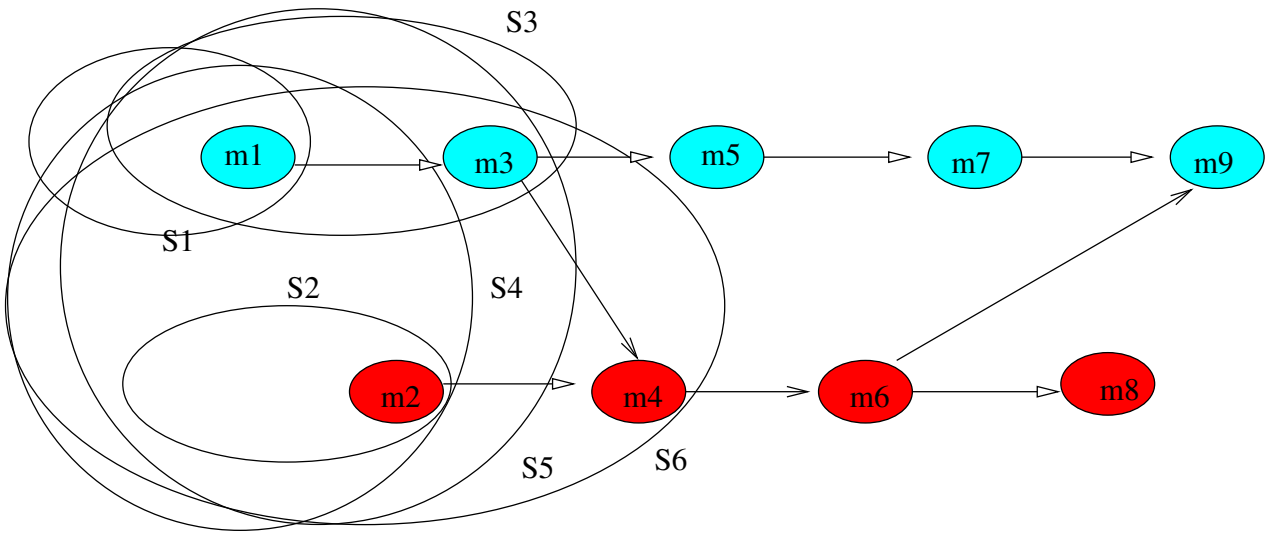
Bemerkung, Beispiel

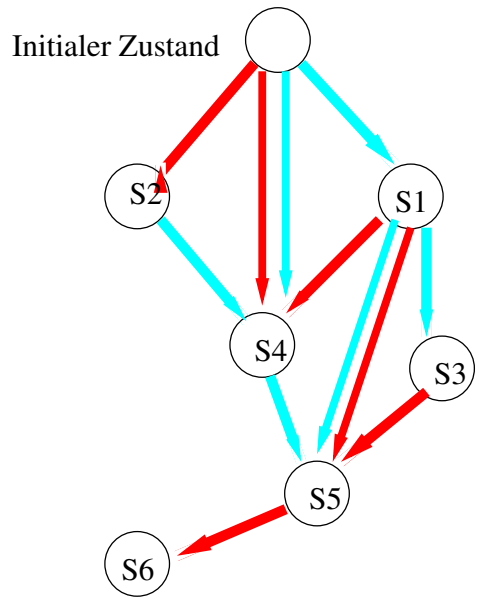
Die Rolle von σ: Momentaufnahmen sind die initialen Segmente der partiell geordneten Menge M. Jedem initialen Segment wir ein Zustand von A (Interpretation von Σ) zugeordnet, der die Wirkung der Programme der im Segment vorkommenden Agenten wiedergibt.
$\rightsquigarrow$"Ergebnis der Durchführung aller Züge" im Segment.



Kohärenz Bedingung, Beispiel

Ist *max* eine Menge maximaler Elemente in einem endlichen initialen Segment X von M und $Y = X \setminus max$, dann ist $\lambda(x)$ ein Agent in $\sigma(Y)$ für $x \in max$ und man erhält $\sigma(X)$ aus $\sigma(Y)$ durch Feuern von $\{\lambda(x) : x \in max\}$ (ihre Programme) in $\sigma(Y)$.





Folgerungen aus der Kohärenz Bedingung

Lemma 4.5. *Alle Linearisierungen eines initialen Segments eines Laufes ϱ haben den selben Endzustand.*

Lemma 4.6. *Eine Eigenschaft P gilt genau dann in allen erreichbaren Zuständen eines Laufes ϱ, wenn sie in jedem erreichbaren Zustand von jeder Linearisierung von ϱ gilt.*

Einfaches Beispiel

Beispiel 4.7. *Seien $\{door, window\}$ aussagenlogische Konstanten in der Signatur mit natürlicher Bedeutung:*
door = true bedeute Tür offen und analog für Fenster.

Das Programm bestehe aus zwei Agenten, einen Tür-Manager d und einen Fenster-Manager w mit Programmen:

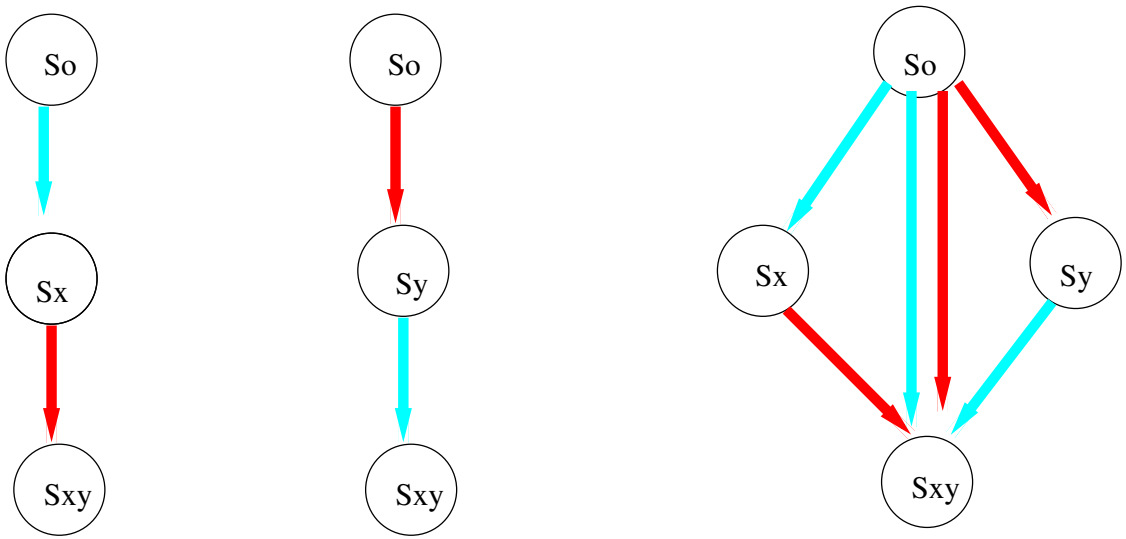
$program_d = door := true \quad // \text{ move } x$
$program_w = window := true \quad // \text{ move } y$

Im initialen Zustand S_0 seien Tür und Fenster geschlossen, d und w seien in der Agentenmenge.

Welche sind die möglichen Läufe?

Einfaches Beispiel (Fort.)

Seien $\varrho_1 = ((\{x, y\}, x < y), id, \sigma)$, $\varrho_2 = ((\{x, y\}, y < x), id, \sigma)$,
$\varrho_3 = ((\{x, y\}, <>), id, \sigma)$ (Gröbste Partialordnung)

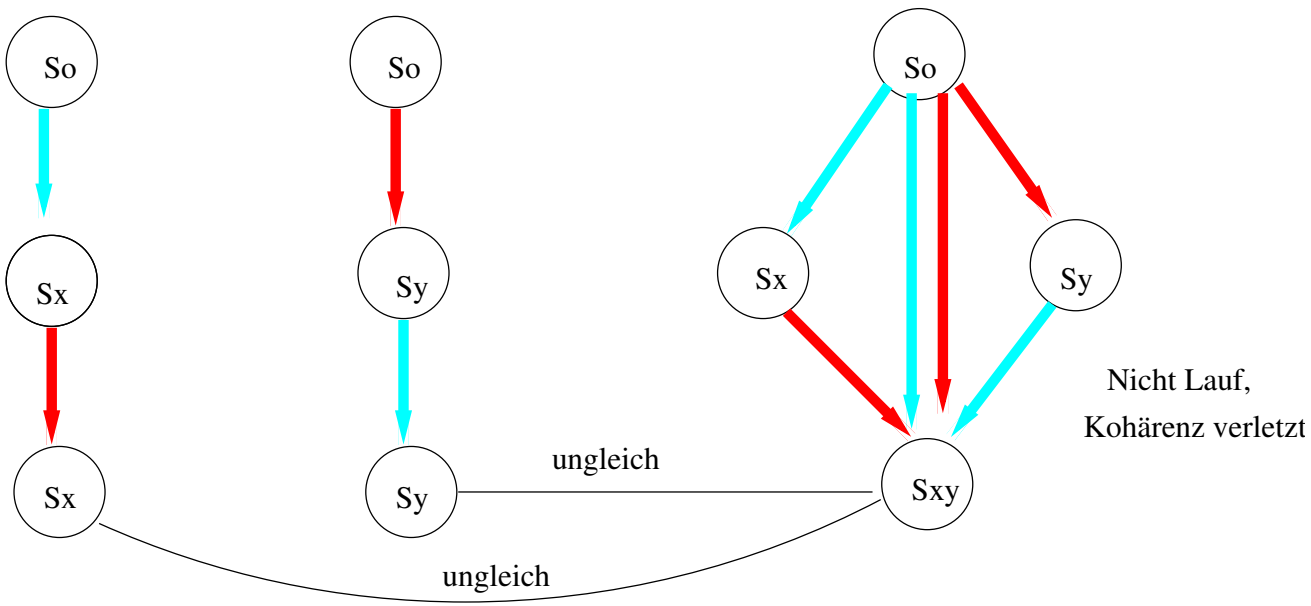


Variante von Einfaches Beispiel

Das Programm bestehe aus zwei Agenten, einen Tür-Manager d und einen Fenster-Manager w mit Programmen:

$program_d = if\ \neg window\ then\ \ door := true$ // move x
$program_w = if\ \neg door\ then\ \ window := true$ // move y

Im initialen Zustand S_0 seien Tür und Fenster geschlossen, d und w seien in der Agentenmenge. Wie sehen nun Läufe aus? Gleiche ϱ's



Mehr Variationen

Übung 4.8. *Betrachte folgende Agentenpaare*
$x, y \in \mathbb{N}$ *(*$x = 2, y = 1$ *im Anfangszustand)*

1. $a = x := x + 1$ *und* $b = x := x + 1$
2. $a = x := x + 1$ *und* $b = x := x - 1$
3. $a = x := y$ *und* $b = y := x$

Welche Läufe sind mit zweielementigen partiell geordneten Mengen möglich?
Versuche alle Läufe zu charakterisieren.

Mehr Variationen

Betrachte folgende Agenten mit üblicher Interpretation:

1. $Program_d =$ *if* $\neg window$ *then* $door := true$ //move x
2. $Program_w =$ *if* $\neg door$ *then* $window := true$ //move y
3. $Program_l =$ *if* $\neg light \wedge (\neg door \vee \neg window)$ *then* //move z
   $light := true$
   $door := false$
   $window := false$

Welche Endzustände sind möglich, wenn im Anfangszustand die drei Konstanten false sind?

Noch zum Üben

Consumer-producer problem: Assume a single producer agent and two or more consumer agents operating concurrently on a global shared structure. This data structure is linearly organized and the producer adds items at the one end side while the consumers can remove items at the opposite end of the data structure. For manipulating the data structure, assume operations *insert* and *remove* as introduced below.

$$\mathit{insert} : \mathit{Item} \times \mathit{ItemList} \rightarrow \mathit{ItemList}$$
$$\mathit{remove} : \mathit{ItemList} \rightarrow (\mathit{Item} \times \mathit{ItemList})$$

(1) Which kind of potential conflicts do you see?
(2) How does the semantic model of partially ordered runs resolve such conflicts?

Umgebung

Reaktive Systeme zeichnen sich durch ihre Interaktion mit der Umgebung aus. Modelliert kann dies mit Hilfe eines Umgebungsagenten. Läufe können nun (mit λ) diesen Agenten enthalten, λ muss dann auch die Updatemenge der Umgebung in den entsprechenden Zug festlegen.
Die Kohärenzbedingung muss dann auch für solche Läufe gelten.

Für extern kontrollierte Funktionen führt dies sicherlich nicht zu Inkonsistenzen bei den Updatemengen, das Verhalten der internen Agenten kann natürlich beeinflusst werden. Bei shared Funktionen können bei der simultanen Ausführung der Züge eines internen Agenten und der Umgebung inkonsistente Updatemengen entstehen.

Oft werden noch Einschränkungen (Annahmen) der Umgebung gemacht. Hier gibt es viele Möglichkeiten: Umgebung wird nur beobachtet oder Umgebung erfüllt Integritätsbedingungen.

Zeit

Die Beschreibung von Real Time Verhalten muss explizit Zeitaspekte berücksichtigen. Dies kann mit Hilfe von Timer (siehe SDL), globale Systemzeit oder lokale Systemzeiten erfolgen.

- Reaktionen können ohne Zeitverzögerung (instantan) sein (das Feuern der Regeln der Agenten erfordert keine Zeit)
- Aktionen benötigen Zeit

Bei der globalen Zeitbetrachtung geht man davon aus, dass eine linear geordnete Domäne *TIME* vorliegt, etwa mit Deklarationen

$$\mathit{domain}\ \ (\mathit{TIME}, \leq),\ \ (\mathit{TIME}, \leq)\ \subset (\mathbb{R}, \leq)$$

Hierbei wird die Zeit von einer diskreten Systemuhr gemessen:

$$\mathit{monitored}\ \ \mathit{now} :\longrightarrow \mathit{TIME}$$

Geldautomat

Übung 4.9. *Abstrakte Modellierung eines Geldautomaten:*
Drei Agenten sind im Modell: GA-Manager, Autentitifikations-Manager, Konto-Manager. Um eine Summe vom Konto abzuheben sind folgende logische Operationen durchzuführen:

1. *Eingabe der Karte (Nummer) und der PIN.*
2. *Überprüfe Gültigkeit der Karte und PIN (AU-manager).*
3. *Eingabe der Summe.*
4. *Überprüfe ob Summe vom Konto abgehoben werden kann (KO-Manager).*
5. *Falls OK aktualisiere Konto Stand und gebe Summe aus.*
6. *Falls nicht OK gebe entsprechende Nachricht aus.*

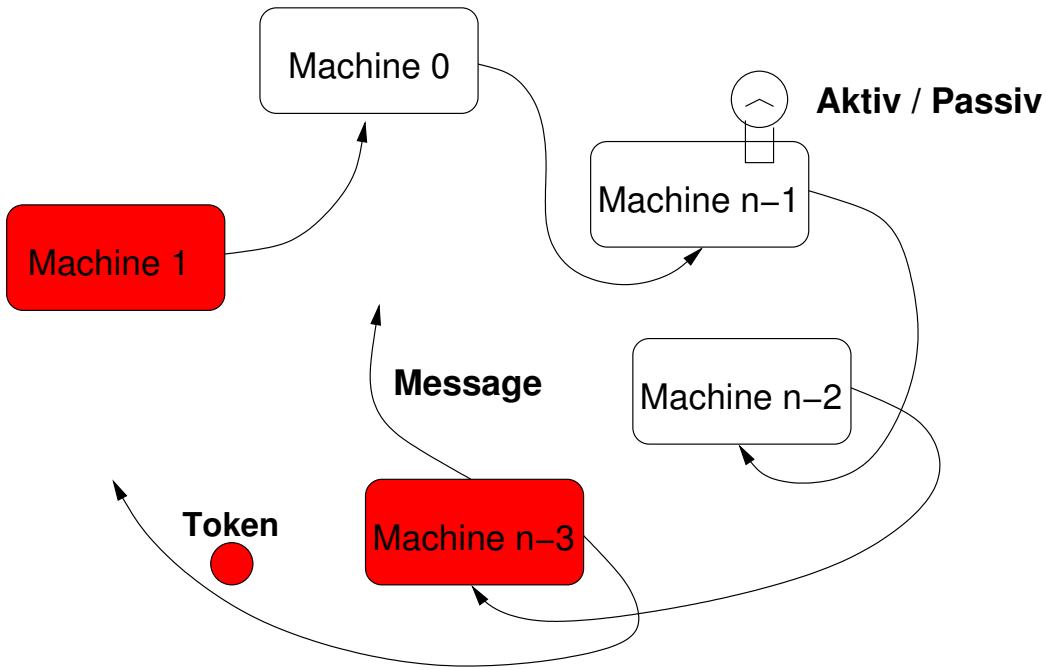
Realisiere ein asynchrones Kommunikationsmodell wobei Timeouts Transaktionen abbrechen können.

Distributed Termination Detection

Beispiel 4.10. *Modelliere folgendes Terminierungsfestellungsprotokoll:*

Genau dann wird eine passive Maschine aktiv, wenn sie eine Nachricht einer anderen Maschine erhält.

Nur aktive Maschinen können Nachrichten senden.



Edsger W. Dijkstra, W. H. J. Feijen, and A.J.M. van Gasteren. Derivation of a Termination Detection Algorithm for Distributed Computations. IPL 16 (1983).

Annahmen für Distributed Termination Detection

Rule 0 Falls $Machine_{i+1}$ aktiv ist, so behält sie das Token; ist sie inaktiv so leitet sie das Token an $Machine_i$ weiter.

Rule 1 Eine Maschine die eine Message sendet färbt sich rot.

Rule 2 Propagiert $Machine_{i+1}$ die Probe, so leitet sie ein rotes Token weiter falls ihre Farbe rot ist, ansonsten leitet sie das Token unverändert weiter an $Machine_i$.

Rule 3 Nach Beendigung einer erfolglosen Probe leitet $Machine_0$ eine neue Probe ein.

Rule 4 $Machine_0$ initiiert eine neue Probe indem sie sich weiß färbt und ein weißes Token an $Machine_{n-1}$ leitet.

Rule 5 Nach Weiterleiten des Tokens an $Machine_i$ wird $Machine_{i+1}$ weiß gefärbt. (Beachte die ursprüngliche Farbe von $Machine_{i+1}$ kann die Farbe des Tokens beeinflusst haben).

Distributed Termination Detection: Verfahren

Signatur:

static

$COLOR = \{red, white\}$ $TOKEN = \{redToken, whiteToken\}$

$MACHINE = \{0, 1, 2, \ldots, n-1\}$

$next : MACHINE \rightarrow MACHINE$

Z.B. mit $next(0) = n-1, next(n-1) = n-2, \ldots, next(1) = 0$

controlled

$color : MACHINE \rightarrow COLOR$ $token : MACHINE \rightarrow TOKEN$

$RedTokenEvent, WhiteTokenEvent : MACHINE \rightarrow BOOL$

monitored $Active : MACHINE \rightarrow BOOL$

$SendMessageEvent : MACHINE \rightarrow BOOL$

Distributed Termination Detection: Verfahren

Makros: (Rule Definitions)

- $ReactOnEvents(m : MACHINE) =$
  $if\ \ RedTokenEvent(m)\ \ then$
  $token(m) := redToken$
  $RedTokenEvent(m) :=\ \ undef$
  $if\ \ WhiteTokenEvent(m)\ \ then$
  $token(m) := whiteToken$
  $WhiteTokenEvent(m) :=\ \ undef$
  $if\ \ SendMessageEvent(m)\ \ then\ \ color(m) := red$ Rule 1

- $Forward(m : MACHINE, t : TOKEN) =$
  $if\ \ t = whiteToken\ \ then$
  $WhiteTokenEvent(next(m)) :=\ \ true$
  $else$
  $RedTokenEvent(next(m)) := true$

Distributed Termination Detection: Verfahren

Programme

- *RegularMachineProgram* =

  ReactOnEvents(*me*)
  if ¬ *Active*(*me*) ∧ *token*(*me*) ≠ *undef* *then* Rule 0
  InitializeMachine(*me*) Rule 5
  if *color*(*me*) = *red* *then*
  Forward(*me*, *redToken*) Rule 2
  else
  Forward(*me*, *token*(*me*)) Rule 2

- Mit *InitializeMachine*(*m* : *MACHINE*) =

  token(*m*) := *undef*
  color(*m*) := *white*

Distributed Termination Detection: Verfahren

Programme

- *SupervisorMachineProgram* =

$$
\begin{array}{l}
\mathit{ReactOnEvents}(\mathit{me}) \\
\mathit{if} \neg\ \mathit{Active}(\mathit{me}) \wedge\ \mathit{token}(\mathit{me}) \neq\ \mathit{undef}\ \ \mathit{then} \\
\quad \mathit{if}\ \ \mathit{color}(\mathit{me}) =\ \mathit{white} \wedge\ \mathit{token}(\mathit{me}) =\ \mathit{whiteToken}\ \ \mathit{then} \\
\quad\quad \mathit{ReportGlobalTermination} \\
\quad \mathit{else} \quad \text{Rule 3} \\
\quad\quad \mathit{InitializeMachine}(\mathit{me}) \quad \text{Rule 4} \\
\quad\quad \mathit{Forward}(\mathit{me}, \mathit{whiteToken}) \quad \text{Rule 4}
\end{array}
$$

Distributed Termination Detection

Initiale Zustände

$$\exists m_0 \in MACHINE$$
$$(program(m_0) = SupervisorMachineProgram \wedge$$
$$token(m_0) = redToken \wedge$$
$$(\forall m \in MACHINE)(m \neq m_0 \Rightarrow$$
$$(program(m) = RegularMachineProgram \wedge token(m) = undef)))$$

Umgebungsconstraints Für alle Läufe und alle Linearisierungen gilt:

$$\mathbf{G}\ (\forall m \in MACHINE)$$
$$((SendMessageEvent(m) = true \Rightarrow (\mathbf{P}(Active(m)) \wedge Active(m))) \wedge$$
$$((Active(m) = true \wedge \mathbf{P}(\neg Active(m)) \Rightarrow$$
$$(\exists m' \in MACHINE)\ (m' \neq m \wedge\ SendMessageEvent(m'))))$$

Nextconstraints

Distributed Termination Detection

Korrektheit nach Dijkstra

Annahmen: Die Maschinen bilden ein geschlossenes System, d.h. Nachrichten können nur untereinander Versendet werden. Das System im Initialzustand kann beliebig gefärbt sein und mehrere Maschinen können aktiv sein. Das Token befindet sich bei der 0'ten Maschine. Die angebebenen Regel beschreiben die Übergabe des Tokens und die Färbung der Maschinen bei bestimmten Aktivitäten. Festzustellen ist ein Zustand bei dem alle Maschinen passiv (nicht aktiv) sind. Dies ist ein stabiler Zustand des Systems, da nur aktive Maschinen Nachrichten versenden können und passive Maschinen nur aktiviert werden können durch Erhalt einer Nachricht.

Die Invariante: Sei t die Stelle an der sich das Token befindet, dann gilt $(\forall i : t < i < n \;\; \mathit{Machine}_i \text{ ist passiv}) \vee (\exists j : 0 \leq j \leq t \;\; \mathit{Machine}_j \text{ ist rot}) \vee (\mathit{Token} \text{ ist rot})$

Distributed Termination Detection

$(\forall i : t < i < n \;\; Machine_i \text{ ist passiv}) \vee (\exists j : 0 \leq j \leq t \;\; Machine_j \text{ ist rot}) \vee$
$(Token \text{ ist rot})$

Korrektheitsargument
Wenn das Token zu $Machine_o$ gelangt ist $t = 0$ und die Invariante gilt.
Falls
$(Machine_o \text{ ist passiv}) \wedge (Machine_o \text{ ist weiß}) \wedge (Token \text{ ist weiß})$
so muß
$(\forall i : 0 < i < n \;\; Machine_i \text{ ist passiv})$ gelten, d.h. Terminierung.

Nachweis der Invariante Induktion nach t:
Der Fall t = n - 1 ist einfach.
Invariante gelte für $0 < t < n$, zeige sie gilt für $t - 1$.

Distributed Termination Detection

Gilt die Invariante auch in allen Zuständen aller Linearisierungen von Läufen der DASM ? **Nein**

- **Problem 1** Rot Färben einer aktiven Maschine die eine Nachricht versendet geschieht in späteren Zustand. Es müsste im selben Zustand passieren in dem die nachrichterhaltende Maschine aktiv wird.
  Lösung *color* ist shared Funktion. Anstatt $SendMessageEvent(m)$ zu setzen wird $color(m) = red$ von der Umgebung gesetzt.
- **Problem 2** Es gibt Zustände für die keine Maschine das Token hat. Die tokenhabende Maschine Initialisiert sich und setzt ein Event, im resultierenden Zustand hat keine Maschine das Token.
  Lösung Statt ein *FarbTokenEvent* zu setzen, wird direkt $token(next(m))$ richtig gesetzt.
- **Resultat** Abstraktere Maschine. Die Umgebung regelt die Aktivität der Maschinen, das Nachrichtenübermitteln und die Färbung.

Verfeinerungsbegriffe für ASM's

Frage: Ist im Terminierungsbeispiel die angegebene DASM eine Verfeinerung der abstrakteren DASM? $\rightsquigarrow$

Allgemeine Verfeinerungsbegriffe für ASM's

- Verfeinerungen werden in der Regel für BASM definiert, d.h. Läufe sind stets linear, was die Betrachtung vereinfacht.
- Verfeinerungen erlauben Abstraktionen, Realisierungen von Daten und Prozeduren.
- ASM Verfeinerungen sind meistens Problemorientiert: Abhängigkeit von der Anwendung und somit sollten sie flexibel sein.
- Beweisaufgaben werden mit Hilfe von korrekten und vollständigen Verfeinerungen Strukturiert und Vereinfacht.

Siehe ASM-Buch.

Algebraische Spezifikation - Gleichheitslogik

Anforderungen an Spezifikationstechniken:

- Abstraktion (Verfeinerung)
- Strukturierungsmechanismen
  Zerlegung, Kombination, Erweiterung-Instantierung
- klare (eindeutige und plausible) Semantik
- Unterstützung des „verify while develop“-Prinzips
- Ausdruckskraft (alle partial rekursiven Funktionen darstellbar)
- Lesbarkeit (Adequatheit) (Angemessenheit)

  ⋮

Algebraische Spezifikation - Algebren

Spezifikation von Datentypen

Syntax	Gleichungen	Programme
Signatur Axiome	$t_1 = t_2$ if φ then $t_1 = t_2$	Datenoperation Gerichtete Anwendung

Algebren

heterogene	ordnungssortierte	homogene
(mehrsortig)	(mehrsortig)	(einsortig)

Einsortige Algebren

Beispiel 6.1. *a) Gruppen*

SORT:: g

SIG:: $\cdot : g, g \to g \qquad 1 :\to g \qquad {}^{-1} : g \to g$

EQN:: $x \cdot 1 = x \qquad x \cdot x^{-1} = 1 \qquad (x \cdot y) \cdot z = x \cdot (y \cdot z)$

All-quantifizierte Gleichungen

Modelle sind Gruppen

Frage: Welche Gleichungen gelten in allen Gruppen,
d. h. $EQN \models t_1 = t_2$

$$1 \cdot x = x \qquad x^{-1} \cdot x = 1 \qquad (x^{-1})^{-1} = x$$

Einsortige Algebren

Gleicheitslogik: Ersetze „Gleiches“ durch „Gleiches“
Problem: Zykel, Nichtterminierung
Lösung: Gerichtete Gleichungen, $\rightsquigarrow$ Termersetzungssysteme

Finde R konvergent mit $\underset{EQN}{=} = \overset{*}{\underset{R}{\Longleftrightarrow}}$

$$
\begin{array}{ll}
x \cdot 1 \to x & 1 \cdot x \to x \\
x \cdot x^{-1} \to 1 & x^{-1} \cdot x \to 1 \\
1^{-1} \to 1 & (x^{-1})^{-1} \to x \\
(x \cdot y)^{-1} \to y^{-1} \cdot x^{-1} & (x \cdot y) \cdot z \to x \cdot (y \cdot z) \\
x^{-1} \cdot (x \cdot y) \to y & x \cdot (x^{-1} \cdot y) \to y
\end{array}
$$

Mehrsortige Algebren

b) Listen über nat-Zahlen

SIG: BOOL, NAT, LIST Sorten

true, false: $\rightarrow$ BOOL

0 $\rightarrow$ NAT suc: NAT $\rightarrow$ NAT

+: NAT, NAT $\rightarrow$ NAT

eq: NAT, NAT $\rightarrow$ BOOL

nil: $\rightarrow$ LIST . : NAT, LIST $\rightarrow$ LIST

app: LIST, LIST $\rightarrow$ LIST

rev: LIST $\rightarrow$ LIST

Mehrsortige Algebren

Axiome sind allquantifizierte Gleichungen, d.h.
$\forall x_1, ..., x_n : \quad t_1(x_1, ..., x_n) = t_2(x_1, ..., x_n)$ wobei

$t_1(x_1, ..., x_n), t_2(x_1, ..., x_n)$ Terme gleicher Sorte in der Signatur sind.

EQN : $n + 0 = n \quad n_1 + \text{suc}(n_2) = \text{suc}(n_1 + n_2)$

$$\text{eq}(0, 0) = \text{true} \quad \text{eq}(0, \text{suc}(n)) = \text{false}$$
$$\text{eq}(\text{suc}(n), 0) = \text{false} \quad \text{eq}(\text{suc}(n_1), \text{suc}(n_2)) = \text{eq}(n_1, n_2)$$

$$\text{app}(\text{nil}, l) = l \quad \text{app}(n.l_1, l_2) = n.\,\text{app}(l_1, l_2)$$

$$\text{rev}(\text{nil}) = \text{nil} \quad \text{rev}(n.l) = \text{app}(\text{rev}(l), n.\text{nil})$$

Mehrsortige Algebren

Terme der Sorten BOOL, NAT, LIST als Bezeichner für Elemente (Standarddefinition!)
Welche Algebra wird spezifiziert? Wie rechnet man in dieser Algebra?
Gleichungen richten $\rightsquigarrow$ Termersetzungssystem R. Offenbar gilt:

$$s^i(0) + s^j(0) \xrightarrow[R]{*} s^{i+j}(0)$$

$$\mathrm{app}(3.1.\mathrm{nil}, \mathrm{app}(5.\mathrm{nil}, 1.2.3.\mathrm{nil})) \xrightarrow[R]{*} 3.1.5.1.2.3.\mathrm{nil}$$

$$\begin{aligned} \mathrm{rev}(3.1.\mathrm{nil}) &\longrightarrow \mathrm{app}(\mathrm{rev}(1.\mathrm{nil}), 3.\mathrm{nil}) \\ &\longrightarrow \mathrm{app}(\mathrm{app}(\mathrm{rev}(\mathrm{nil}), 1.\mathrm{nil}), 3.\mathrm{nil}) \\ &\longrightarrow \mathrm{app}(\mathrm{app}(\mathrm{nil}, 1.\mathrm{nil}), 3.\mathrm{nil}) \\ &\longrightarrow \mathrm{app}(1.\mathrm{nil}, 3.\mathrm{nil}) \xrightarrow{*} 1.3.\mathrm{nil} \end{aligned}$$

Frage: Gilt

$$\mathrm{app}(x.y.\mathrm{nil}, z.\mathrm{nil}) =_E \mathrm{app}(x.\mathrm{nil}, y.z.\mathrm{nil}) \ ?$$

Mehrsortige Algebren

Manche Gleichungen gelten nicht in allen Modellen von EQN$=E$.
z. B.

$$x + y \neq_E y + x$$
$$\text{app}(x, \text{app}(y, z)) \neq_E \text{app}(\text{app}(x, y), z)$$
$$\text{rev}(\text{rev}(x)) \neq_E x$$

die Termpaare sind nicht zusammenführbar.

Unterscheidung:

- Gleichungen die in allen Modellen von E gelten.
- Gleichungen die im Datenmodell von E gelten.

$$x + y = y + x :: s^i 0 + s^j 0 = s^j 0 + s^i 0 \text{ alle } i, j$$

$$\text{rev}(\text{rev}(x)) = x \text{ für } x \equiv s^{i_1} 0.s^{i_2} 0. \ldots . s^{i_n} 0.\text{nil}$$

These: Datentypen sind Algebren

ADT: Abstrakte Datentypen.
Unabhängig von der Repräsentation der Daten.

Spezifikation abstrakter Datentypen:

Konzepte aus Logik/universelle Algebra
Ziel: gemeinsame Sprachebene für Spezifikation und Implementierung.

Mittel für Korrektheitsbeweise:

Syntax, L Formeln (P-Logik,Hoare,...)

Cl: Folgerungsabschluß (Z.B. $\models$, $Th(A)$,...)

Folgerungssbschluß

$Cl : \mathbb{P}(L) \to \mathbb{P}(L)$ (Teilmengen von L) mit

a) $A \subset L \rightsquigarrow A \subset Cl(A)$
b) $A, B \subset L, A \subseteq B \rightsquigarrow Cl(A) \subseteq Cl(B)$ (Monotonie)
c) $Cl(A) = Cl(Cl(A))$ (Maximalität)

Wichtige Begriffe:

Konsistenz: $A \subsetneq L$ A ist konsistent falls $Cl(A) \subsetneq L$

Implementierung: A implementiert B

(Verfeinerung) $L \subset L', Cl(B) \subseteq Cl(A)$

Signatur - Terme

Definition 6.2. *a) Signatur ist Tripel* $\mathrm{sig} = (S, F, \tau)$ *(Abgekzt.:* Σ*)*

- S *endliche Menge von Sorten*
- F *Menge von Operatoren (Funktionssymbolen)*
- $\tau : F \to S^+$ *Funktion Stelligkeit, d. h.*
  $\tau(f) = s_1 \cdots s_n s$, $n \geq 0$, s_i *Argumentsorten*, s *Zielsorte.*

Schreibe: $f : s_1, \ldots, s_n \to s$

(Beachte $n = 0$*) möglich, Konstante der Sorte* S*.*

Signatur - Terme

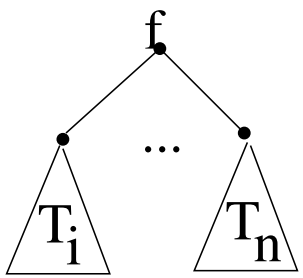
b) Term(F): Menge der Grundterme über sig und ihre Baumpräsentation.

$$\mathrm{Term}(F) := \dot{\bigcup_{s \in S}} \mathrm{Term}_s(F)$$

rekursiv def.

- $f :\longrightarrow s$, so $f \in \mathrm{Term}_s(F)$ Repräsentation: $\cdot f$
- $f : s_1, \ldots, s_n \longrightarrow s$, $t_i \in \mathrm{Term}_{s_i}(F)$ mit Rep. T_i so $f(t_1, \ldots, t_n) \in \mathrm{Term}_s(F)$ mit Rep.

Beachte Repräsentation durch geordnete Bäume



Signatur - Terme

c) $V = \dot{\bigcup_{s \in S}} V_s$ Variablensystem $V \cap F = \varnothing$.

Jedes $x \in V_s$ hat Funktionalität $x :\longrightarrow s$

Setze: $\mathrm{Term}(F, V) := \mathrm{Term}(F \cup V)$.

Bezeichnung: Terme über sig in Var. V.
(F und τ geeignet erweitert).

Intention: Für Variablen dürfen beliebige Objekte, d.h. Terme, eingesetzt werden.

Striktheit - Positionen-Teilterme

Definition 6.3. *a)* $s \in S$ *strikt, falls* $\mathrm{Term}_s(F) \neq \varnothing$
Gibt es zu jeder Sorte $s \in S$ *entweder eine Konstante der Sorte* S *oder eine Funktion* $f : s_1, \ldots, s_n \longrightarrow s$, *so dass die* s_i *strikt sind, dann sind alle Sorten der Signatur strikt.*$\rightsquigarrow$ *strikte Signaturen (Generalvoraussetzung)*

b) Teilterme $(t) = \{t_p \mid p$ *Stelle (Position) in* p, t_p *Teilterm in* $p\}$
Stellen werden durch Folgen über $\mathbb{N}$ *dargestellt*
(Elemente von $\mathbb{N}^*$, e *leere Folge).*
$O(t)$ *Menge der Stellen in* t,
Für $p \in O(t)$ t_p *(oder* $t|_p$*) Teilterm von* t *an Stelle* p

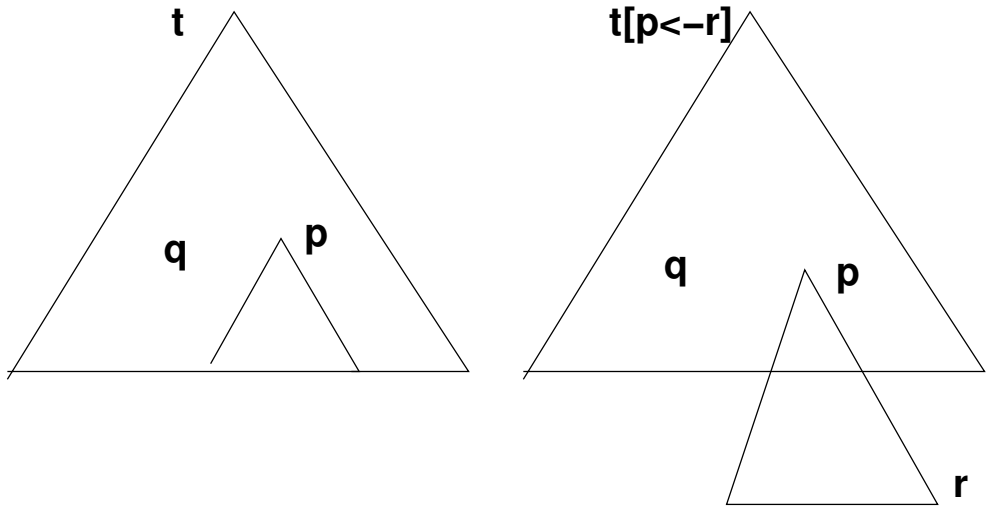
- t *Konstante oder Variable:* $O(t) = \{e\} \quad t_e \equiv t$
- $t \equiv f(t_1, \ldots, t_n)$ *so*
  $O(t) = \{ip \mid 1 \leq i \leq n, p \in O(t_i)\} \cup \{e\}$
  $t_{ip} \equiv t_i|_p$ *und* $t_e \equiv t$.

Termersetzung

c) Termersetzung: $t, r \in \mathrm{Term}(F, V)$
$p \in O(t)$: mit $r, t_p \in \mathrm{Term}_s(F, V)$ für eine Sorte s.

Dann ist
$t[r]_p$, bzw. $t[p \leftarrow r]$ bzw. t_p^r der Term der aus t entsteht durch Ersetzen vom Teilterm t_p durch r.

Also $t[p \leftarrow r]_q = t_q$ für $q \mid p$ und



$t[p \leftarrow r]_p = r$

Signaturen - Terme

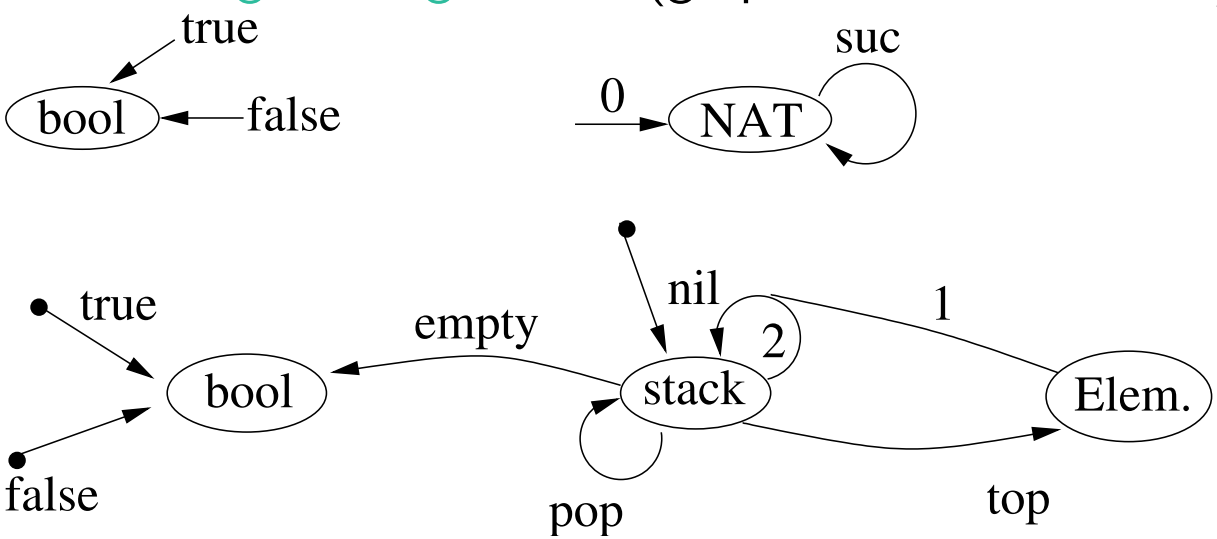
Beispiel 6.4. $S = (\mathrm{BOOL}, \mathrm{NAT}, \mathrm{LIST})$, $F = \{true, false, \ldots\}$,
$\tau : F \to S^* :: true :\to \mathrm{BOOL}$, $eq : \mathrm{NAT}, \mathrm{NAT} \to \mathrm{BOOL} \ldots$

$$V = \underbrace{V_{\mathrm{BOOL}}}_{\{b_i : i \in \mathbb{N}\}} \cup \underbrace{V_{\mathrm{NAT}}}_{\{x_i : i \in \mathbb{N}\}} \cup \underbrace{V_{\mathrm{LIST}}}_{\{l_i : i \in \mathbb{N}\}}$$

Grundterme:
$true, false, eq(0, \mathrm{suc}(0)) \in \mathrm{Term}_{\mathrm{BOOL}}(S)$
$0, \mathrm{suc}(0), \mathrm{suc}(0) + (\mathrm{suc}(\mathrm{suc}(0)) + 0) \in \mathrm{Term}_{\mathrm{NAT}}(S)$
$\mathrm{app}(nil, \mathrm{suc}(0).(\mathrm{suc}(\mathrm{suc}(0)).nil) \in \mathrm{Term}_{\mathrm{LIST}}(S)$
$0.\,\mathrm{suc}(0), eq(true, false), \mathrm{rev}(0)$ *keine Terme.*
Allgemeine Terme:
$eq(x_1, x_2) \in \mathrm{Term}_{\mathrm{BOOLE}}(F, V), suc(x_1) + (x_2 + \mathrm{suc}(0)) \in \mathrm{Term}_{\mathrm{NAT}}(F, V)$
$\mathrm{app}(l_1, x_1.l_0) \in \mathrm{Term}_{\mathrm{LIST}}(F, V)$
$\mathrm{rev}(x_1.l) \in \mathrm{Term}_{\mathrm{LIST}}(F, V)$
$\mathrm{app}(x_1, l_2)$ *kein Term.*

Signaturen

Darstellung von Signaturen (graphisch oder normiert)



Notationen:

sig ...
sorts ...
ops ...
op: $W \to S$
$op_1, \ldots, op_i : W \to S$

Interpretationen: sig-Algebren

Definition 6.5. $\mathrm{sig} = (S, F, \tau)$ *Signatur. Eine sig-Algebra* $\mathfrak{A}$ *besteht aus*

1) *Trägermenge* $A = \bigcup_{s \in S} A_s, A_s \neq \varnothing$ *Trägermenge der Sorte* s.
2) *Funktionensystem* $F_{\mathfrak{A}} = \{f_{\mathfrak{A}} : f \in F\}$ *mit*
   $f_{\mathfrak{A}} : A_{s_1} \times \cdots \times A_{s_n} \to A_s$ *Funktion und* $\tau(f) = s_1 \cdots s_n s$.

Beachte: Die $f_{\mathfrak{A}}$ sind totale Funktionen.

Die Voraussetzung $A_s \neq \varnothing$ ist nicht zwingend.

Interpretationen: sig-Algebren

Beispiel 6.6. *Interpretationen*

a) sig $\equiv$ BOOL-*Algebren*, *true*, *false* $:\longrightarrow$ BOOL

$$\left.\begin{array}{llll} \mathfrak{A}_1 & \{0,1\} & true_{\mathfrak{A}_1} = 0 & false_{\mathfrak{A}_1} = 1 \\ \mathfrak{A}_2 & \{0,1\} & true_{\mathfrak{A}_2} = 0 & false_{\mathfrak{A}_2} = 0 \\ \mathfrak{A}_3 & \mathbb{N} & true_{\mathfrak{A}_3} = 4 & false_{\mathfrak{A}_3} = 5 \\ \mathfrak{A}_4 & \{true, false\} & true_{\mathfrak{A}_4} = true & false_{\mathfrak{A}_4} = false \end{array}\right\} \textit{bool-Alg.}$$

b) sig $\equiv$ NAT, 0, suc

$$\left\{\begin{array}{lllll} A_{i_{\mathrm{NAT}}} & \mathbb{N} & \mathbb{Z} & \mathbb{N} & \{true, false\} \quad \{0, \mathrm{suc}^i(0)\} \\ 0_{\mathfrak{A}_i} & 0 & 0 & 1 & true \quad 0 \\ \mathrm{suc}_{\mathfrak{A}_i} & \mathrm{suc}_{\mathbb{N}} & \mathrm{pred}_{\mathbb{Z}} & \mathrm{id}_{\mathbb{N}} & \mathrm{suc}(true) = false \quad \mathrm{suc}(0) = suc(0) \\ & & & & \mathrm{suc}(false) = true \quad \mathrm{suc}(\mathrm{suc}^i(0)) = \mathrm{suc}^{i+1}(0) \end{array}\right.$$

Die von V erzeugte freie sig-Algebra

Definition 6.7.

- $\mathfrak{A} = (A, F_{\mathfrak{A}})$ *mit:* $A = \bigcup_{s \in S} A_s$
  $A_s = \mathrm{Term}_s(F, V)$, *d. h.* $A = \mathrm{Term}(F, V)$
  $F \ni f : s_1, \ldots, s_n \to s$*:*
  $f_{\mathfrak{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$

  $\mathfrak{A}$ *ist sig-Algebra::* $T_{\mathrm{sig}}(V)$ *die von* V *erzeugte freie Termalgebra mit Variablenmenge* V*.*
- $V = \varnothing$*:* $A_s = \mathrm{Term}_s(F)$ *Menge der Grundterme* ($A_s \neq \varnothing$*, da sig strikt).*

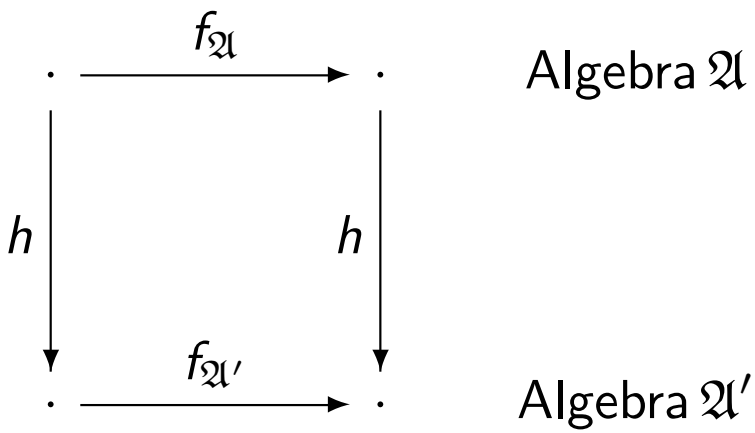
  $\mathfrak{A}$ *Grundtermalgebra::* T_{sig}

Homomorphismen

Definition 6.8. *sig-Homomorphismus:* $\mathfrak{A}, \mathfrak{A}'$ *sig-Algebren*
$h : \mathfrak{A} \to \mathfrak{A}'$ *Familie von Abbildungen.*
$h = \{h_s : A_s \to A'_s : s \in S\}$ *ist sig-Homomorphismus.*
Wenn

$$h_s(f_{\mathfrak{A}}(a_1, \ldots, a_n)) = f_{\mathfrak{A}'}(h_{s_1}(a_1), \ldots, h_{s_n}(a_n))$$

Wie üblich: injektiv, surjektiv, bijektiv, Isomorphismus



Kanonische Homomorphismen

Lemma 6.9. *$\mathfrak{A}$ sig-Algebra, T_{sig}*

a) *Die Familie der Interpretationsfunktionen $h_s : \text{Term}_s(F) \to A_s$ ist definiert durch*

$$h_s(f(t_1, \ldots, t_n)) = f_{\mathfrak{A}}(h_{s_1}(t_1), \ldots, h_{s_n}(t_n))$$

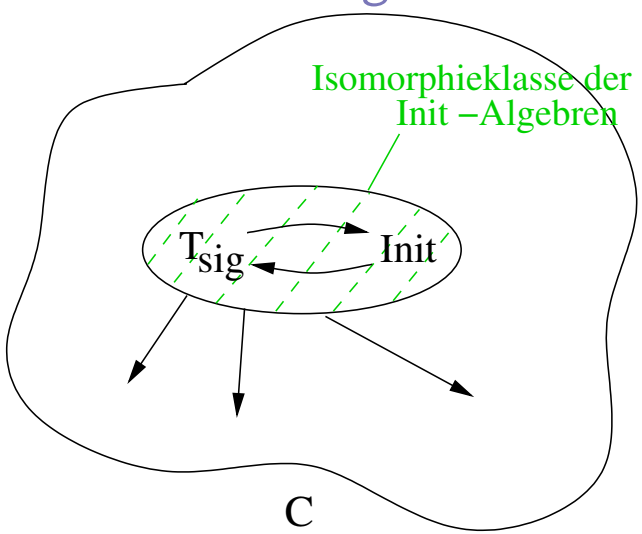
mit $h_s(c) = c_{\mathfrak{A}}$ ist ein sig-Homomorphismus.

b) *Es gibt keinen anderen sig-Homomorphismus von T_{sig} nach $\mathfrak{A}$. Eindeutigkeit!*

Beweis: Übung macht den Meister!

Initiale Algebren

Definition 6.10. *Initiale Algebren:*
Eine sig-Algebra $\mathfrak{A}$ heißt Initial in einer Klasse C von sig-Algebren, wenn für jede sig-Algebra $\mathfrak{A}' \in C$ genau ein sig-Homomorphismus $h : \mathfrak{A} \to \mathfrak{A}'$ existiert.
Insbesondere: T_{sig} ist initial in der Klasse aller sig-Algebren.
Fakt: Initiale Algebren sind isomorph.



Analog lassen sich Finale Algebren definieren.

Kanonische Homomorphismen

$\mathfrak{A}$ sig-Algebra, $h : T_{\text{sig}} \to \mathfrak{A}$ Interpretationshomomorphismus.

$\mathfrak{A}$ sig-erzeugt (Term-erzeugt) gdw
$\forall s \in S \quad h_s : \text{Term}_s(F) \to A_s$ surjektiv
Die (freie) Termalgebra ist sig-erzeugt.

ADT Forderungen:

- Repräsentationsunabhängig (Isomorphieklasse)
- Operationserzeugt (sig-erzeugt)

These: Ein ADT ist die Isomorphieklasse einer initialen Algebra.
Termalgebren als initiale Algebren sind ADT.
Beachte Eigenschaften der freien Termalgebra:
Abbildungen von V in $\mathfrak{A}$ lassen sich eindeutig zu Homomorphismen von $T_{sig}(V)$ in $\mathfrak{A}$ fortsetzen.

Gleichungsspezifikationen

Für Spezifikationsformalismen:
Klassen von Algebren die initialen Algebren zulassen.

$$\rightsquigarrow \text{ Horn (Siehe Literatur)}$$

sig INT sorts int
ops $0 :\longrightarrow$ int
$\text{suc} : \text{int} \longrightarrow \text{int}$
$\text{pred} : \text{int} \longrightarrow \text{int}$

Gleichungsspezifikation

Definition 6.11. $\mathrm{sig} = (S, F, \tau)$ *Signatur,* V *Variablensystem*

a) *Gleichung:* $(u, v) \in \mathrm{Term}_s(F, V) \times \mathrm{Term}_s(F, V)$

Schreibe: $u = v$

Gleichungssystem E *über sig*, V*: Menge von Gleichungen*

b) *(Gleichungs)-Spezifikation:* $\mathrm{spec} = (\mathrm{sig}, E)$
E *Gleichungssystem über* $F \cup V$*.*

Notation

Schlüsselwort eqns

spec INT		
sorts int		implizite
ops	$0 :\longrightarrow \text{int}$	All-Quantifikation
	$\text{suc}, \text{pred}: \text{int} \longrightarrow \text{int}$	oft Deklaration
eqns	$\text{suc}(\text{pred}(x)) = x$	der Sorten
	$\text{pred}(\text{suc}(x)) = x$	der Variablen

Semantik::

- loose alle Modelle (PL1)
- enge (spezielles Modell initial, final)
- operational (Gleichungskalkül+Induktionsprinzip)

Modelle von spec = (sig, E)

Definition 6.12. *$\mathfrak{A}$ sig-Algebra, $V(S)$-Variablensystem*

a) *Belegungsfunktion φ für $\mathfrak{A}$: $\varphi_s : V_s \to A_s$* *Induziert*

Bewertung $\varphi : \mathrm{Term}(F, V) \to \mathfrak{A}$ *durch*

$\varphi(f) = f_{\mathfrak{A}}$, *f konstant*, $\varphi(x) := \varphi_s(x)$, $x \in V_s$
$\varphi(f(t_1, \ldots, t_n)) = f_{\mathfrak{A}}(\varphi(t_1), \ldots, \varphi(t_n))$

$$\begin{array}{rcll} V_s & \xrightarrow{\varphi_s} & A_s & \\ \mathrm{Term}_s(F, V) & \xrightarrow{\varphi_s} & A_s & \\ \mathrm{Term}(F, V) & \xrightarrow{\varphi} & \mathfrak{A} & \textit{Homomorphismus} \end{array}$$

Beweis!

Modelle von spec $= (\text{sig}, E)$

b) $s = t$ Gleichung über sig, V
$\mathfrak{A} \models_{\varphi} s = t$: $\mathfrak{A}$ erfüllt $s = t$ mit Belegung φ gdw $\varphi(s) = \varphi(t)$,
Gleichheit in A.

c) $\mathfrak{A}$ erfüllt $s = t$ bzw. $s = t$ gilt in $\mathfrak{A}$
$\mathfrak{A} \models s = t$:
Für jede Belegung φ $\mathfrak{A} \models_{\varphi} s = t$

d) $\mathfrak{A}$ ist Modell von spec $= (\text{sig}, E)$
gdw $\mathfrak{A}$ erfüllt jede Gleichung von E
$\mathfrak{A} \models E$ ALG(spec) Klasse der Modelle von spec.

Beispiele

Beispiel 6.13. *1)*

spec	NAT
sorts	nat
ops	$0 :\longrightarrow \text{nat}$
	$s : \text{nat} \longrightarrow \text{nat}$
	$\_ + \_ : \text{nat}, \text{nat} \longrightarrow \text{nat}$
eqns	$x + 0 = x$
	$x + s(y) = s(x + y)$

Beispiele

sig-Algebren

a) $\mathfrak{A} = (\mathbb{N}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = 0 \quad \hat{s}(n) = n + 1 \quad n \hat{+} m = n + m$

b) $\mathfrak{B} = (\mathbb{Z}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = 1 \quad \hat{s}(i) = i \cdot 5 \quad i \hat{+} j = i \cdot j$

c) $\mathfrak{C} = (\{\mathsf{true}, \mathsf{false}\}, \hat{0}, \hat{+}, \hat{s})$
$\hat{0} = \mathsf{false} \quad \hat{s}(\mathsf{true}) = \mathsf{false} \quad \hat{s}(\mathsf{false}) = \mathsf{true}$
$i \hat{+} j = i \vee j$

Beispiele

$\mathfrak{A}, \mathfrak{L}, \mathfrak{C}$ sind Modelle von spec NAT

z.B. $\mathfrak{B}$: $\varphi(x) = a \quad \varphi(y) = b \quad a, b \in \mathbb{Z}$

$$\varphi(x + 0) = a \hat{+} \hat{0} = a \cdot 1 = a = \varphi(x)$$

$$\begin{aligned} \varphi(x + s(y)) &= a \hat{+} \hat{s}(b) = a \cdot (b \cdot 5) \\ &= (a \cdot b) \cdot 5 = \hat{s}(a \hat{+} b) \\ &= \varphi(s(x + y)) \end{aligned}$$

Beispiele

2)

spec	LIST(NAT)
use	NAT
sorts	nat, list
ops	$\mathsf{nil} :\longrightarrow \mathsf{list}$
	$\_.\_ : \mathsf{nat}, \mathsf{list} \longrightarrow \mathsf{list}$
	$\mathsf{app} : \mathsf{list}, \mathsf{list} \longrightarrow \mathsf{list}$
eqns	$\mathsf{app}(\mathsf{nil}, q_2) = q_2$
	$\mathsf{app}(x.q_1, q_2) = x.\,\mathsf{app}(q_1, q_2)$

Beispiele

spec-Algebra

$$\mathfrak{A} \qquad \mathbb{N}, \mathbb{N}^*$$
$$\hat{0} = 0 \quad \hat{+} = + \quad \hat{s} = +1$$
$$\hat{\mathsf{nil}} = e \qquad (\text{leeresWort})$$
$$\hat{:}\ (i, z) = i\ z$$
$$\widehat{\mathsf{app}}(z_1, z_2) = z_1 z_2\ (\text{Konkatenation})$$

Beispiele

3) spec INT $\quad \mathrm{suc}(\mathrm{pred}(x)) = x \quad \mathrm{pred}(\mathrm{suc}(x)) = x$

	1	2	3
A_{int}	$\mathbb{Z}$	$\mathbb{N}$	$\{\mathrm{true}, \mathrm{false}\}$
$0_{\mathfrak{A}_i}$	0	0	true
$\mathrm{suc}_{\mathfrak{A}_i}$	$\mathrm{suc}_{\mathbb{Z}}$	$\mathrm{suc}_{\mathbb{N}}$	$\left\{ \begin{array}{c} \mathrm{true} \longrightarrow \mathrm{false} \\ \mathrm{false} \longrightarrow \mathrm{true} \end{array} \right\}$
$\mathrm{pred}_{\mathfrak{A}_i}$	$\mathrm{pred}_{\mathbb{Z}}$	$\left\{ \begin{array}{c} n+1 \longrightarrow n \\ 0 \longrightarrow 0 \end{array} \right\}$	$\left\{ \begin{array}{c} \mathrm{true} \longrightarrow \mathrm{false} \\ \mathrm{false} \longrightarrow \mathrm{true} \end{array} \right\}$
	+	−	+

Beispiele

	4	5	6
A_{int}	$\{a, b\}^* \cup \mathbb{Z}$	$\{1\}^+ \cup \{0\}^+ \cup \{z\}$	!
$0_{\mathfrak{A}_i}$	0	z	!
$\text{suc}_{\mathfrak{A}_i}$	$\text{suc}_{\mathbb{Z}}$	$\left\{ \begin{array}{c} 1^n \longrightarrow 1^{n+1} \\ z \longrightarrow 1 \\ 0^{n+1} \longrightarrow 0^n \\ 0 \longrightarrow z \end{array} \right\}$	id
$\text{pred}_{\mathfrak{A}_i}$	$\text{pred}_{\mathbb{Z}}$	$\left\{ \begin{array}{c} 1^{n+1} \longrightarrow 1^n \\ 1 \longrightarrow z \\ z \longrightarrow 0 \\ 0^n \longrightarrow 0^{n+1} \end{array} \right\}$	id
	$-$	$+$	$+$

Substitution

Definition 6.14. *sig, Term(F, V) Eine Substitution ist eine Abbildung*
$\sigma :: \sigma_s : V_s \to \mathrm{Term}_s(F, V)$, $\sigma_s(x) \in \mathrm{Term}_s(F, V)$, $x \in V_s$
$\sigma(x) = x$ *für fast alle* $x \in V$

$D(\sigma) = \{x \mid \sigma(x) \neq x\}$ *endlich Definitionsbereich*
Schreibe $\sigma = \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$

Fortsetzung zu Homomorphismus $\sigma : \mathrm{Term}(F, V) \to \mathrm{Term}(F, V)$

$$\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots, \sigma(t_n))$$

Grundsubstitution: $t_i \in \mathrm{Term}_s(F) \quad x_i \in D(\sigma)_s$

Lose Semantik

Definition 6.15. spec $= (\text{sig}, E)$
ALG(spec)$= \{\mathfrak{A} \mid$ *sig-Algebra*, $\mathfrak{A} \models E\}$

Gesucht: Charakterisierungen der Gleichungen die in ALG(spec) bzw. $\text{ALG}_{\text{TE}}(\text{spec})$ *(Term erzeugte).*

a) *Semantische Gleichheit*: $E \models s = t$

b) *Operationale Gleichheit*: $t_1 \underset{E}{\vdash\!\dashv} t_2$ *gdw*

Es gibt $p \in 0(t_1), s = t \in E$, *Substitution* σ *mit*
$t_1|_p \equiv \sigma(s),\ t_2 \equiv t_1[\sigma(t)]_p \quad (t_1[p \leftarrow \sigma(t)])$
oder
$t_1|_p \equiv \sigma(t),\ t_2 \equiv t_1[\sigma(s)]_p$

$t_1 =_E t_2$ gdw $t_1 \overset{*}{\underset{E}{\vdash\!\dashv}} t_2$

Formalisierung von: Ersetze Gleiches $\leftrightarrow$ *Gleiches*

Gleicheitskalkül

c) Gleichheitskalkül: Inferenzregeln (deduktiv)

Reflexivität $\dfrac{}{t = t}$

Symmetrie $\dfrac{t = t'}{t' = t}$

Transitivität $\dfrac{t = t', t' = t''}{t = t''}$

Ersetzung $\dfrac{t' = t''}{s[t']_p = s[t'']_p} \quad p \in 0(s)$

(oft auch mit Substitution σ)

Gleicheitskalkül

$E \vdash s = t$ gdw es gibt einen Beweis für $s = t$ aus E, d. h.

$P =$ Folge von Gleichungen die mit $s = t$ endet, wobei für $t_1 = t_2 \in P$ gilt.

i) $t_1 = t_2 \in \sigma(E)$ für ein σ: Substitution

ii) $t_1 = t_2 \ldots$ aus vorangehenden Gleichungen durch Anwendung einer der Inferenzregeln.

Eigenschaften und Beispiele

Folgerung 6.16. *Eigenschaften und Beispiele*

a) *Gilt* $E \models s = t$ *oder* $s =_E t$ *oder* $E \vdash s = t$

i) *Ist* σ *eine Substitution, so auch*

$E \models \sigma(s) = \sigma(t)$ / $\sigma(s) =_E \sigma(t)$ / $E \models \sigma(s) = \sigma(t)$
d. h. die induzierten Äquivalenzrelationen auf $\mathrm{Term}(F, V)$ *sind stabil bzgl. Substitutionen*

ii) $r \in \mathrm{Term}(F, V)$, $p \in 0(r)$, $r|_p$, $s, t \in \mathrm{Term}_{s'}(F, V)$ *so*

$E \models r[s]_p = r[t]_p$ / $r[s]_p =_E r[t]_p$ / $E \vdash r[s]_p = r[t]_p$
Monotonie Eigenschaft

Kongruenz auf $\mathrm{Term}(F, V)$ *die stabil ist.*

Kongruenzen / Quotientenalgebren

b) $\mathfrak{A} = (A, F_{\mathfrak{A}})$ sig-Algebra. $\sim$ bin. Relation auf A ist Kongruenzrelation auf $\mathfrak{A}$, falls

i) $a \sim b \rightsquigarrow \exists s \in S : a, b \in A_s$ (Sortentreu)
ii) $\sim$ ist Äquivalenzrelation
iii) $a_i \sim b_i$ $(i = 1, \ldots, n)$, $f_{\mathfrak{A}}(a_1, \ldots, a_n)$ definiert
$\rightsquigarrow f_{\mathfrak{A}}(a_1, \ldots, a_n) \sim f_{\mathfrak{A}}(b_1, \ldots, b_n)$ (monoton)

$\mathfrak{A}/\sim$ Quotientenalgebra:
$A/\sim = \bigcup_{s \in S}(A_s/\sim)_s$ mit $(A_s/\sim)_s = \{[a]_\sim : a \in A_s\}$ und $f_{\mathfrak{A}/\sim}$ mit
$f_{\mathfrak{A}/\sim}([a_1], \ldots, [a_n]) = [f_{\mathfrak{A}}(a_1, \ldots, a_n)]$

wohldefiniert, d. h. $\mathfrak{A}/\sim$ ist sig-Algebra.

$\varphi : \mathfrak{A} \to \mathfrak{A}_\sim$ mit $\varphi_s(a) = [a]_\sim$ ist surjektiver Homomorphismus, kanonischer Homomorphismus.

Zusammenhänge zwischen $\models, =_E, \vdash_E$

c) $\mathfrak{A}, \mathfrak{A}'$ sig-Algebren $\varphi : \mathfrak{A} \to \mathfrak{A}'$ surjektiver Homomorphismus.
Dann $\mathfrak{A} \models s = t \rightsquigarrow \mathfrak{A}' \models s = t$

d) spec $=$ (sig, E):

$$s =_E t \text{ gdw } E \vdash s = t$$

e) $\mathfrak{A}$ sig-Algebra, R eine sortentreue bin. Relation auf $\mathfrak{A}$. Dann gibt es eine kleinste Kongruenz $\equiv_R$ auf $\mathfrak{A}$ die R enthält, d. h. $R \subseteq \equiv_R$
$\equiv_R$ die von R erzeugte Kongruenz

Zusammenhänge zwischen $\models, =_E, \vdash_E$

f) $\mathfrak{A}$ sig-Algebra, E Gleichungssystem über (sig, V). E induziert eine Relation $\underset{E,\mathfrak{A}}{\sim}$ auf $\mathfrak{A}$ wobei

$a \underset{E,\mathfrak{A},s}{\sim} a'$ $(a, a' \in A_s)$ gdw es gibt $t = t' \in E$ und Belegung $\varphi : V \to \mathfrak{A}$ mit $\varphi(t) = a$, $\varphi(t') = a'$

Diese Relation ist sortentreu.

Fakt: Sei $\equiv$ eine Kongruenz auf $\mathfrak{A}$ die $\underset{E,\mathfrak{A}}{\sim}$ enthält, dann ist $\mathfrak{A}/\equiv$ eine spec $= (\text{sig}, E)$-Algebra, d. h. Modell von E.

g) Existenz: $\mathfrak{A} = T_{\text{sig}}$ die Termalgebra, dann ist $=_E$ auf T_{sig} die kleinste Kongruenz die $\underset{E,\mathfrak{A}}{\sim}$ umfasst.

Insbesondere ist $T_{\text{sig}}/=_E$ ein Modell von E.

Beispiel

spec :: INT mit $\text{pred}(\text{suc}(x)) = x$, $\text{suc}(\text{pred}(x)) = x$

$$
\begin{aligned}
(T_{\text{INT}}/=_E)_{\text{int}} = \quad & \{[0] = \{0, \text{pred}(\text{suc}(0)), \text{suc}(\text{pred}(0)), \ldots \\
& [\text{suc}(0)] = \{\text{suc}(0), \text{pred}(\text{suc}(\text{suc}(0))), \ldots \\
& [\text{suc}(\text{suc}(0))] = \{\cdots \\
& [\text{pred}(0)] = \{\text{pred}(0), \text{suc}(\text{pred}(\text{pred}(0))) \ldots
\end{aligned}
$$

$$
\begin{aligned}
\text{suc}_{T_{\text{INT}}/=_E} \quad & ([\text{pred}(\text{suc}(0))]) = [\text{suc}(\text{pred}(\text{suc}(0)))] \\
& = [\text{suc}(0)] \\
& = \text{suc}_{T_{\text{INT}}/=_E}([0])
\end{aligned}
$$

Satz von Birkhoff

Satz 6.17. *Birkhoff*
Für jede Spezifikation $spec = (sig, E)$ *gilt*

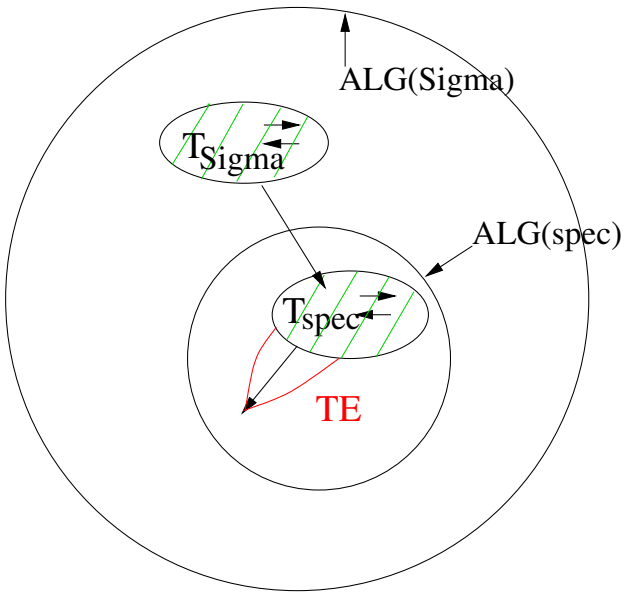
$$E \models s = t \text{ gdw } E \vdash s = t \quad (\text{d. h. } s =_E t)$$

Definition 6.18. *Initiale Semantik*
Sei $spec = (sig, E)$ *die Algebra* $T_{sig}/ =_E$ *(Quotiententermalgebra)*
($=_E$ *die von* E *erzeugte kleinste Kongruenzrelation auf* T_{sig}*)*
wird als initiale Algebra Semantik von $spec = (sig, E)$ *definiert.*

Sie ist operationserzeugt und initial in $ALG(spec)$*!*

Initiale Algebra Semantik

Initiale Algebra Semantik ordnet jeder Gleichungsspezifikation spec die Isomorphieklasse der (initialen) Quotiententermalgebra $T_{sig}/=_E$ zu.
Schreibe: T_{spec} oder $I(E)$



$$sig = \Sigma,\ spec = (\Sigma, E)$$

Quotiententermalgebren

Quotiententermalgebren sind ADT.

Beispiel 7.1. *(Fortsetzung)* $spec = INT$

A^i_{int}	$\mathbb{Z}$	$\{true, false\}$	$\{1\}^+ \cup \{0\}^+ \cup \{z\}$
0_{A^i}	0	$true$	z
suc_{A^i}	$\text{suc}_{\mathbb{Z}}$	not	$\ldots$
pred_{A^i}	$\text{pred}_{\mathbb{Z}}$	not	$\ldots$

$T_{\text{INT}}/=_E$ $[0] \mapsto true$ $[\text{suc}^{2n}(0)] \mapsto true$
$[\text{suc}^{2n+1}(0)] \mapsto false$ $[\text{pred}^{2n+1}(0)] \mapsto false$
$[\text{pred}^{2n}(0)] \mapsto true$

Initiale Algebra

spec $= (\text{sig}, E)$ Initiale Algebra $T_{\text{spec}} \quad (I(E))$
Probleme:

- Ist T_{spec} berechenbar?
- Wortproblem $(T_{\text{sig}}, =_E)$ lösbar?
- Operationalisierung von T_{spec}?
- Welche (PL1-) Eigenschaften gelten in T_{spec} ?
- Wie beweist man diese Eigenschaften?

Gleichheitstheorie/Induktive Theorie

Definition 7.2. *Gleichungseigenschaften*

a) $TH(E) = \{s = t : E \models s = t\}$ *Gleichheitstheorie*
Gleichungen, die in allen spec-Algebren gelten.

b) $ITH(E) = \{s = t : T_{spec} \models s = t\}$ *induktive (=)-Theorie*
Gleichungen, die in allen termerzeugten spec-Algebren gelten.

Gleichheitstheorie/Induktive Theorie

Folgerung 7.3. *Einfache Eigenschaften*

a) $TH(E) \subseteq ITH(E)$, *da* T_{spec} *Modell von* E.

b) *i. Allg.* $TH(E) \subsetneq ITH(E)$
$= \quad$ so E *ω-vollständig*
$\rightsquigarrow$ *Konsistenzbeweise Inductionless Induction*
E *r.a., so* $TH(E)$ *r.a., aber* $ITH(E)$ *i. Allg. nicht r.a.*

c) $T_{spec} \models s = t \text{ gdw } \sigma(s) =_E \sigma(t)$ *für jede Grundsubstitution der Var. in* s, t. $\rightsquigarrow$ *Induktive Beweismethoden, Coversetinduction*

d) $E : x + 0 = x \qquad x + s(y) = s(x + y)$
$\rightsquigarrow x + y = y + x \in ITH(E) - TH(E)$
$(x + y) + z = x + (y + z)$ *Beweis !*

Beispiele

Beispiel 7.4. *Einfache Beispiele*

a) spec	BOOL
sorts	bool
ops	$true, false :\longrightarrow$ bool
	not : bool $\longrightarrow$ bool
	and, or, impl, eqv : bool, bool $\longrightarrow$ bool
	if _ then _ else _ : bool, bool, bool $\longrightarrow$ bool

Beispiel (Forts.)

$$
\begin{array}{ll}
\underline{\text{eqns}} & \text{not}(\text{true}) = \text{false} \\
& \text{not}(\text{false}) = \text{true} \\
& \text{and}(\text{true}, b) = b \\
& \text{and}(\text{false}, b) = \text{false} \\
& \text{or}(b, b') = \text{not}(\text{and}(\text{not}(b), \text{not}(b'))) \\
& \text{impl}(b, b') = \text{or}(\text{not}(b), b') \\
& \text{eqv}(b, b') = \text{and}(\text{impl}(b, b'), \text{impl}(b, b')) \\
& \text{if true } b' \text{ else } b'' = b' \\
& \text{if false } b' \text{ else } b'' = b''
\end{array}
$$

$(T_{\text{BOOL}})_{\text{bool}} = \{[\text{true}], [\text{false}]\}$ (Beweis!)

$\rightsquigarrow$ Definierte- und Konstruktor-Funktionen.

Beispiel (Forts.)

b) spec	SET-OF-CHARACTERS
sorts	char, set
ops	$a, b, c, \cdots :\longrightarrow \text{char}$
	$\varnothing :\longrightarrow \text{set}$
	$\text{insert} : \text{char}, \text{set} \longrightarrow \text{set}$
eqns	$\text{insert}(x, \text{insert}(x, s)) = \text{insert}(x, s)$
	$\text{insert}(x, \text{insert}(y, s)) = \text{insert}(y, \text{insert}(x, s))$

$(T_{\text{soc}})_{\text{char}} = \{a, b, c, \ldots\}$

$(T_{\text{soc}})_{\text{set}} = \{[\varnothing], [\text{insert}(a, \varnothing)], \ldots$

$\{\varnothing\}\{\text{insert}(a, \text{insert}(a, \ldots, \text{insert}(a, \varnothing)\}$

Beispiel (Forts.)

c)

spec NAT

sorts nat

ops $0 :\rightarrow \text{nat}$

$\text{suc} : \text{nat} \rightarrow \text{nat}$

$\_ + \_, \_ * \_ : \text{nat}, \text{nat} \rightarrow \text{nat}$

eqns $x + 0 = x$

$x + \text{suc}\, y = \text{suc}(x + y)$

$x * 0 = 0$

$x * \text{suc}(y) = (x * y) + x$

$(T_{\text{NAT}})_{\text{nat}} = \{\ [0, 0 + 0, 0 * 0, \ldots$

$[\text{suc}\, 0, 0 + \text{suc}\, 0, \ldots$

$[\text{suc}(\text{suc}(0)), \ldots$

Beispiel (Forts.)

d) Binärbäume

spec BIN-TREE

sorts nat, tree

ops $0 :\longrightarrow \text{nat}$

$\text{suc} : \text{nat} \longrightarrow \text{nat}$

$\text{max} : \text{nat}, \text{nat} \longrightarrow \text{nat}$

$\text{leaf} :\longrightarrow \text{tree}$

$\text{left} : \text{tree} \longrightarrow \text{tree}$

$\text{right} : \text{tree} \longrightarrow \text{tree}$

$\text{both} : \text{tree}, \text{tree} \longrightarrow \text{tree}$

$\text{height} : \text{tree} \longrightarrow \text{nat}$

$\text{dleft} : \text{tree} \longrightarrow \text{tree}$

$\text{dright} : \text{tree} \longrightarrow \text{tree}$

Beispiel

Fortsetzung zu d) Binärbäume

$$
\begin{array}{ll}
\underline{\text{eqns}} & \max(0, n) = n \\
& \max(n, 0) = n \\
& \max(\text{suc}(m), \text{suc}(n)) = \text{suc}(\max(m, n)) \\
& \text{height}(\text{leaf}) = 0 \\
& \text{height}(\text{both}(t, t')) = \text{suc}(\max(\text{height}(t), \text{height}(t'))) \\
& \text{height}(\text{left}(t)) = \text{suc}(\text{height}(t)) \\
& \text{height}(\text{right}(t)) = \text{suc}(\text{height}(t))
\end{array}
$$

Korrektheit

Definition 7.5. *Eine Spezifikation* $spec = (sig, E)$ *ist sig-korrekt bzgl. einer sig-Algebra* $\mathfrak{A}$ *gdw* $T_{spec} \cong \mathfrak{A}$ *(d. h. der eindeutige Hom. ist Bijektion).*

Beispiel 7.6. *Anwendung:*
INT korrekt für $\mathbb{Z}$, *BOOL korrekt für* $\mathbb{B}$

Beachte: Begriff ist hier auf initiale Semantik beschränkt!

Einschränkungen/Vergißbilder

Definition 7.7. *Einschränkungen/Vergißbilder*

a) *$sig = (S, F, \tau)$, $sig' = (S', F', \tau')$ Signaturen mit $sig \subseteq sig'$, d. h. $(S \subseteq S', F \subseteq F', \tau \subseteq \tau')$.*

Für jede sig'-Algebra $\mathfrak{A}$ sei sig-Anteil $\mathfrak{A}|_{sig}$ von $\mathfrak{A}$ die sig-Algebra mit

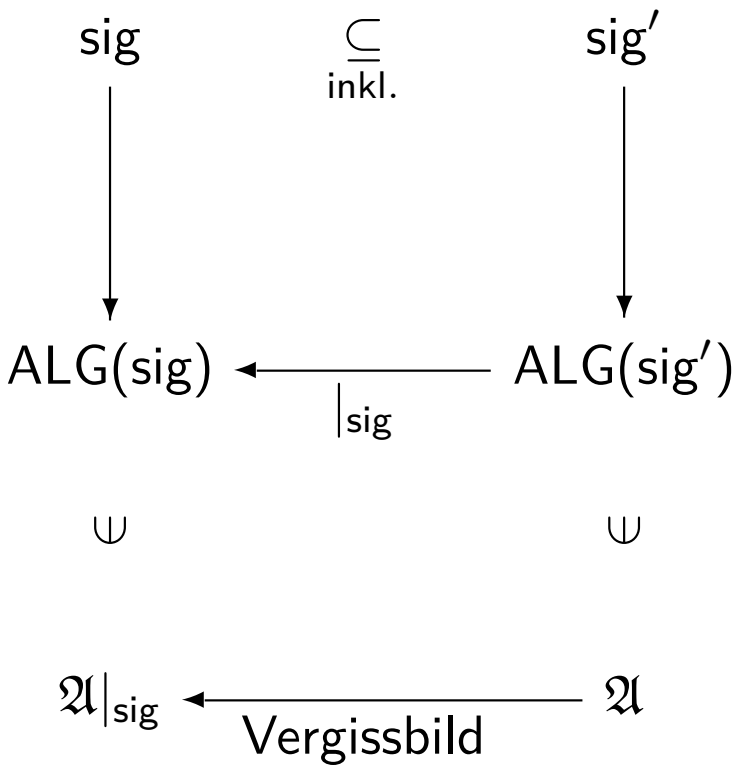
i) $(\mathfrak{A}|_{sig})_s = A_s$ *für* $s \in S$
ii) $f_{\mathfrak{A}|_{sig}} = f_{\mathfrak{A}}$ *für* $f \in F$

Beachte: $\mathfrak{A}|_{\text{sig}}$ ist sig - Algebra.

$\mathfrak{A}|_{\text{sig}}$ heißt auch Vergißbild von $\mathfrak{A}$ (bzgl. sig).

Einschränkungen/Vergißbilder

$\mathfrak{A}|_{\text{sig}}$ Vergißbild von $\mathfrak{A}$ (bzgl. sig). Vergißbild induziert somit eine Abbildung zwischen Algebrenklassen wie folgt:



Einschränkungen/Vergißbilder

b) Eine Spezifikation $\text{spec} = (\text{sig}', E)$ mit $\text{sig} \subseteq \text{sig}'$ ist korrekt bzgl. sig-Algebra $\mathfrak{A}$ gdw

$$(T_{\text{spec}})|_{\text{sig}} \cong \mathfrak{A}$$

(Der Hom. von $(T_{\text{spec}})|_{\text{sig}}$ nach $\mathfrak{A}$ ist Bijektion).

c) Eine Spezifikation $\text{spec}' = (\text{sig}', E')$ implementiert eine Spezifikation $\text{spec} = (\text{sig}, E)$ gdw

$$\text{sig} \subseteq \text{sig}' \text{ und } (T_{\text{spec}'})|_{\text{sig}} \cong T_{\text{spec}}$$

Beachte:

- Konsistenz-Begriff für $=$-Spezifikation nicht notwendig. (Modelle existieren immer!).
- Allgemeiner Implementierungsbegriff ($Cl(\text{spec}) \subseteq Cl(\text{spec}')$) reduziert sich immer auf $=$.
  "vollständige„ Theorien.

Probleme

Nachweis $s = t \in Th(E)$ oder $\in ITH(E)$.

Für $Th(E)$ finde $=_E$ äquivalentes konvergentes Termersetzungssystem (Siehe Gruppenbeispiel).

Für $ITH(E)$ Induktionsmethoden:

s, t induzieren Funktionen auf T_{spec}. Sind $x_1, \ldots, x_n$ die Variablen in s und t, Sorten $s_1, \ldots, s_n$.
$s : (T_{\text{spec}})_{s_1} \times \cdots \times (T_{\text{spec}})_{s_n} \longrightarrow (T_{\text{spec}})_s$
$s = t \in ITh(E)$ gdw s und t induzieren gleiche Funktionen $\rightsquigarrow$ Beweise durch Induktion über Aufbau der Terme.
NAT $0, \text{suc}, +$ $x + y = y + x \in ITH$
$0 + x = x$

Probleme

- $0 + 0 = 0 \qquad \text{Ann} : 0 + a = a$
  $0 + Sa =_E S(0 + a) =_I S(a)$
- $x + 0 = 0 + x \qquad \text{Ann} : x + a = a + x$
  $x + Sa =_E S(x + a) =_I S(a + x) =_E a + Sx \stackrel{?}{=} Sa + x$
- $x + Sy = Sx + y$
  $x + So =_E S(x + o) =_E Sx =_E Sx + o$
  $x + SSa =_E S(x + Sa) =_I S(Sx + a) = Sx + Sa$

$\text{spec}(\text{sig}, E)$	$P_{\text{spec}}(\text{sig}, E, Prop)$
Gleichungen reichen	Eigenschaften die gelten sollen!
oft nicht aus	Verifikationsaufgaben

Strukturierungsmechanismen

Horizontal: - Zerlegung, - Kombination,
- Erweiterung, - Instantiierung
Vertikal: - Realisierung, - Information hiding,
- Vertikale Komposition

Hier:
Kombination, Anreicherung, Erweiterung,
Modularisierung, Parametrisierung
$\rightsquigarrow$ **Wiederverwendbarkeit.**

Strukturierungsmechanismen

BIN-TREE

1)

spec	NAT	
sorts	nat	
ops	$0 :\longrightarrow \text{nat}$	
	$\text{suc} : \text{nat} \longrightarrow \text{nat}$	

2)

spec	NAT1
use	NAT
ops	$\max : \text{nat}, \text{nat} \longrightarrow \text{nat}$
eqns	$\max(0, n) = n$
	$\max(n, 0) = n$
	$\max(s(m), s(n)) =$
	$s(\max(m, n))$

Strukturierungsmechanismen

BIN-TREE (Forts.)

3) spec BINTREE1
sorts bintree
ops leaf :⟶ bintree

left, right : bintree
⟶ bintree
both : bintree, bintree
⟶ bintree

4) spec BINTREE2
use NAT1, BINTREE1
ops height : bintree ⟶ nat

eqns :

Kombination

Definition 7.8. *Sei* $spec_1 = (sig_1, E_1)$, *mit* $sig_1 = (S_1, F_1, \tau_1)$ *Signatur und* $sig_2 = [S_2, F_2, \tau_2]$ *Tripel,* E_2 *Gleichungsmenge.*

$comb = spec_1 + (sig_2, E_2)$ *heißt Kombination*
gdw
$spec = ((S_1 \cup S_2), (F_1 \cup F_2), (\tau_1 \cup \tau_2)), E_1 \cup E_2)$ *eine Spezifikation ist.*

Insbesondere ist $((S_1 \cup S_2), (F_1 \cup F_2), (\tau_1 \cup \tau_2))$ *eine Signatur und* E_2 *enthält „syntaktisch korrekte“ Gleichungen.*

Die Semantik von comb: $T_{comb} := T_{spec}$

Die Semantik von comb

$T_{\text{comb}} := T_{\text{spec}}$

Typische Fälle:

$S_2 = \varnothing$, F_2 neue Funktionssymbole mit Stelligkeiten τ_2 (alte).

S_2 neue Sorten, F_2 neue Funktionssymbole.
τ_2 Stelligkeiten neue + alte.

E_2 nur „neue“ Gleichungen.

Notationen: use, include (protected)

Beispiel

Beispiel 7.9.

a) *Schrittweiser Entwurf der ganzen Zahlen*

		Semantik
spec	INT1	
sorts	int	$T_{\text{INT1}} \cong (\mathbb{N}, 0, \text{suc}_{\mathbb{N}})$
ops	$0 :\longrightarrow \text{int}$	
	$\text{suc} : \text{int} \longrightarrow \text{int}$	
	$\cap$	$\cap$
spec	INT2	
use	INT1	$T_{\text{INT2}} \cong (\mathbb{Z}, 0, \text{suc}_{\mathbb{Z}}, \text{pred}_{\mathbb{Z}})$
ops	$\text{pred} : \text{int} \longrightarrow \text{int}$	
eqns	$\text{pred}(\text{suc}(x)) = x$	
	$\text{suc}(\text{pred}(x)) = x$	

Beispiel (Forts.)

Frage: Stimmt der INT1-Anteil von T_{INT2} mit T_{INT1} überein??
Implementiert INT2 INT1?

$$(T_{\mathsf{INT2}})|_{\mathsf{INT1}} \cong T_{\mathsf{INT1}}$$

$$(\mathbb{Z}, 0, \mathsf{suc}_{\mathbb{Z}}, \mathsf{pred}_{\mathbb{Z}})|_{\mathsf{INT1}}$$
$$\shortparallel$$
$$(\mathbb{Z}, 0, \mathsf{suc}_{\mathbb{Z}}) \qquad \not\cong (\mathbb{N}, 0, \mathsf{suc}_{\mathbb{N}})$$

Vorsicht: Nicht immer erfasst man das richtige!
Hier wurden neue Daten von der Sorte int eingeführt.

Beispiel (Forts.)

b) spec NAT2
use NAT
eqns $\mathsf{suc}(\mathsf{suc}(x)) = x$

$$(T_{\mathsf{NAT2}})|_{\mathsf{NAT}} = (\mathbb{N} \bmod 2)|_{\mathsf{NAT}} = \mathbb{N} \bmod 2 \ncong \mathbb{N} = T_{\mathsf{NAT}}$$

Problem: Hinzufügen neuer bzw. Identifizieren alter Elemente.

Probleme mit der Kombination

Sei

$$\text{comb} = \text{spec}_1 + (\text{sig}, E)$$

$$\left.\begin{array}{l} (T_{\text{comb}})|_{\text{spec}_1} \text{ ist } \text{spec}_1 \text{ Algebra} \\ T_{\text{spec}_1} \text{ ist initiale } \text{spec}_1 \text{ Algebra} \end{array}\right\} \rightsquigarrow$$

$\exists!$ Homomorphismus $h : T_{\text{spec}_1} \to (T_{\text{comb}})|_{\text{spec}_1}$

Eigenschaften von
h: nicht injektiv / nicht surjektiv / bijektiv.

z. B. $(T_{\text{BINTREE2}})|_{\text{NAT}} \cong T_{\text{NAT}}$.

Erweiterung und Anreicherung

Definition 7.10.

a) *Eine Kombination* $\mathrm{comb} = spec_1 + (sig, E)$ *ist eine Erweiterung gdw*

$$(T_{\mathrm{comb}})|_{spec_1} \cong T_{spec_1}$$

b) *Eine Erweiterung heißt Anreicherung, wenn sig keine neuen Sorten enthält, d. h.* $sig = [\varnothing, F_2, \tau_2]$

- *hinreichende Bedingungen die garantieren, dass eine Kombination eine Erweiterung ist?*

Parametrisierung

Definition 7.11. *Eine parametrisierte Spezifikation Parameter=(Formal, Body) besteht aus zwei Spezifikationen: Formal und Body mit Formal $\subseteq$ Body.*

D. h. Formal=(sig_F, E_F), Body=(sig_B, E_B), wobei
$sig_F \subseteq sig_B \qquad E_F \subseteq E_B$.

Notation: Body[Formal]

Syntaktisch: Body = Formal $+(sig', E')$ *Kombination*

Beachte: Es wird i.A. nicht verlangt, dass Formal oder Body[Formal] eine initiale Semantik besitzen. Es müssen in der Regel keine Grundterme der Sorten in Formal existieren. Erst wenn eine konkrete Spezifikation "eingesetzt" wird muss dies verlangt werden.

Beispiel

Beispiel 7.12.

spec	ELEM
sorts	elem
ops	next : elem $\longrightarrow$ elem

$(T_{spec})_{\text{elem}} = \varnothing$

spec	STRING[ELEM]
use	ELEM
sorts	string
ops	empty :$\longrightarrow$ string
	unit : elem $\longrightarrow$ string
	concat : string, string $\longrightarrow$ string
	ladd : elem, string $\longrightarrow$ string
	radd : string, elem $\longrightarrow$ string

$(T_{\text{spec}})_{\text{string}} = \{[\text{empty}]\}$

Beispiel (Forts.)

$$
\begin{aligned}
\text{eqns}\quad & \text{concat}(s, \text{empty}) = s \\
& \text{concat}(\text{empty}, s) = s \\
& \text{concat}(\text{concat}(s_1, s_2), s_3) = \text{concat}(s_1, \text{concat}(s_2, s_3)) \\
& \text{ladd}(e, s) = \text{concat}(\text{unit}(e), s) \\
& \text{radd}(s, e) = \text{concat}(s, \text{unit}(e))
\end{aligned}
$$

Parameterübergabe: ELEM $\longrightarrow$ NAT

$$\text{STRING[ELEM]} \longrightarrow \text{STRING[NAT]}$$

Zuordnung: formale Parameter $\longrightarrow$ aktuelle Parameter

$$
\begin{aligned}
S_F &\longrightarrow S_A \\
Op &\longrightarrow Op_A
\end{aligned}
$$

Zuordnung der Sorten und Funktionen, Semantik?

Signaturmorphismen - Parameterübergabe

Definition 7.13.

a) *Seien* $sig_i = (S_i, F_i, \tau_i)$ $i = 1, 2$ *Signaturen. Ein Paar* $\sigma = (g, h)$ *mit* $g : S_1 \to S_2, h : F_1 \to F_2$ *von Funktionen ist Signaturmorphismus, falls für alle* $f \in F_1$

$$\tau_2(hf) = g(\tau_1 f)$$

(g *fortgesetzt auf* $g : S_1^* \to S_2^*$).

im Bsp. g : elem $\to$ nat h :: next $\to$ suc

oder $\sigma : sig_{\text{BOOL}} \to sig_{\text{NAT}}$ *mit*

g ::	*bool* $\to$ nat	h ::	not $\to$ suc
h ::	*true* $\to$ 0		*and* $\to$ plus
	false $\to$ 0		*or* $\to$ times

Signaturmorphismen - Parameterübergabe

b) spec = Body[Formal] Parameterspezifikation
Eine Parameterübergabe ist ein Signaturmorphismus σ : sig(Formal) $\rightarrow$ sig(Actual) wobei Actual eine Spezifikation ist:
Die aktuelle Parameterspezifikation.

(Actual, σ) definiert eine Spezifikation VALUE durch folgende Änderungen von Body:

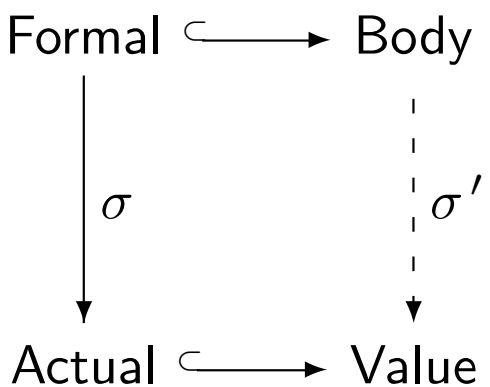
1) Ersetze Formal durch Actual: Body[Actual].
2) Ersetze in $op : s_1 \ldots s_n \rightarrow s_0 \in$ Body, das nicht in Formal auftritt jedes s_i, $s_i \in$ Formal durch $\sigma(s_i)$.
3) Ersetze in jeder Gleichung $L = R$ aus Body, die nicht in Formal ist, jedes $o_P \in$ Formal durch $\sigma(o_P)$.
4) Fasse jede Variable einer Sorte s mit $s \in$ Formal als Variable der Sorte $\sigma(s)$ auf.
5) Vermeide Namenskonflikte zwischen Actual und Body/Formal.

Parameterübergabe

Bezeichnung:

$$\text{Value} = \text{Body}[\text{Actual}, \sigma]$$

Man erhält somit für $\sigma : \text{sig}(\text{Formal}) \longrightarrow \text{sig}(\text{Actual})$ einen Signaturmorphismus
$\sigma' : \text{sig}(\text{Body}[\text{Formal}]) \longrightarrow \text{sig}(\text{Body}[\text{Actual}, \sigma]$



$$\sigma'(x) = \begin{cases} \sigma(x) & x \in \text{Formal} \\ x' & x \notin \text{Formal} \end{cases}$$

x' ist dabei eine Umbenennung, wenn Konflikte.

Signaturmorphismen (Forts.)

Definition 7.14. *Sei $\sigma : sig' \to sig$ Signaturmorphismus.*

Dann ist für jede sig-Algebra $\mathfrak{A}$, $\mathfrak{A}|_\sigma$ eine sig'-Algebra, wobei für $sig' = (S', F', \tau')$

$(A|_\sigma)_s = A_{\sigma(s)}$ $s \in S'$ *und* $f_{\mathfrak{A}|_\sigma} = \sigma(f)_\mathfrak{A}$ $f \in F'$.

$\mathfrak{A}|_\sigma$ *heißt Vergißbild von* $\mathfrak{A}$ *entlang* σ

(Spezialfall: $sig' \subseteq sig :\hookrightarrow$*)* $|_{sig'}$

Beispiel

Beispiel 7.15. $\mathfrak{A} = T_{\mathsf{NAT}}$ *(Mit* $0, \mathsf{suc}, \mathsf{plus}, \mathsf{times}$*)*
$sig' = sig(\mathsf{BOOL}) \quad sig = sig(\mathsf{NAT})$
$\sigma : sig' \rightarrow sig$ *wie oben definiert.*

$$\begin{aligned}((T_{\mathsf{NAT}})|_{sig})_{\mathsf{bool}} &= (T_{\mathsf{NAT}})_{\sigma(\mathsf{bool})} = (T_{\mathsf{NAT}})_{\mathsf{nat}} \\ &= \{[0], [\mathsf{suc}(0)], \ldots\}\end{aligned}$$

$$\begin{aligned}true_{(T_{\mathsf{NAT}})|_\sigma} &= \sigma(true)_{T_{\mathsf{NAT}}} = [0] \\ false_{(T_{\mathsf{NAT}})|_\sigma} &= \sigma(false)_{T_{\mathsf{NAT}}} = [0] \\ not_{(T_{\mathsf{NAT}})|_\sigma} &= \sigma(not)_{T_{\mathsf{NAT}}} = \mathsf{suc}_{T_{\mathsf{NAT}}} \\ and_{(T_{\mathsf{NAT}})|_\sigma} &= \sigma(and)_{T_{\mathsf{NAT}}} = \mathsf{plus}_{T_{\mathsf{NAT}}} \\ or_{(T_{\mathsf{NAT}})|_\sigma} &= \sigma(or)_{T_{\mathsf{NAT}}} = \mathsf{times}_{T_{\mathsf{NAT}}}\end{aligned}$$

Vergißbilder für Homomorphismen

Definition 7.16. *Ist $\sigma : sig' \to sig$ Signaturmorphismus, $\mathfrak{A}, \mathfrak{B}$ sig-Algebren und $h : \mathfrak{A} \to \mathfrak{B}$ sig-Homomorphismus, dann ist*

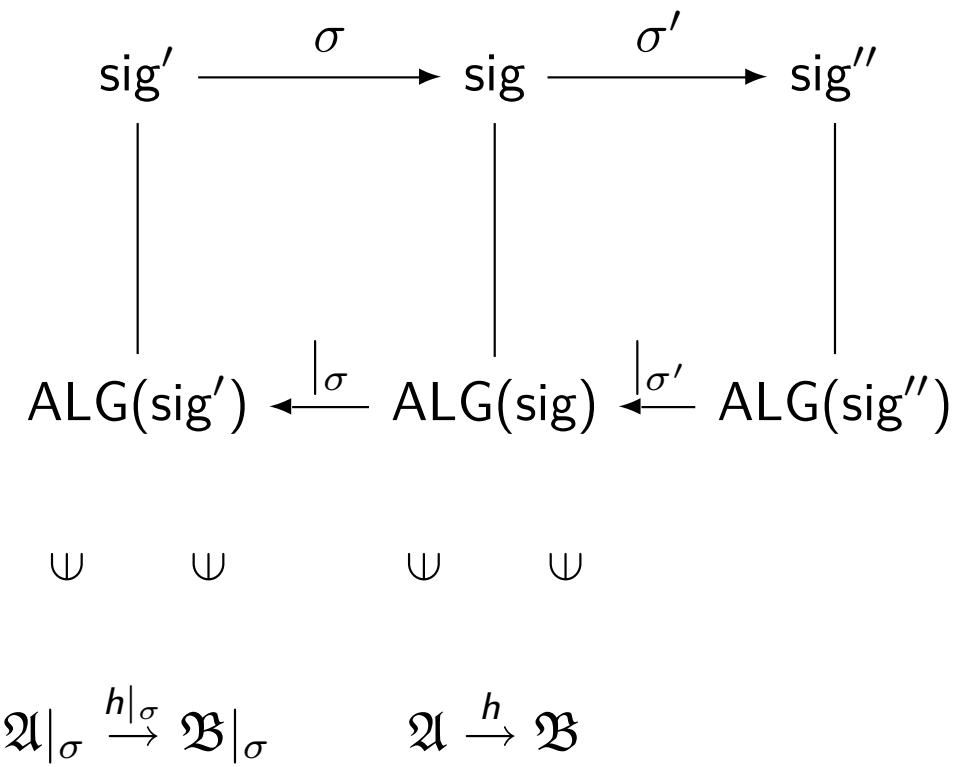
$h|_\sigma := \{h_{\sigma(s)} \mid s \in S'\}$, *wobei* $sig' = (S', F', \tau')$, *ein sig'-Homomorphismus:*

$$\begin{array}{rccc} (h|_\sigma)_s = h_{\sigma(s)} : & A_{\sigma(s)} & \to & B_{\sigma(s)} \\ & \| & & \| \\ & (A|_\sigma)_s & \to & (B|_\sigma)_s \end{array}$$

$h|_\sigma$ *heißt Vergißbild von h entlang* σ

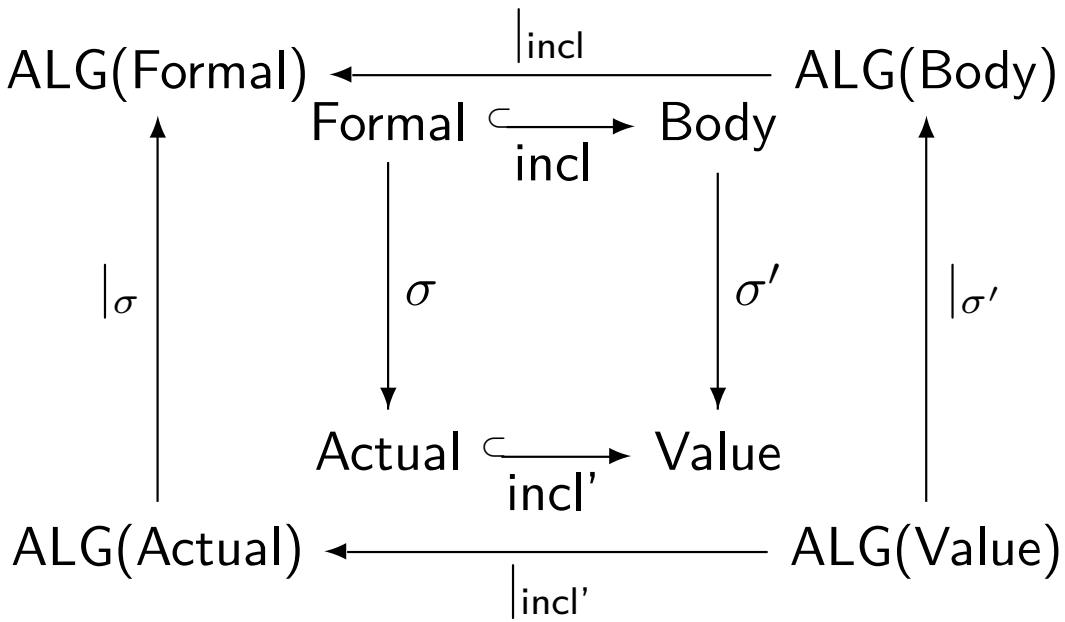
Vergißbilder

Eigenschaften von $h|_\sigma$ (Vergißbild von h entlang σ)



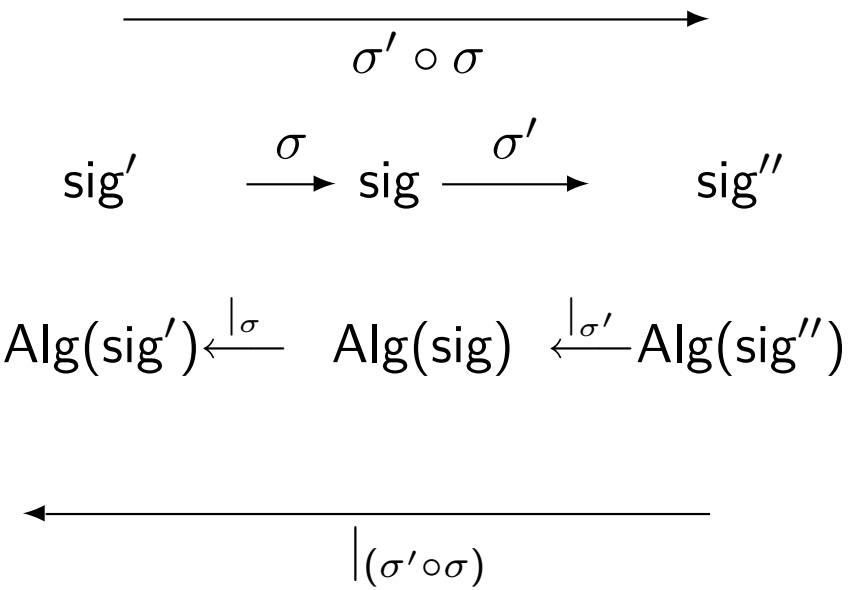
Verträglichkeit mit Identität, Komposition und Homomorphismen.

Parameterspezifikation

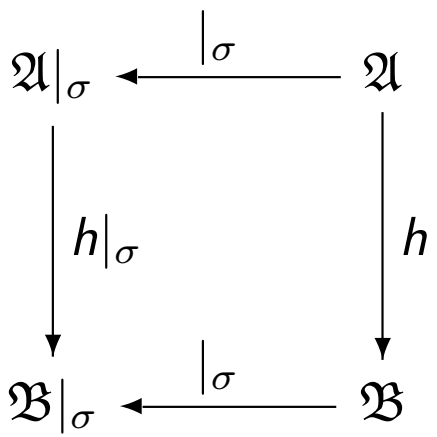


Parameterspezifikation

$\sigma : \text{sig}' \rightarrow \text{sig}$, $\mathfrak{A}, \mathfrak{B}$, sig-Algebren.
$h : \mathfrak{A} \rightarrow \mathfrak{B}$, sig-Homomorphismus.
$h|_\sigma = \{h_{\sigma(s)} \mid s \in S'\}$, $\text{sig}' = (S', F', \tau') =$ mit
$h|_\sigma : A|_\sigma \rightarrow B|_\sigma$ Vergißbild von h entlang σ.



Parameterspezifikation



Semantik der Parameterübergabe (nur Signatur)

Definition 7.17. *Sei Body[Formal] Parameterspezifikation.*
σ : Formal $\longrightarrow$ Actual *Signaturmorphismus.*

Semantik der Parameterübergabe [Actual, σ].

Zuordnung: σ : Formal $\longrightarrow$ Actual

$\downarrow$

initiale Semantik von Value. D. h.

$T_{\text{Body[Actual},\sigma]}$

Betrachte: $S :: (T_{\text{Actual}}, \sigma) \mapsto T_{\text{Body[Actual},\sigma]}$

Abbildung zwischen init Algebren. Kann aufgefasst werden als Zuordnung zwischen Formal Algebren $\longrightarrow$ Body-Algebren.

Semantik Paraübergabe $(T_{\text{Actual}})|_\sigma \mapsto (T_{\text{Body}[\text{Actual},\sigma]})|_{\sigma'}$

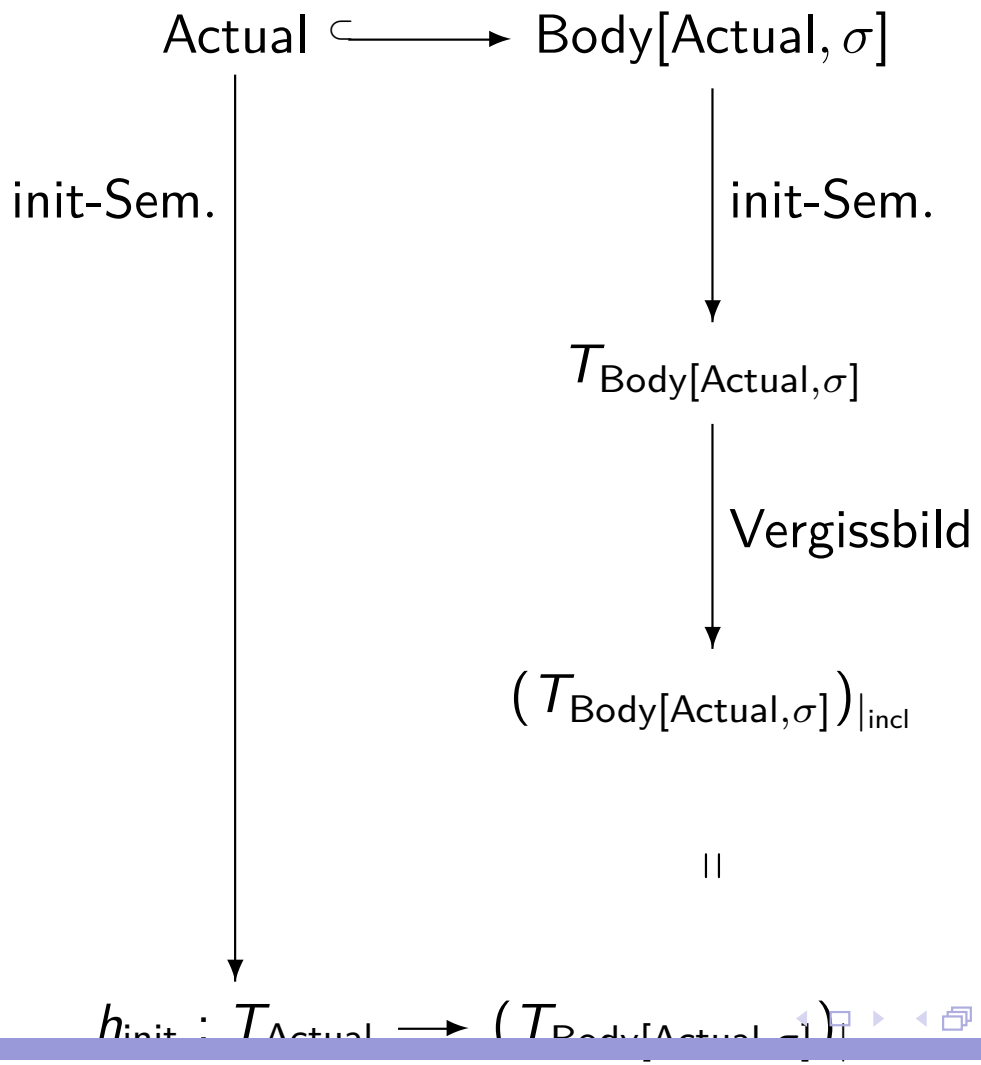


Abbildung zwischen init Algebren

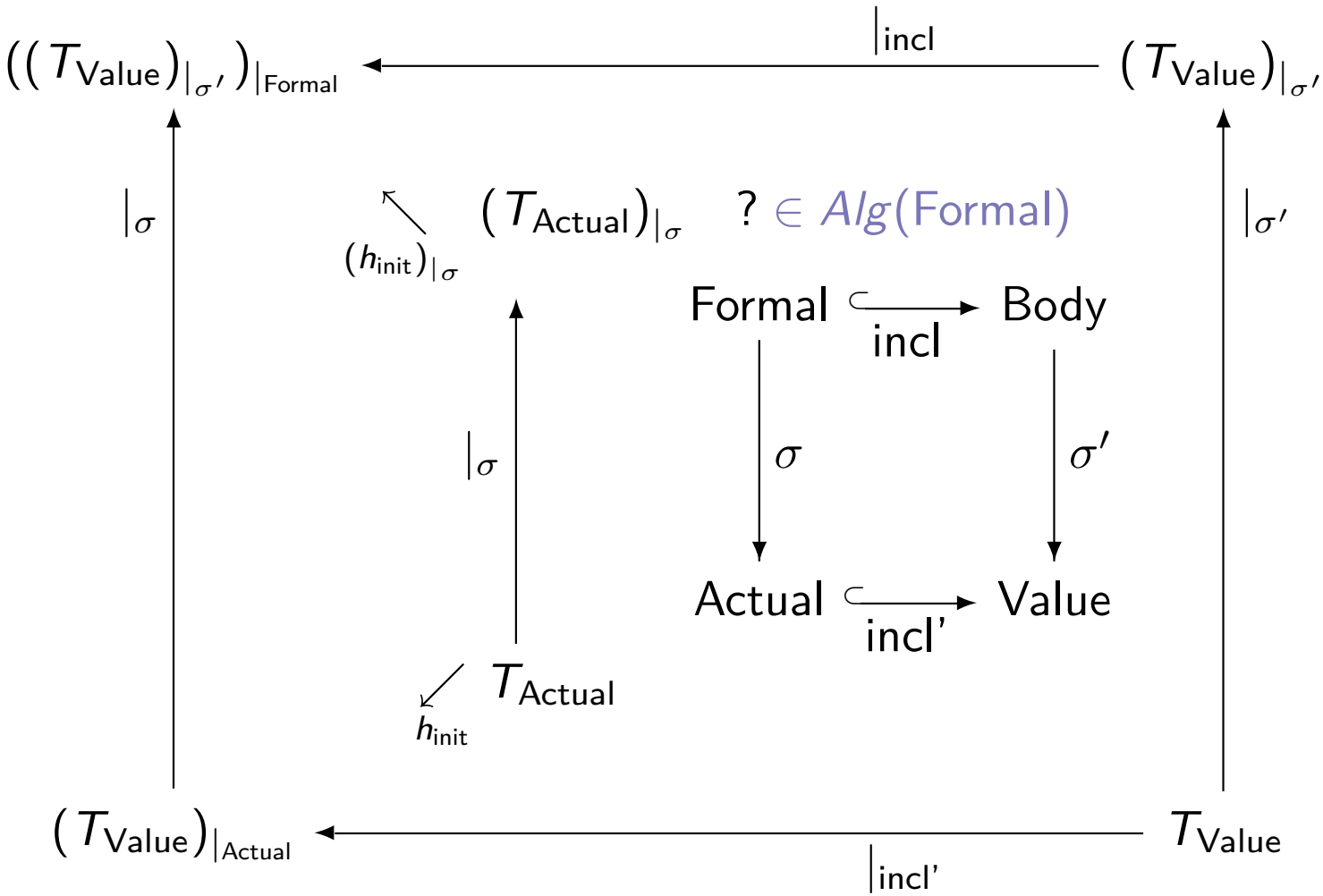


Abbildung zwischen init Algebren

Formal		elem $\longrightarrow$ nat	Actual	
sorts	elem	$\xrightarrow{\sigma}$	sorts	nat
ops	$a, b :\longrightarrow$ elem	$a \longrightarrow 0$	ops	$0, 1 :\longrightarrow$ nat
eqns	$a = b$	$b \longrightarrow 1$		

$$\mathfrak{A} = T_{\text{Actual}} \quad A_{\text{nat}} = \{0, 1\}$$

$\mathfrak{A}|_\sigma \in \text{Alg}(\text{sig Formal})$ $(A|_\sigma)_{\text{elem}} = \{0, 1\}$

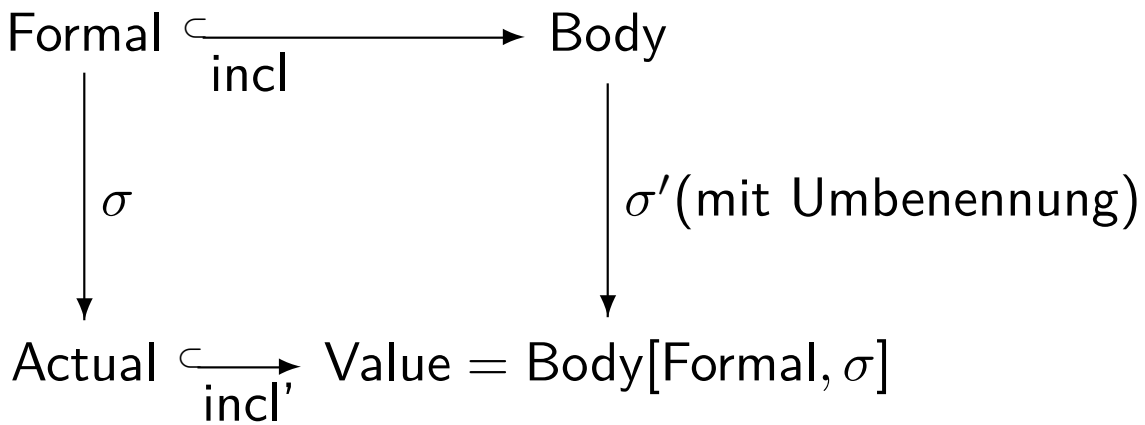
$a|_{\mathfrak{A}|_\sigma} = 0 \neq 1 = b|_{\mathfrak{A}|_\sigma}$

Gleichung von Formal nicht erfüllt! D. h. $\mathfrak{A}|_\sigma \notin \text{Alg}(\text{Formal})$.

Parameterübergabe (Actual, σ)

Body[Formal]

$$\sigma : \text{sig(Formal)} \rightarrow \text{sig(Actual)}$$
Signatur Morphismus



Vor: sig(Actual) und sig(Value) strikt.

Parameterübergabe (Actual, σ)

Vergißbilder: $|_\sigma : \mathsf{Alg}(\mathsf{sig}) \to \mathsf{Alg}(\mathsf{sig}')$

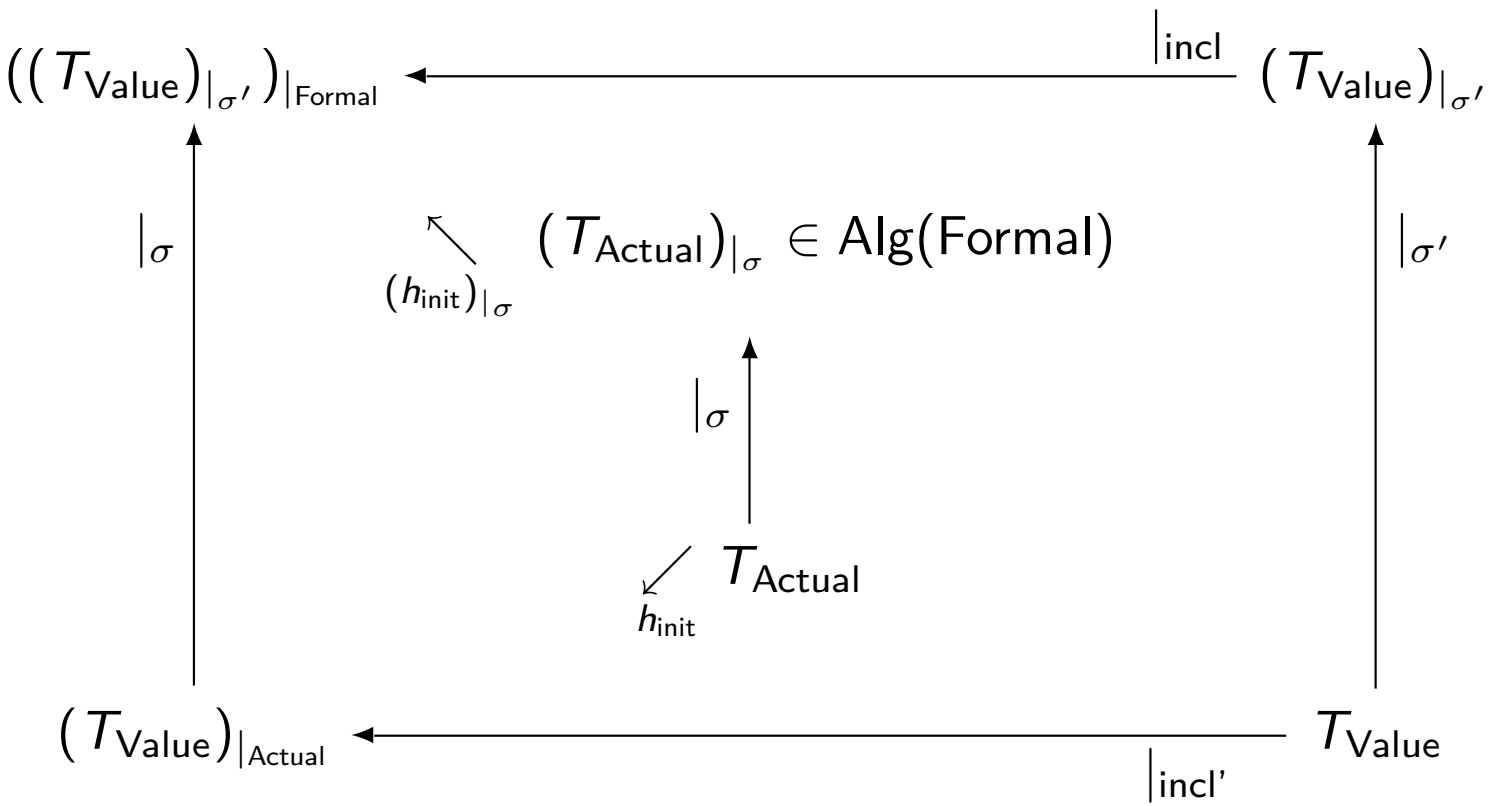
$$\mathfrak{A}|_\sigma \text{ für } \sigma : \mathsf{sig}' \to \mathsf{sig}$$

$h : \mathfrak{A} \to \mathfrak{B}$ sig-Homomorphismus

$$h|_\sigma : \mathfrak{A}|_\sigma \to \mathfrak{B}|_\sigma$$

sig$'$-Homomorphismus

Parameterübergabe (Actual, σ)



Probleme: 1) $(T_{\text{Actual}})|_{\sigma} \notin \text{Alg(Formal)}$, 2) h_{init} keine Bijektion.

Spezifikationsmorphismus

Definition 7.18. *Seien* $spec' = (sig', E'), spec = (sig, E)$ *(allg.) Spezifikationen.*
Ein Signaturmorphismus $\sigma : sig' \to sig$ *heißt Spezifikationsmorphismus, falls für alle* $s = t \in E'$ *gilt* $\sigma(s) = \sigma(t) \in Th(E)$.

Schreibe: $\sigma : spec' \to spec$

Fakt: Für $\mathfrak{A} \in \mathrm{Alg}(spec)$ *gilt* $\mathfrak{A}|_\sigma \in \mathrm{Alg}(spec')$
D. h. $|_\sigma : \mathrm{Alg}(spec) \to \mathrm{Alg}(spec')!$

Oft verlangt man „nur“ $\sigma(s) = \sigma(t) \in ITh(E).!$

Semantisch korrekte Parameterübergabe

Eine Parameterübergabe für Body[Formal] ist ein Paar (Actual, σ): Actual Spezifikation und σ : Formal $\longrightarrow$ Actual Spezifikationsmorphismus.

$(T_{\text{Actual}})|_\sigma \in \text{Alg}(\text{Formal})$

- verlange auch h_{init} Bijektion.
Syntaktische Einschränkungen die dies garantieren.

Spezifikationssprachen

CLEAR, Act-one, -Cip-C, Affirm, ASL, Aspik, OBJ, ASF, $\underset{+}{\rightsquigarrow}$ neuere Sprachen: - Spectrum, - Troll, -Maude

Beispiel

Beispiel 7.19.

$$\text{Formal} :: \begin{cases} \textit{spec} & \text{ELEMENT} \\ \text{use} & \text{BOOL} \\ \text{sorts} & \text{elem} \\ \text{ops} & . \leq . : \text{elem}, \text{elem} \rightarrow \text{bool} \\ \text{eqns} & x \leq x = \textit{true} \\ & \text{imp}(x \leq y \ \textit{and}\ y \leq z, x \leq z) = \textit{true} \\ & x \leq y \ \textit{or}\ y \leq x = \textit{true} \end{cases}$$

Beispiel (Forts.)

spec	LIST[ELEMENT]
use	ELEMENT
sorts	list
ops	$\mathsf{nil} :\longrightarrow \mathsf{list}$
	$. : \mathsf{elem}, \mathsf{list} \longrightarrow \mathsf{list}$
	$\mathsf{insert} : \mathsf{elem}, \mathsf{list} \longrightarrow \mathsf{list}$
	$\mathsf{case} : \mathsf{bool}, \mathsf{list}, \mathsf{list} \longrightarrow \mathsf{list}$
	$\mathsf{sorted} : \mathsf{list} \longrightarrow \mathsf{bool}$

Beispiel (Forts.)

eqns
$$\begin{aligned}
&\mathsf{case}(\mathsf{true}, l_1, l_2) = l_1\\
&\mathsf{case}(\mathsf{false}, l_1, l_2) = l_2\\
&\\
&\mathsf{insert}(x, \mathsf{nil}) = x.\mathsf{nil}\\
&\mathsf{insert}(x, y.l) = \mathsf{case}(x \leq y, \mathsf{insert}(x, \mathsf{insert}(y, l)), \mathsf{insert}(y, \mathsf{insert}(x, l)))\\
&\\
&\mathsf{sorted}(\mathsf{nil}) = \mathsf{true}\\
&\mathsf{sorted}(x.\mathsf{nil}) = \mathsf{true}\\
&\mathsf{sorted}(x.y.l) = \mathsf{if}\ x \leq y\ \mathsf{then}\ \mathsf{sorted}(y, l)\ \mathsf{else}\ \mathsf{false}
\end{aligned}$$

Eigenschaft: $\mathsf{sorted}(\mathsf{insert}(x, l)) = \mathsf{true}$

Beispiel (Forts)

ACTUAL $\equiv$ BOOL

σ : elem $\longrightarrow$ bool, bool $\longrightarrow$ bool
. $\leq$. $\longrightarrow$ impl

Die Gleichungen von ELEMENT sind in Th(BOOL)

$\rightsquigarrow$ Spezifikationsmorphismus

Beispiel (Forts.)

ACTUAL $\equiv$ NAT

σ :

bool $\longrightarrow$ nat	elem $\longrightarrow$ nat
true $\longrightarrow$ suc(0)	nicht erlaubt
false $\longrightarrow$ 0	
not $\longrightarrow$ suc	
or $\longrightarrow$ plus	
and $\longrightarrow$ times	
$\vdots$	
$. \leq . \longrightarrow \cdots$	

kein Spezifikationsmorphismus
not(false) = true
not(true) = false gilt nicht!.

Abstrakte Reduktionssysteme: Begriffswelt

Definition 8.1. $(U, \rightarrow) U \neq \varnothing, \rightarrow$ *Binärrelation heißt Reduktionssystem.*

- *Begriffe:*
- $x \in U$ *reduzibel* $\mathrm{gdw}\ \exists y : x \rightarrow y$
  irreduzibel
- $x \xrightarrow{*} y$ *reflexiv, transitive Hülle,* $x \xrightarrow{+} y$ *transitive Hülle,* $x \xleftrightarrow{*} y$ *reflexive, symmetrische, transitive Hülle. (Analog* $x \xrightarrow{i} y$*...)*
- $x \xrightarrow{*} y$, *y irreduzibel, so y Normalform für x.*
- $\Delta(x) = \{y \mid x \rightarrow y\}$ *unmittelbare Nachfolger.*
- $\Delta^+(x)$ *echte Nachfolger,* $\Delta^*(x)$ *Nachfolger.*

Begriffswelt

- $\Lambda(x) = \max\{i \mid \exists y : x \xrightarrow{i} y\}$ Ableitungskomplexität. $\Lambda : U \to \mathbb{N}_\infty$
- $\to$ noethersch (terminierend, erfüllt Kettenbedingung), falls es keine unendliche Kette $x_1 \to x_2 \to x_3 \to \cdots$ gibt.
- $\to$ beschränkt, falls $\Lambda : U \to \mathbb{N}$.
- $\to$ kreisfrei, $\neg\exists x \in U : x \xrightarrow{+} x$
- $\to$ lokal endlich $x \left\{ \begin{array}{c} \nearrow \\ \to \\ \searrow \end{array} \right\}$,d.h $\Delta(x)$ endlich für alle x.

Begriffswelt

Einfache Zusammenhänge:

- $\rightarrow$ kreisfrei, so $\overset{*}{\longrightarrow}$ Partialordnung.
- $\rightarrow$ noethersch, so $\rightarrow$ kreisfrei.
- $\rightarrow$ beschränkt, so $\rightarrow$ noethersch.
  aber nicht umgekehrt!
- $\rightarrow \subset \overset{+}{\Rightarrow}, \Rightarrow$ noethersch, so $\rightarrow$ noethersch.

Prinzip der noetherschen Induktion

Definition 8.2. $\longrightarrow$ *Relation auf* U, P *Prädikat auf* U.
P *ist* $\longrightarrow$*-vollständig, falls*

$$\forall x[(\forall y \in \Delta^+(x) : P(y)) \supset P(x)]$$

PNI: Prinzip der noetherschen Induktion:
Ist $\longrightarrow$ *noethersch und ist* P $\longrightarrow$*-vollständig, so gilt* $P(x)$ *für alle* $x \in U$.

Anwendungen

Lemma 8.3. *$\to$ noethersch, so hat jedes $x \in U$ mindestens eine NF.*
(weitere Anwendungen Lemma von König)

Definition 8.4. *Zentrale Eigenschaften:* $(U, \to)$

- $\to$ *konfluent* gdw $\overset{*}{\leftarrow} \circ \overset{*}{\rightarrow} \subset \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$
- $\to$ *Church-Rosser* gdw $\overset{*}{\leftrightarrow} \subset \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$
- $\to$ *lokal-konfluent* gdw $\leftarrow \circ \rightarrow \subset \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$
- $\to$ *streng-konfluent* gdw $\leftarrow \circ \rightarrow \subset \overset{*}{\rightarrow} \circ \overset{\leq 1}{\leftarrow}$
- *Abkürzung: zusammenführbar* $\downarrow$*:*

$$\downarrow = \overset{*}{\rightarrow} \circ \overset{*}{\leftarrow}$$

Wichtige Zusammenhänge

Lemma 8.5. $\to$ *konfluent* gdw $\to$ *Church-Rosser.*

Satz 8.6. *(Newmann Lemma) Sei* $\to$ *noethersch, dann* $\to$ *konfluent* gdw $\to$ *lokal konfluent.*

Folgerung 8.7.

a) $\to$ *konfluent und* $x \overset{*}{\longleftrightarrow} y$.

   i) *Ist* y *irreduzibel, so* $x \overset{*}{\longrightarrow} y$.
   Insbesondere, wenn x, y *irreduzibel, so* $x = y$.

   ii) $x \overset{*}{\longleftrightarrow} y$ gdw $\Delta^*(x) \cap \Delta^*(y) \neq \varnothing$.

   iii) *Hat* x *eine NF, so ist sie eindeutig.*

   iv) *Ist* $\to$ *noethersch, so hat jedes* $x \in U$ *genau eine NF:*
   Schreibweise $x\downarrow$

b) $(U, \to)$ *hat jedes* $x \in U$ *genau eine NF, so ist* $\to$ *konfluent, i. Allg. nicht noethersch.*

Konvergente Reduktionssysteme

Definition 8.8. $(U, \to)$ *konvergent* $\mathrm{gdw} \to$ *noethersch und konfluent.*

Wichtig da: $x \overset{*}{\longleftrightarrow} y \ \mathrm{gdw}\ x\downarrow = y\downarrow$

$\to$ *effektiv so Entscheidungsverfahren für WP:*

Für Programmierung: $x \overset{*}{\longrightarrow} x\downarrow$, $f(t_1, \ldots, t_n) \overset{*}{\longrightarrow}$ *„Wert“*

i. Allg. Unentscheidbare Eigenschaften

Terminierung und Konfluenz

Hinreichende Bedingungen/Techniken

Lemma 8.9. *$(U, \to)$, $(M, \succ)$, $\succ$ WF Partialordnung. Gibt es $\varphi : U \to M$ mit $\varphi(x) \succ \varphi(y)$, falls $x \to y$, so ist $\to$ noethersch.*

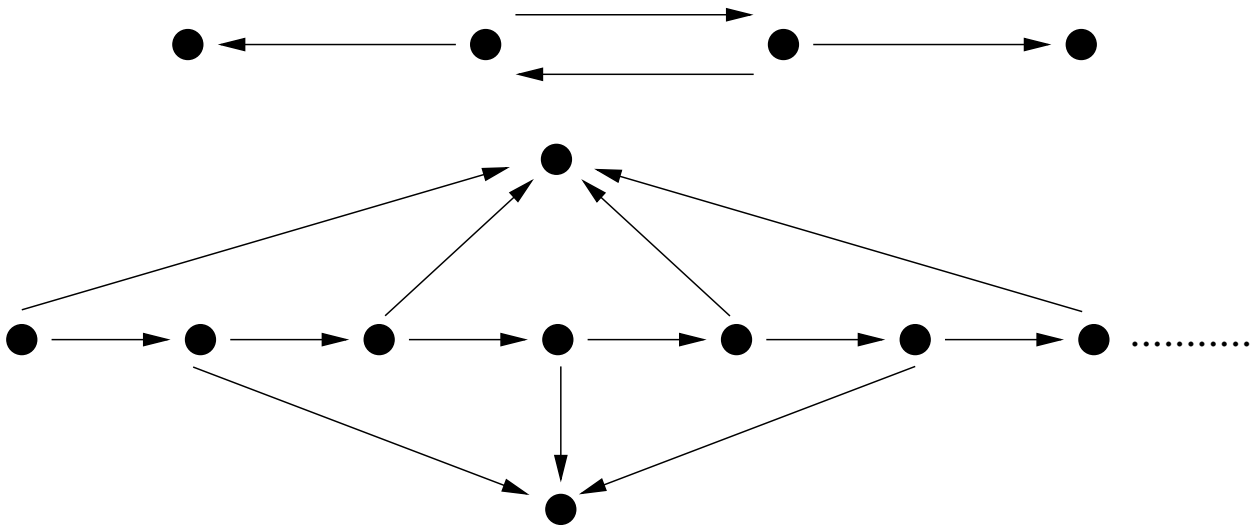
Beispiel 8.10. *Oft* $(\mathbb{N}, >), (\Sigma^*, >)$
$w \in \Sigma^*$, $|w|$ Länge, $|w|_a$ a-Länge $a \in \Sigma$.
WF-Partialordnungen auf Σ^*

- $x > y$ gdw $|x| > |y|$
- $x > y$ gdw $|x|_a > |y|_a$
- $x > y$ gdw $|x| > |y|$, $|x| = |y| \wedge x \succ_{lex} y$

Beachte reine Lex-Ordnung nicht noethersch.

Hinreichende Bedingung für Konfluenz

Terminierung: Konfluenz gdw lokale Konfluenz
Ohne Terminierung gilt dies nicht!

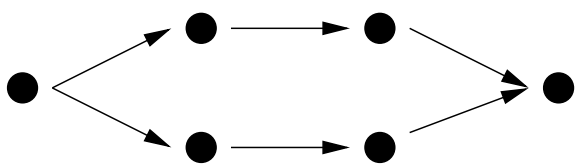


Abschwächung der Terminierung

Satz 8.11. *$\to$ ist konfluent* gdw *für alle $u \in U$ gilt: aus $u \to x$ und $u \overset{*}{\to} y$ folgt $x \downarrow y$.*

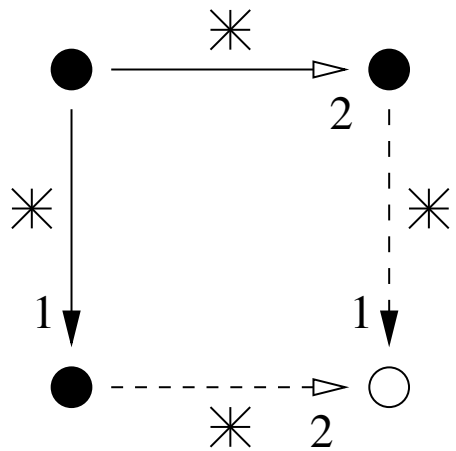
▷ *einseitige Lokalisierung der Konfluenz* ◁

Satz 8.12. *Ist $\to$ streng konfluent, so ist $\to$ konfluent.*

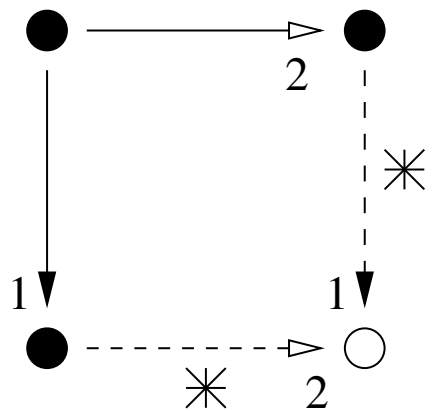


Zusammenhang von Relationen

Definition 8.13. *Zwei Relationen* $\to_1, \to_2$ *auf* U *kommutieren, falls gilt* ${}_1\overset{*}{\leftarrow} \circ \overset{*}{\to}_2 \subseteq \overset{*}{\to}_2 \circ {}_1\overset{*}{\leftarrow}$.
Sie kommutieren lokal falls gilt ${}_1\leftarrow \circ \to_2 \subseteq \overset{*}{\to}_2 \circ {}_1\overset{*}{\leftarrow}$.



kommutierend



lokal kommutierend

Zusammenhang von Relationen

Lemma 8.14. *Sei* $\rightarrow \quad = \quad \rightarrow_1 \cup \rightarrow_2$
(1) Kommutieren $\rightarrow_1$ *und* $\rightarrow_2$ *lokal, und ist* $\rightarrow$ *Noethersch, so kommutieren* $\rightarrow_1$ *und* $\rightarrow_2$*.*
(2) Sind $\rightarrow_1$ *und* $\rightarrow_2$ *konfluent und kommutieren, so ist auch* $\rightarrow$ *konfluent.*

Problem: Nicht -Orientierbarkeit:
(a) $x + 0 = x, \;\; x + s(y) = s(x + y)$
(b) $x + y = y + x, \;\; (x + y) + z = x + (y + z)$

▷ *Problem: permutative Regeln (b)* ◁

Nicht-Orientierbarkeit

Definition 8.15. *$(U, \to, \vdash\!\dashv)$ mit $\to$ Relation, $\vdash\!\dashv$ symmetrische Relation.*

Sei $\models\!\models = \leftrightarrow \cup \vdash\!\dashv$, $\sim = \overset{*}{\vdash\!\dashv}$, $\approx = \overset{*}{\models\!\models}$,

$\to_\sim = \sim \circ \to \circ \sim$, $\downarrow_\sim = \overset{*}{\to} \circ \sim \circ \overset{*}{\leftarrow}$.

Gilt $x \downarrow_\sim y$, dann heißen $x, y \in U$ zusammenführbar modulo $\sim$.

$\to$ *heißt Church-Rosser modulo* $\sim$: $\mathrm{gdw} \approx \subseteq \downarrow_\sim$

$\to$ *heißt lokal konfluent modulo* $\sim$: $\mathrm{gdw} \leftarrow \circ \to \subseteq \downarrow_\sim$

$\to$ *heißt lokal kohärent modulo* $\sim$: $\mathrm{gdw} \leftarrow \circ \vdash\!\dashv \subseteq \downarrow_\sim$

Nicht-Orientierbarkeit- Reduktion Modulo $\vdash$

Satz 8.16. *Sei $\to_\sim$ terminierend. Dann ist $\to$ genau dann Church-Rosser modulo $\sim$, wenn $\sim$ lokal konfluent modulo $\sim$ und lokal kohärent modulo $\sim$ ist.*

Häufigste Anwendung: Modulo AC (Assoziativität + Kommutativität)

Darstellung von Äquivalenzrelationen durch konvergente Reduktionsrelationen

Situation: gegeben: $(U, \vdash\!\!\!\!-)$, gesucht: $(U, \rightarrow)$ mit
(i) $\rightarrow$ konvergent bzgl. Noetherscher PO $>$ auf U und
(ii) $\overset{*}{\leftrightarrow} \;=\; \sim$ mit $\sim \;=\; \overset{*}{\vdash\!\!\!\!-}$

Idee: Approximation von $\rightarrow$ durch Transformationen
$(\vdash\!\!\!\!-, \emptyset) = (\vdash\!\!\!\!-_0, \rightarrow_0) \vdash (\vdash\!\!\!\!-_1, \rightarrow_1) \vdash (\vdash\!\!\!\!-_2, \rightarrow_2) \vdash \ldots$

Invariante im i-ten Schritt:
(i) $\sim \;=\; (\vdash\!\!\!\!-_i \cup \leftrightarrow_i)^*$ und
(ii) $\rightarrow_i \;\subseteq\; >$

Ziel: $\vdash\!\!\!\!-_i = \emptyset$ für ein i und $\rightarrow_i$ konvergent.

Darstellung von Äquivalenzrelationen durch konvergente Reduktionsrelationen

Erlaubte Operationen im i-ten Schritt:

(1) $u \to_{i+1} v$, falls $u > v$ und $u \vdash_i v$

(2) $u \vdash_{i+1} v$, falls $u \;{}_i\!\leftarrow w \to_i v$

(3) verkleinere $u \vdash_i v$ zu $u \vdash_{i+1} w$, falls $v \to_i w$

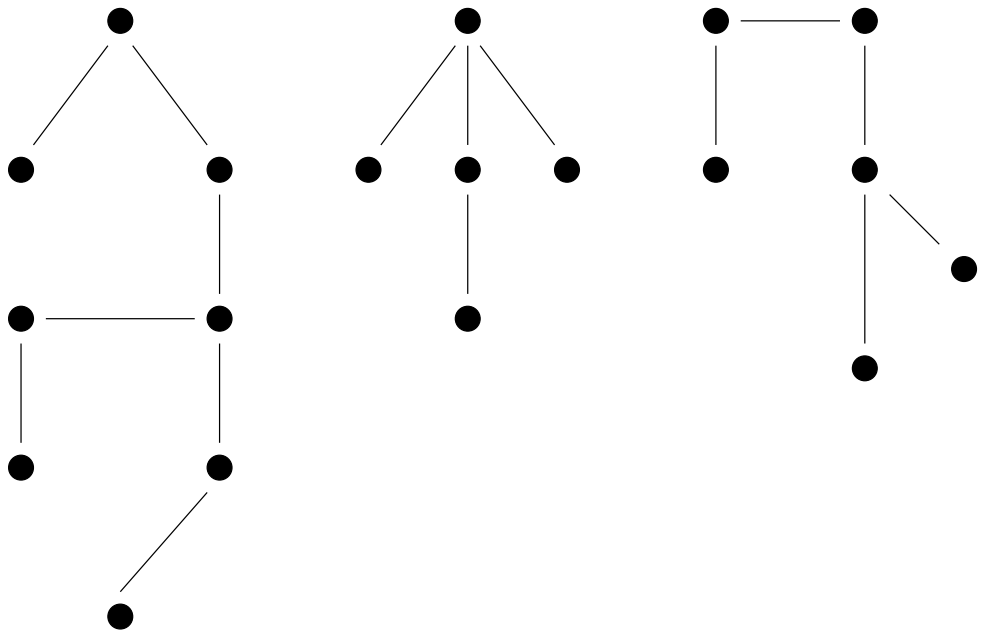
Ziel: Grenzsystem

$\to \;=\; \to_\infty \;=\; \bigcup\{\to_i \mid i \in \mathbb{N}\}$ mit $\vdash_\infty \;=\; \emptyset$

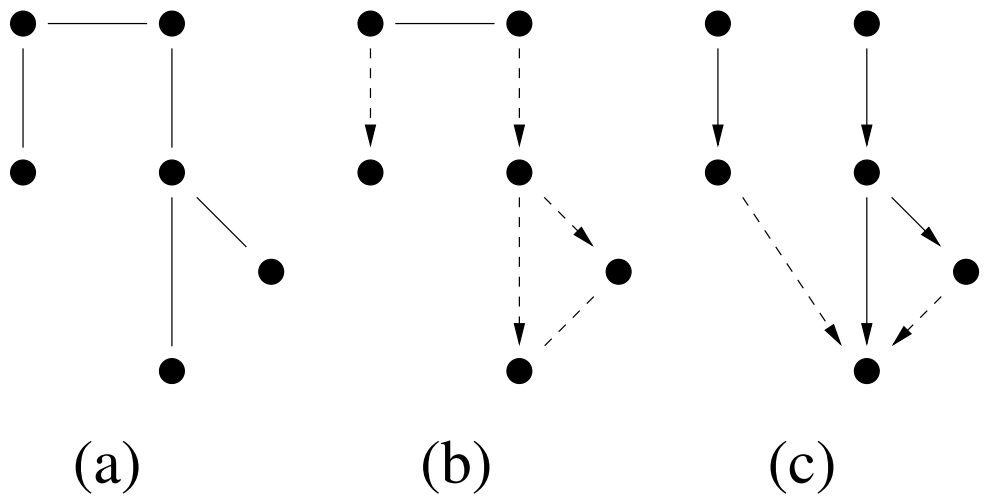
Also:

- $\longrightarrow_\infty \subseteq >$, d. h. noethersch
- $\overset{*}{\longleftrightarrow} = \sim$
- $\longrightarrow_\infty$ konvergent !

Graphische Darstellung einer Äquivalenzrelation



Transformation einer Äquivalenzrelation



Inferenzsystem zur Transformation einer Äquivalenzrelation

Definition 8.17. *Sei $>$ eine Noethersche PO auf U. Das Inferenzsystem $\mathcal{P}$ besteht aus folgenden Regeln:*

(1) *Orientieren*

$$\frac{(\vdash \cup \{u \vdash v\}, \rightarrow)}{(\vdash, \rightarrow \cup \{u \rightarrow v\})} \text{ falls } u > v$$

(2) *Neue Konsequenz einführen*

$$\frac{(\vdash, \rightarrow)}{(\vdash \cup \{u \vdash v\}, \rightarrow)} \text{ falls } u \leftarrow \circ \rightarrow v$$

(3) *Simplifizieren*

$$\frac{(\vdash \cup \{u \vdash v\}, \rightarrow)}{(\vdash \cup \{u \vdash w\}, \rightarrow)} \text{ falls } v \rightarrow w$$

Inferenzsystem (Fort.)

(4) Identitäten entfernen

$$\frac{(\vdash \cup \{u \vdash u\}, \rightarrow)}{(\vdash, \rightarrow)}$$

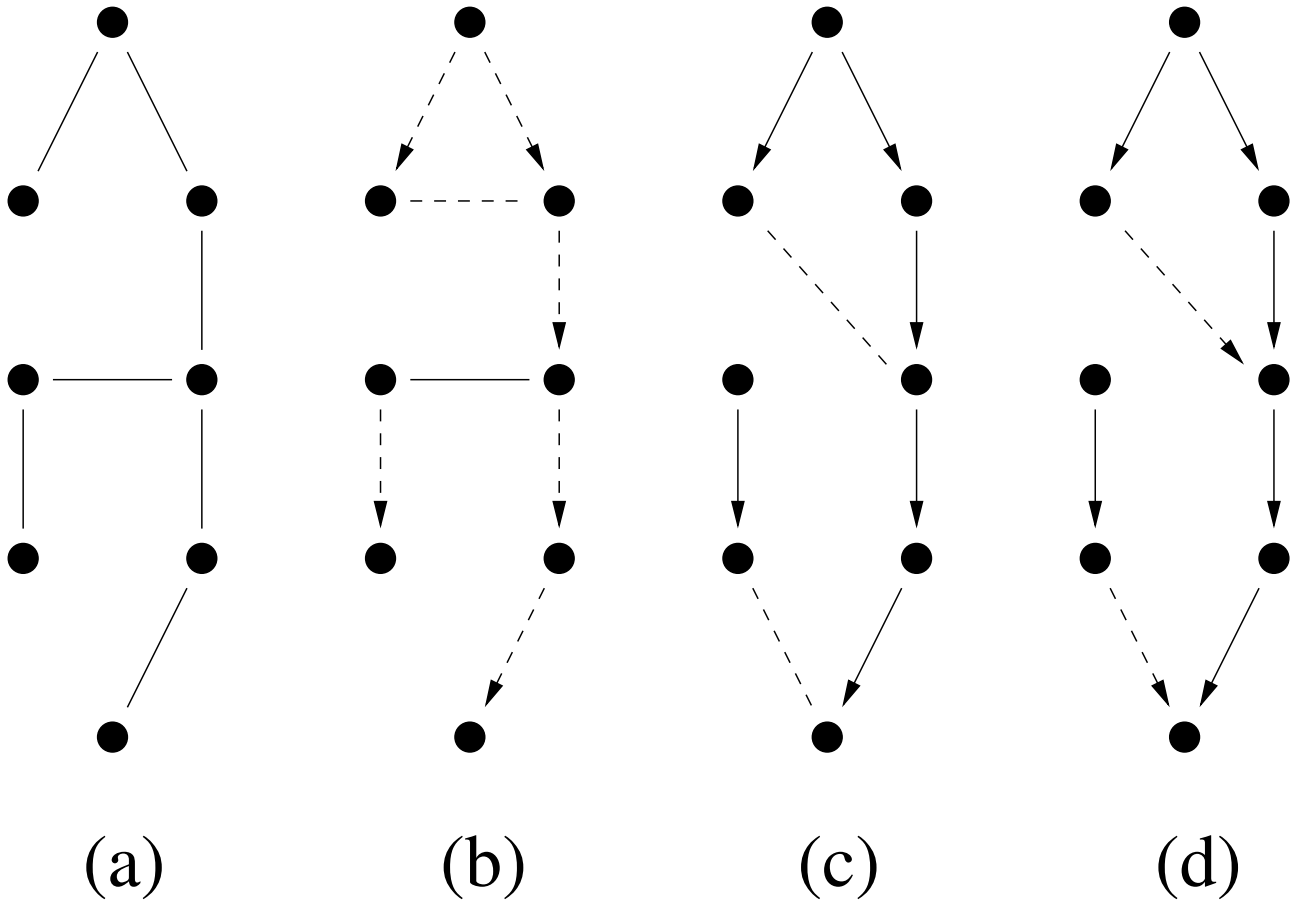
$(\vdash, \rightarrow) \vdash_{\mathcal{P}} (\vdash', \rightarrow')$ falls $(\vdash, \rightarrow)$ mit $\mathcal{P}$ in einem Schritt in $(\vdash', \rightarrow')$ überführt werden kann.

$\vdash_{\mathcal{P}}^{*}$ Überführungsrelation in endlich vielen Schritten.

Eine Folge $((\vdash_i, \rightarrow_i))_{i \in \mathbb{N}}$ heißt $\mathcal{P}$-*Ableitung*, falls

$(\vdash_i, \rightarrow_i) \vdash_{\mathcal{P}} (\vdash_{i+1}, \rightarrow_{i+1})$ für alle $i \in \mathbb{N}$.

Transformation mit dem Inferenzsystem



Eigenschaften des Inferenzsystems

Lemma 8.18. *Sei* $(\vdash, \rightarrow) \vdash_{\mathcal{P}} (\vdash', \rightarrow')$

(a) *Ist* $\rightarrow \subseteq >$, *so gilt auch* $\rightarrow' \subseteq >$

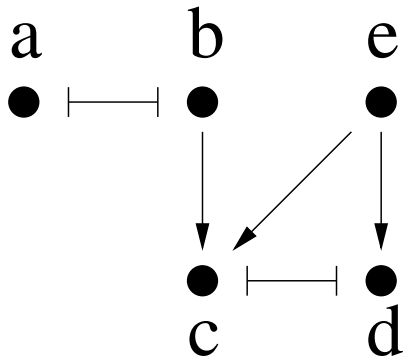
(b) *Es gilt* $(\vdash \cup \leftrightarrow)^* = (\vdash' \cup \leftrightarrow')^*$

Problem: Wann liefert $\mathcal{P}$ konvergente Reduktionsrelation $\rightarrow$?

Idee: Definiere Ordnung $>_{\mathcal{P}}$ auf Äquivalenz-Beweisen, und zeige, daß das Inferenzsystem $\mathcal{P}$ Beweise bzgl. $>_{\mathcal{P}}$ verkleinert! Dabei sollten $\overset{*}{\longrightarrow} \circ \overset{*}{\longleftarrow}$ Beweise in der Ordnung minimal sein.

Eigenschaften des Inferenzsystems

Definition 8.19. *Sei $(\vdash, \to)$ gegeben und $>$ eine Noethersche PO auf U. Sei weiter $(\vdash \cup \leftrightarrow)^* = \sim$.*
Ein Beweis für $u \sim v$ ist eine Folge $u_0 *_1 u_1 *_2 \cdots *_n u_n$ mit $*_i \in \{\vdash, \leftarrow, \to\}$, $u_i \in U$, $u_0 = u$, $u_n = v$ und für alle i gilt $u_i *_{i+1} u_{i+1}$.
$P(u) = u$ ist Beweis für $u \sim u$.
Ein Beweis der Form $u \xrightarrow{*} z \xleftarrow{*} v$ heißt V-Beweis.



Beweise für $a \sim e$:

$$P_1(a, e) = a \vdash b \to c \vdash d \leftarrow e$$
$$P_2(a, e) = a \vdash b \to c \leftarrow e$$

Beweisordnungen

Beachte: Sind $P_1(u, v), P_2(v, w)$ und $P_3(w, z)$ Beweise, so ist auch $P(u, z) = P_1(u, v)P_2(v, w)P_3(w, z)$ ein Beweis.

Definition 8.20. *Eine Beweisordnung $>_B$ ist eine PO auf der Menge der Beweise, die monoton ist, d.h. $P >_B Q$ für jeden Teilbeweis, und aus $P >_B Q$ folgt $P_1PP_2 >_B P_1QP_2$.*

Lemma 8.21. *Sei $>$ Noethersche PO auf U und $(\vdash, \rightarrow)$, dann existieren Noethersche Beweisordnungen.*

Beweis: Über Multimengenordnungen.

Multimengenordnung

Instrumentarium: Multimengenordnungen

Objekte: U, $Mult(U)$ Multimengen über U

$A \in Mult(U)$:gdw $A : U \to \mathbb{N}$ mit $\{u \mid A(u) > 0\}$ endlich

Operationen: $\cup, \cap, -$

$(A \cup B)(u) := A(u) + B(u)$
$(A \cap B)(u) := min\{A(u), B(u)\}$
$(A - B)(u) := max\{0, A(u) - B(u)\}$

Multimengenordnung

Definition 8.22. *Erweiterung von $(U, >)$ zu $(Mult(U), \gg)$*
$A \gg B$:gdw es gibt $X, Y \in Mult(U)$ mit $\emptyset \neq X \subseteq A$ und $B = (A - X) \cup Y$, so daß $\forall y \in Y \;\; \exists x \in X \; x > y$

Eigenschaften:

(1) $>$ PO $\rightsquigarrow \gg$ PO
(2) $\{m_1\} \gg \{m_2\}$ gdw $m_1 > m_2$
(3) $>$ total $\rightsquigarrow \gg$ total
(4) $A \gg B \rightsquigarrow A \cup C \gg B \cup C$
(5) $>$ Noethersch gdw $\gg$ Noethersch

Konstruktion der Beweisordnung

Ordne jedem „atomaren“ Beweis eine Komplexität zu

$$c(u * v) = \begin{cases} \{u\} & \text{falls } u \longrightarrow v \\ \{v\} & \text{falls } u \longleftarrow v \\ \{u, v\} & \text{falls } u \vdash\!\dashv v \end{cases}$$

Erweitere diese Komplexität auf „zusammengesetzte“ Beweise durch

$c(P(u)) = \emptyset$
$c(P(u, v)) = \{c(u_i *_{i+1} u_{i+1}) \mid i = 0, \ldots n-1\}$

beachte: $c(P(u, v)) \in \mathit{Mult}(\mathit{Mult}(U))$

Definiere Ordnung auf Beweisen durch
$P >_{\mathcal{P}} Q : \text{gdw } c(P) \ggg c(Q)$

Konstruktion der Beweisordnung

Es gilt: $>_{\mathcal{P}}$ ist Noethersche Beweisordnung!

Welche Beweisschritte sind groß bzw. klein

Betrachte dazu:
(a) $P_1 = x \leftarrow u \rightarrow y$, $P_2 = x \mapsto y$
$c(P_1) = \{\{u\}, \{u\}\} \ggg \{\{x, y\}\} = c(P_2)$
$\rightsquigarrow P_1 >_{\mathcal{P}} P_2$
analog

(b) $P_1 = x \mapsto y$, $P_2 = x \rightarrow y$

(c) $P_1 = u \mapsto v$, $P_2 = u \mapsto w \leftarrow v$

(d) $P_1 = u \mapsto v$, $P_2 = u \rightarrow w \leftarrow v$

Faire Ableitungen in $\mathcal{P}$

Definition 8.23. *Sei $(\vdash_i, \to_i)_{i\in\mathbb{N}}$ eine $\mathcal{P}$-Ableitung. Setze*

$$\vdash^\infty = \bigcup_{i\geq 0} \bigcap_{j\geq i} \vdash_i \text{ und } \to^\infty = \bigcup_{i\geq 0} \to_i.$$

Die $\mathcal{P}$-Ableitung heißt fair, falls gilt

(1) $\vdash^\infty = \emptyset$ und
(2) Ist $x \overset{\infty}{\leftarrow} u \to^\infty y$, dann existiert $k \in \mathbb{N}$ mit $x \vdash_k y$.

Lemma 8.24. *Sei $(\vdash_i, \to_i)_{i\in\mathbb{N}}$ eine faire $\mathcal{P}$-Ableitung*

(a) Zu jedem Beweis P in $(\vdash_i, \to_i)$ gibt es einen äquivalenten Beweis P' in $(\vdash_{i+1}, \to_{i+1})$ mit $P \geq_{\mathcal{P}} P'$.

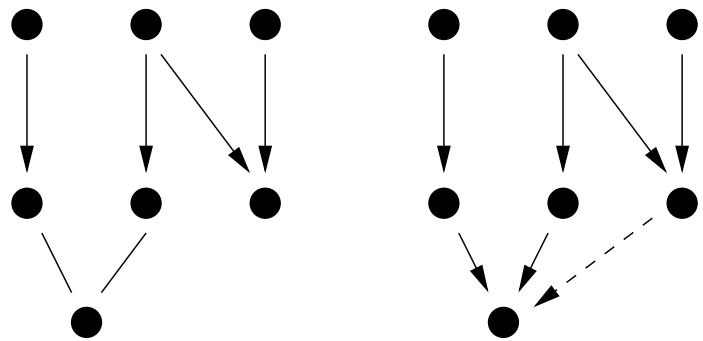
(b) Sei $i \in \mathbb{N}$ und P Beweis in $(\vdash_i, \to_i)$, der kein V-Beweis ist. Dann existiert ein $j > i$ und einen zu P äquivalenten Beweis P' in $(\vdash_j, \to_j)$ mit $P >_{\mathcal{P}} P'$.

Hauptergebniss

Satz 8.25. *Sei* $(\vdash_i, \to_i)_{i\in\mathbb{N}}$ *eine faire* $\mathcal{P}$*-Ableitung und* $\to \quad = \quad \to^\infty$

(a) Gilt $u \sim v$*, so existiert ein* $i \in \mathbb{N}$ *mit* $u \xrightarrow{*}_i \circ {}_i\xleftarrow{*} v$

(b) $\to$ *ist konvergent, und es gilt* $\overset{*}{\leftrightarrow} \quad = \quad \sim$



Termersetzungssysteme

Ziel: Operationalisierung von Spezifikationen und Implementierung funktionaler Programmiersprachen

$spec = (sig, E)$ Wann ist T_{spec} berechenbare Algebra?

$(T_{spec})_s = \{[t] : t \in Term(sig)_s\}$
T_{spec} ist berechenbare Algebra wenn es eine berechenbare Funktion $rep : Term(sig) \to Term(sig)$ gibt, mit $rep(t) \in [t]$ eindeutiger Repräsentant in seiner Äquivalenzklasse.

Paradigma: Wähle als Repräsentanten minimale Objekte in der Äquivalenzklasse bezüglich einer Ordnung.

$f(x_1, ..., x_n) : ((T_{spec})_{s_1} \times ...(T_{spec})_{s_n}) \to (T_{spec})_s$
$f([r_1], ..., [r_n]) := [rep(f(rep(r_1), ..., (rep(r_n))]$

Termersetzungssysteme

Definition 9.1. *Regeln, Regelmengen, Reduktionsrelation*

- *Mengen von Variablen: Für $t \in Term_s(F, V)$ sei $V(t)$ die Menge der Variablen in t (Rek. Definition, immer endlich)*
  Beachte: $V(t) = \emptyset$ gdw t ist Grundterm.
- *Eine Regel ist ein Paar $(l, r), l, r \in Term_s(F, V)(s \in S)$ mit $Var(r) \subseteq Var(l)$*
  Schreibe: $l \to r$
- *Ein Regelsystem R ist eine Menge von Regeln.*
  R definiert eine Reduktionsrelation $\to_R$ auf $Term(F, V)$ durch:

  $$t_1 \to_R t_2 \quad \text{gdw } \exists\, l \to r \in R, p \in O(t_1), \sigma \text{ Substitution } : \\ t_1|_p = \sigma(l) \wedge t_2 = t_1[\sigma(r)]_p$$

- *$(Term(F, V), \to_R)$ sei das von R definierte Reduktionssystem (Termersetzungssystem).*

Termersetzungssysteme

Ziel: Transformiere E in R, so dass $=_E=\overset{*}{\longleftrightarrow}_R$ gilt und $\rightarrow_R$ gutartige Terminierungs- und Konfluenzeigenschaften hat.
Etwa konvergent oder konfluent. Oft genügt es wenn diese Eigenschaften "nur" auf der Menge der Grundterme gelten.

Beachte:

- Die Bedingung $V(r) \subseteq V(l)$ in Regel $l \rightarrow r$ ist notwendig für die Terminierung.
  Gilt weder $V(r) \subseteq V(l)$ noch $V(l) \subseteq V(r)$ in einer Gleichung $l = r$ einer Spezifikation, so hat man wohl überflüssige Variablen bei der Funktionsdefinition verwendet.
- $\rightarrow_R$ ist verträglich mit der Substitution und der Termersetzung. D.h. Aus $s \rightarrow_R t$ folgt, $\sigma(s) \rightarrow_R \sigma(t)$ und $u[s]_p \rightarrow_R u[t]_p$
- Insbesondere $=_R=\overset{*}{\longleftrightarrow}_R$

Matching oder Treffende Substitution

Definition 9.2. *Seien $l, t \in Term_s(F, V)$. Eine Substitution σ heißt Match (treffende Substitution) von l auf t, falls $\sigma(l) = t$*

Folgerung 9.3. *Es gilt:*

- $\forall\ \sigma$ *Substitution* $O(l) \subseteq O(\sigma(l))$.
- $\exists \sigma : \sigma(l) = t$ *gdw es gilt für* σ *definiert durch*
  $\forall u\ O(l) : l|_u = x \in V \rightsquigarrow u \in O(t) \wedge \sigma(x) = t|_u$
  σ *ist Substitution* $\wedge\ \sigma(l) = t$.
- *Gibt es eine solche Substitution, so ist sie eindeutig auf* $V(l)$.
  Existenz und gegebenfalls Berechnung sind effektiv.
- *Es ist entscheidbar ob* t *mit Regel* $l \to r$ *reduzierbar.*
- *Ist* R *endlich, so* $\Delta(s) = \{t : s \to_R t\}$ *endlich un berechenbar.*

Beispiele

Beispiel 9.4. *Ganze Zahlen*

$$\begin{aligned} sig &: 0 :\longrightarrow int \\ & s, p : int \longrightarrow int \\ & if0 : int, int, int \longrightarrow int \\ & F : int, int \longrightarrow int \end{aligned}$$

$$\begin{aligned} eqns &: 1 :: p(0) = 0 \\ & 2 :: p(s(x)) = x \\ & 3 :: if0(0, x, y) = x \\ & 4 :: if0(s(z), x, y) = y \\ & 5 :: F(x, y) = if0(x, 0, F(p(x), F(x, y))) \end{aligned}$$

Interpretation: $\langle \mathbb{N}, ..., \rangle$ *spec- Algebra mit Funktionen*
$O_{\mathbb{N}} = 0, s_{\mathbb{N}} = \lambda n.\ n + 1,$
$p_{\mathbb{N}} = \lambda n.\ if\ \ n = 0\ \ then\ \ 0\ \ else\ \ n - 1\ \ fi$
$if0_{\mathbb{N}} = \lambda i, j, k.\ if\ \ i = 0\ then\ \ j\ else\ \ k\ \ fi$
$F_{\mathbb{N}} = \lambda m, n.\ 0$

Orientiere Gleichungen von links nach rechts $\rightsquigarrow$ *Regeln R (Variablenbedingung ist erfüllt). Ist R terminierend? Nicht mit syntaktischer Ordnung, da linke Seite in rechter Seite vorkommt.*

Beispiel (Fort.)

Reduktionsfolge:

$$F(s(0),0) \to_5 if0(s(0),0,F(\underbrace{p(s(0))}_{2},\underbrace{F(s(0),0)}_{5}))$$

$$\to_4 F(\underbrace{p(s(0))}_{},\underbrace{F(s(0),0)}_{})$$

$$\to_2 F(0,\underbrace{F(s(0),0)}_{})$$

$$\to_5 if0(0,0,F(\underbrace{p(0)}_{},F(0,\underbrace{F(s(0),0)}_{}))) \to_3 0$$

Äquivalenz

Definition 9.5. *Seien* $spec = (sig, E), spec' = (sig, E')$ *Spezifikationen. Sie sind äquivalent falls* $=_E = =_{E'}$, *d.h.* $T_{spec} = T_{spec'}$.
Ein Regelsystem R *über* sig *ist äquivalent zu* E, *falls* $=_E = \overset{*}{\longleftrightarrow}_R$.

Beachte: Ist R endlich, konvegent, äquivalent zu E so ist die E Gleichheit entscheidbar

$$s =_E t \quad gdw \; s\downarrow = t\downarrow \text{ d.h. identische NF}$$

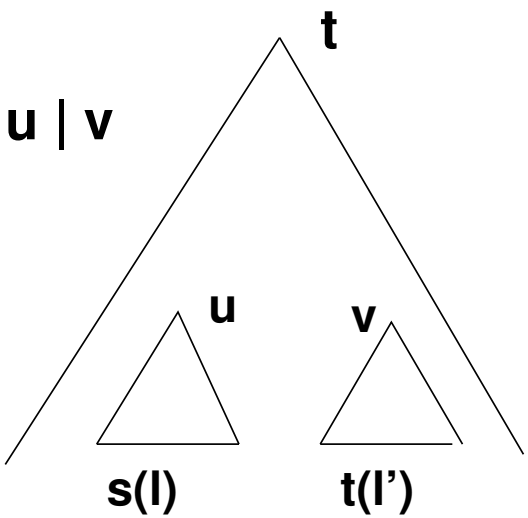
Für funktionale Programme und Berechnungen in T_{spec} genügt die Grundkonvergenz, d.h. Konvergenz auf Grundtermen.
Probleme: Entscheide ob

- R noethersch (grundnoethersch)
- R konfluent (grundkonfluent)
- R Wie transformiert man E in äquivalentes R mit diesen Eigenschaften?

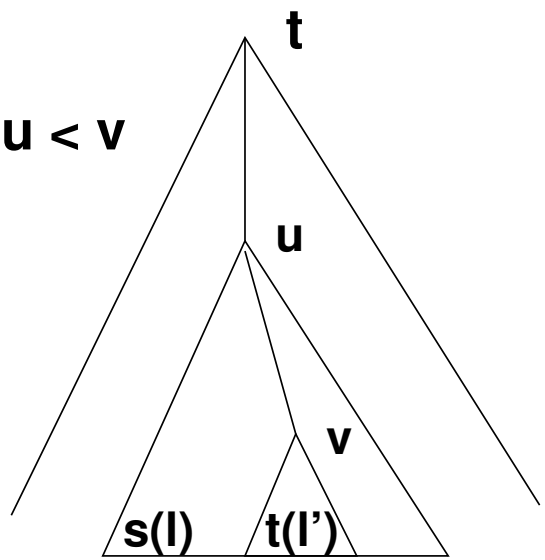
Entscheidbarkeitsfragen

Für endliche Grundtermersetzungssysteme sind die Probleme entscheidbar.

Für terminierende Systeme genügt es lokale Konfluenz zu entscheiden, d.h. aus $t_1 \leftarrow t \rightarrow t_2$ folgt $t_1 \downarrow t_2$.



Zusammenführbar



Kritische Paare

Kritische Paare

Betrachte Gruppenaxiome:

$\underbrace{(x' \cdot y') \cdot z}_{l_1} \to x' \cdot (y' \cdot z)$ und $\underbrace{x \cdot x^{-1}}_{l_2} \to 1.$

"Überlappungen" (Superpositionen)

$$\begin{array}{ccc} & (x \cdot x^{-1}) \cdot z & \\ & \swarrow_{l_2} \quad \searrow_{l_1} & \\ 1 \cdot z & & x \cdot (x^{-1} \cdot z) \end{array} \qquad \begin{array}{ccc} & (x \cdot y) \cdot (x \cdot y)^{-1} & \\ & \swarrow_{l_2} \quad \searrow_{l_1} & \\ 1 & & x \cdot (y \cdot (x \cdot y)^{-1}) \end{array}$$

- $l_1|_1$ ist "unifizierbar" mit l_2 mit Substitution
  $\sigma :: \{x' \leftarrow x, y' \leftarrow x^{-1}, x \leftarrow x\} \rightsquigarrow \sigma(l_1|_1) = \sigma(l_2)$
- l_1 "unifizierbar" mit l_2 mit Substitution
  $\sigma :: \{x' \leftarrow x, y' \leftarrow y, z \leftarrow (x \cdot y)^{-1}, x \leftarrow x \cdot y\} \rightsquigarrow \sigma(l_1) = \sigma(l_2)$

Subsumption, Unifikation

Definition 9.6. *Subsumptionsordnung auf Terme:*
$s \preceq t$ *gdw* $\exists \sigma$ *Substitution* : $\sigma(s)$ *Teilterm von* t
$s \approx t$ *gdw* $(s \preceq t \wedge t \preceq s)$
$s \succ t$ *gdw* $(t \preceq s \wedge \neg(s \preceq t))$
$\succeq$ *ist noethersche Partialordnung auf* $Term(F, V)$ *Beweis!.*

Beachte:
$O(\sigma(t)) = O(t) \cup \bigcup_{w \in O(t): t|_w = x \in V} \{wv : v \in O(\sigma(x))\}$

Verträglichkeitseigenschaften:
- $t|_u = t' \rightsquigarrow \sigma(t)|_u = \sigma(t')$
- $t|_u = x \in V \rightsquigarrow \sigma(t)|_{uv} = \sigma(x)|_v \quad (v \in O(\sigma(x)))$
- $\sigma(t)[\sigma(t')]_u = \sigma(t[t']_u)$ für $u \in O(t)$

Definition 9.7. $s, t \in Term(F, V)$ *sind gdw unifizierbar wenn es Substitution* σ *gibt mit* $\sigma(s) = \sigma(t)$. σ *heißt Unifikator von* s *und* t.

Unifikation, Allgemeinster Unifikator

Definition 9.8. *Sei $V' \subseteq V, \sigma, \tau$ Substitutionen.*

- $\sigma \preceq \tau \quad (V')$ *gdw* $\exists \rho$ *Substitution :* $\rho \circ \sigma|_{V'} = \tau|_{V'}$
  Sprechweise: σ ist allgemeiner als τ auf V'
- $\sigma \approx \tau \quad (V')$ *gdw* $\sigma \preceq \tau(V') \wedge \tau \preceq \sigma(V')$
- $\sigma \prec \tau \quad (V')$ *gdw* $\tau \preceq \sigma \quad (V') \wedge \neg(\sigma \preceq \tau \quad (V'))$
- *Beachte: $\prec$ ist noethersche Partialordnung auf den Substitutionen.*

Frage: *Seien s, t unifizierbar.*

- *Gibt es einen allgemeinsten Unifikator $mgu(s,t)$ auf $V = Var(s) \cup Var(t)$.*
  D.h. für Unifikator σ für s, t gilt stets $mgu(s,t) \preceq \sigma(V)$?
- *Ist $mgu(s,t)$ eindeutig? (bis auf Umbenennung von Variablen).*

Unifikationsproblem und seine Lösung

Definition 9.9.

- *Ein Unifikationsproblem besteht aus einer Menge* $E = \{s_i \stackrel{?}{=} t_i : i = 1, ..., n\}$ *von Gleichungen.*
- σ *heißt Lösung (oder Unifikator) falls* $\sigma(s_i) = \sigma(t_i)$ *für* $i = 1, ..., n$.
- *Gilt* $\tau \succeq \sigma(Var(E))$ *für jede Lösung* τ *von* E, *so* $mgu(E) := \sigma$ *allgemeinste Lösung oder allgemeinster Unifikator.*
- $Sol(E)$ *sei die Menge der Lösungen von* E.
  E *und* E' *sind äquivalent, falls* $Sol(E) = Sol(E')$.
- E' *ist in gelöster Form, falls*
  $E' = \{x_j \stackrel{?}{=} t_j : x_i \neq x_j \ (i \neq j),\ x_i \notin Var(t_j) \ (1 \leq i \leq j \leq m)\}$
- E' *ist gelöste Form zu* E, *falls* E' *in gelöster Form und äquivalent zu* E *mit* $Var(E') \subseteq Var(E)$.

Beispiele

Beispiel 9.10. *Betrachte*

- $s = f(x, g(x, a)) \stackrel{?}{=} f(g(y, y), z) = t$
  $\rightsquigarrow x \stackrel{?}{=} g(y, y) \quad g(x, a) \stackrel{?}{=} z$ *split*
  $\rightsquigarrow x \stackrel{?}{=} g(y, y) \quad g(g(y, y), a) \stackrel{?}{=} z$ *merge*
  $\rightsquigarrow \sigma :: x \leftarrow g(y, y) \quad z \leftarrow g(g(y, y), a) \quad y \leftarrow y$
- $f(x, a) \stackrel{?}{=} g(a, z)$ *Unlösbar (nicht unifizierbar).*
- $x \stackrel{?}{=} f(x, y)$ *Unlösbar da $f(x, y)$ nicht x frei.*
- $x \stackrel{?}{=} f(a, y) \rightsquigarrow$ *Lösung $\sigma :: x \leftarrow f(a, y)$ ist allgemeinste Lösung.*

Inferenzsystem für die Unifikation

Definition 9.11. *Kalkül **UNIFY**. Sei σ = Bindungsmenge.*

(1) *Löschen* $$\frac{(E \cup \{s \stackrel{?}{=} s\}, \sigma)}{(E, \sigma)}$$

(2) *Split (Zerlegen)* $$\frac{(E \cup \{f(s_1, ..., s_m) \stackrel{?}{=} g(t_1, ..., t_n)\}, \sigma)}{\lightning \text{ (Unlösbar)}} \text{ falls } f \neq g$$

$$\frac{(E \cup \{f(s_1, ..., s_m) \stackrel{?}{=} f(t_1, ..., t_m)\}, \sigma)}{(E \cup \{s_i \stackrel{?}{=} t_i : i = 1, ..., m\}, \sigma)}$$

(3) *Merge (Lösen)* $$\frac{(E \cup \{x = t\}, \sigma)}{(\tau(E), \sigma \cup \tau)} \text{ falls } x \notin Var(t), \tau = \{x \stackrel{?}{=} t\}$$

"occur check" $$\frac{(E \cup \{x = t\}, \sigma)}{\lightning \text{ (Unlösbar)}} \text{ falls } x \in Var(t) \wedge x \neq t$$

Unifikationsalgorithmen

Unifikationsalgorithmen basierend auf diesem Kalkül starten stets mit $(E_0, S_0) := (E, \emptyset)$ liefern eine Folge $(E_0, S_0) \vdash_{UNIFY} ... \vdash_{UNIFY} (E_n, S_n)$ Sie sind erfolgreich falls sie mit $E_n = \emptyset$, unerfolgreich falls sie mit $S_n = \lightning$ enden. S_n beschreibt eine Substitution σ die $Sol(S_n)$ und somit auch $Sol(E)$ erfasst.

Lemma 9.12. *Korrektheit.*
Jede Folge $(E_0, S_0) \vdash_{UNIFY} ... \vdash_{UNIFY} (E_n, S_n)$ bricht ab: Entweder mit $\lightning$ (Unlösbar, nicht unifizierbar) oder mit $(\emptyset, S)$ und S ist eine gelöste Form zu E.

Beachte: Darstellungen der Substitution in gelöster Form können recht unterschiedlich sein (Aufwand!!):
$s \stackrel{?}{=} f(x_1, ..., x_n) \qquad t \stackrel{?}{=} f(g(x_0, x_0), ..., g(x_{n-1}, x_{n-1}))$
$S = \{x_i \stackrel{?}{=} g(x_{i-1}, x_{i-1}) : i = 1, ..., n\}$ und
$S_1 = \{x_{i+1} \stackrel{?}{=} t_i : t_0 = g(x_0, x_0), t_{i+1} = g(t_i, t_i)\ i = 0, ..., n-1\}$
sind beide in gelöster Form. Die Länge der t_i wächst exponentiell mit i.

Beispiel

Beispiel 9.13. *Ausführung:*

$$f(x, g(a, b)) \stackrel{?}{=} f(g(y, b), x)$$

E_i	S_i	*Regel*
$f(x, g(a, b)) \stackrel{?}{=} f(g(y, b), x)$	$\emptyset$	
$x \stackrel{?}{=} g(y, b), x \stackrel{?}{=} g(a, b)$	$\emptyset$	*split*
$g(y, b) \stackrel{?}{=} g(a, b)$	$x \stackrel{?}{=} g(a, b)$	*lösen*
$y \stackrel{?}{=} a, b \stackrel{?}{=} b$	$x \stackrel{?}{=} g(a, b)$	*split*
$b \stackrel{?}{=} b$	$x \stackrel{?}{=} g(a, b), y \stackrel{?}{=} a$	*lösen*
	$x \stackrel{?}{=} g(a, b), y \stackrel{?}{=} a$	*löschen*

Lösung: $mgu = \sigma = \{x \leftarrow g(a, b), y \leftarrow a\}$

Kritische Paare - Lokale Konfluenz

Definition 9.14. *Sei R Regelsystem und $l_1 \to r_1, l_2 \to r_2 \in R$ mit $V(l_1) \cap V(l_2) = \emptyset$ (Umbenennung von Variablen falls nötig, u.U. $l_1 \approx l_2$ bzw. $l_1 \to r_1 \approx l_2 \to r_2$).*

Sei $u \in O(l_1)$ mit $l_1|_u \notin V, \sigma = mgu(l_1|_u, l_2)$ existiere.

$\sigma(l_1)$ heißt dann eine Überlappung (Superposition) von $l_2 \to r_2$ in $l_1 \to r_1$ und $(\sigma(r_1), \sigma(l_1[r_2]_u))$ ist das zugehörige kritische Paar zu $l_1 \to r_1, l_2 \to r_2, u \in O(l_1)$, sofern $\sigma(r_1) \neq \sigma(l_1[r_2]_u)$.

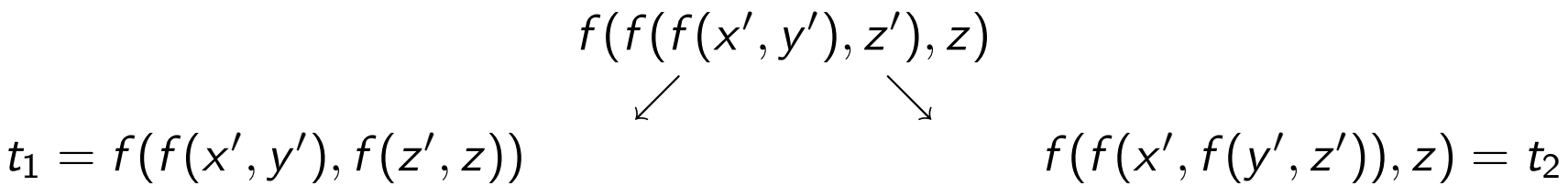
$CP(R)$ sei die Menge aller kritischen Paare die man mit Regeln aus R bilden kann.

Beachte: Die Überlappung und somit die Menge der kritischen Paare ist bis auf Umbenennung der Variablen eindeutig.

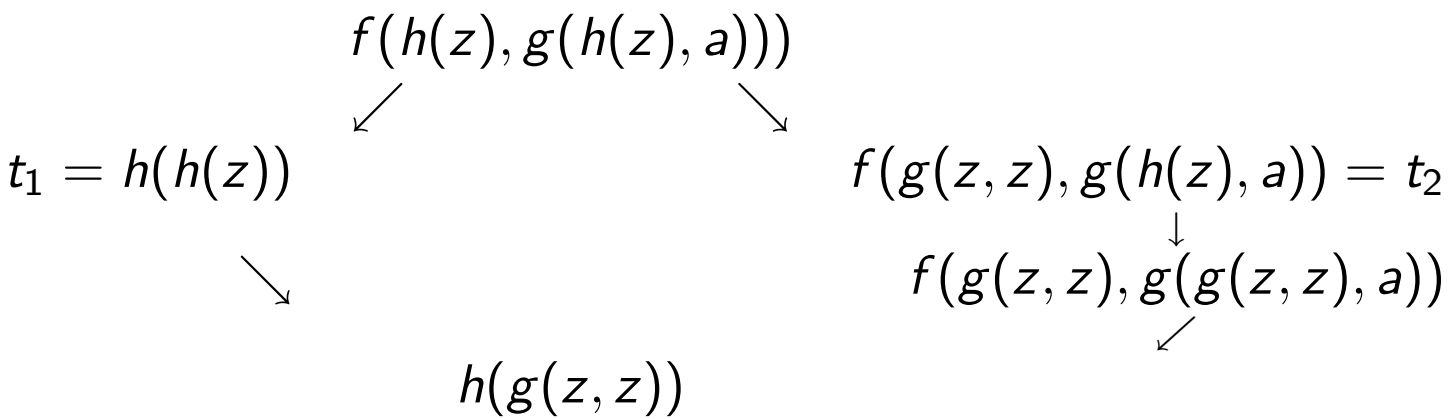
Beispiele

Beispiel 9.15. *Betrachte*

- $f(f(\underline{x}, \underline{y}), z) \to f(x, f(y, z))$ $\quad f(\underline{f(x', y')}, \underline{z'}) \to f(x', f(y', z'))$
  Unifizierbar mit $x \leftarrow f(x', y'), y \leftarrow z'$



- $t = f(x, g(x, a)) \to h(x)$ $\quad h(x') \to g(x', x'), t|_1 = t|_{21} = x$
  Keine kritischen Paare. Betrachte Variablenüberlappung:



Eigenschaften

- Seien σ, τ Substitutionen, $x \in V$, $\sigma(y) = \tau(y)$ für $y \neq x$ und $\sigma(x) \to_R \tau(x)$. Dann gilt für jeden Term t:

$$\sigma(t) \overset{*}{\to}_R \tau(t)$$

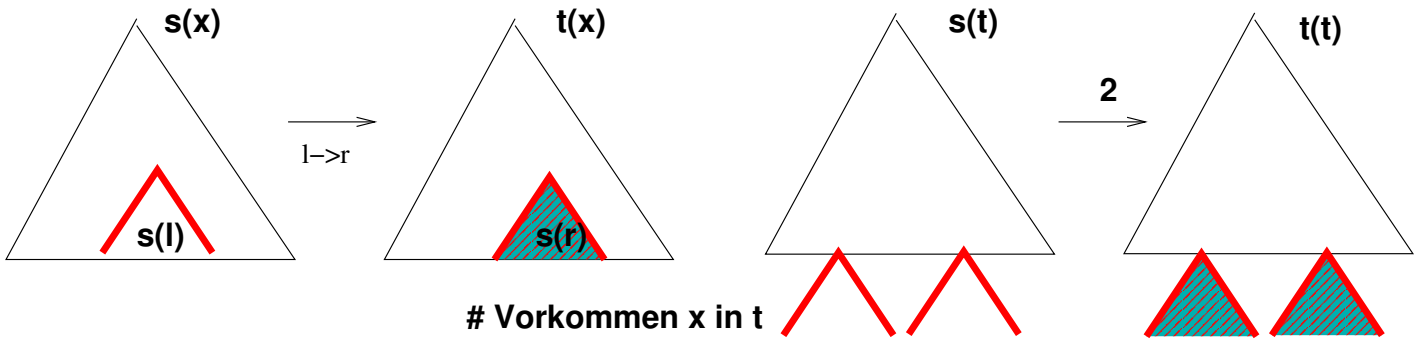
- Seien $l_1 \to r_1, l_2 \to r_2$ Regeln, $u \in O(l_1), l_1|_u = x \in V$. Sei $\sigma(x)|_w = \sigma(l_2)$, d.h. $\sigma(l_2)$ wird durch $\sigma(x)$ eingeführt. Dann gilt $t_1 \downarrow_R t_2$ für

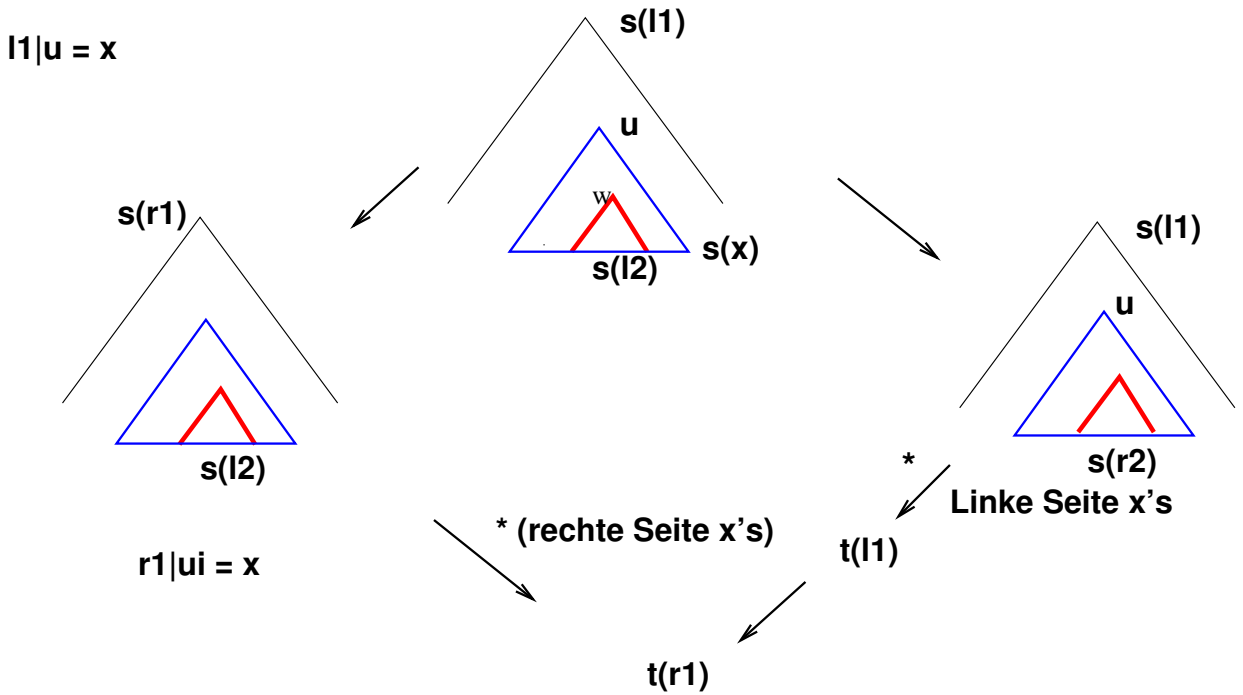
$$t_1 := \sigma(r_1) \leftarrow \sigma(l_1) \to \sigma(l_1)[\sigma(r_2)]_{uw} =: t_2$$

Lemma 9.16. *Critical-Pair Lemma von Knuth/Bendix*
Sei R ein Regelsystem. Dann gilt:

Aus $t_1 \leftarrow_R t \to_R t_2$ folgt $t_1 \downarrow_R t_2$ oder $t_1 \leftrightarrow_{CP(R)} t_2$.

Beweise


s(x)
t(x)
s(t)
t(t)
l->r
s(l)
s(r)
2
Vorkommen x in t


l1|u = x
s(l1)
u
w
s(x)
s(l2)
s(r1)
s(l2)
s(l1)
u
s(r2)
*
Linke Seite x's
* (rechte Seite x's)
t(l1)
r1|ui = x
t(r1)

Konfluenztest

Satz 9.17. *Hauptergebniss: Sei R ein Regelsystem.*

- *R ist genau dann lokal konfluent, wenn alle Paare $(t_1, t_2) \in CP(R)$ zusammenführbar sind.*
- *Ist R terminierend, so gilt:*
  R konfluent gdw. $(t_1, t_2) \in CP(R) \rightsquigarrow t_1 \downarrow t_2$.
- *Sei R linear (d.h. für $l, r \in l \to r \in R$ kommen Variablen höchstens einmal vor). Gilt $CP(R) = \emptyset$, so ist R konfluent.*

Beispiel 9.18.

- *Sei $R = \{f(x,x) \to a, f(x,s(x)) \to b, a \to s(a)\}$.*
  R ist lokal konfluent,aber nicht konfluent:

$$a \leftarrow f(a,a) \to f(a,s(a)) \to b$$

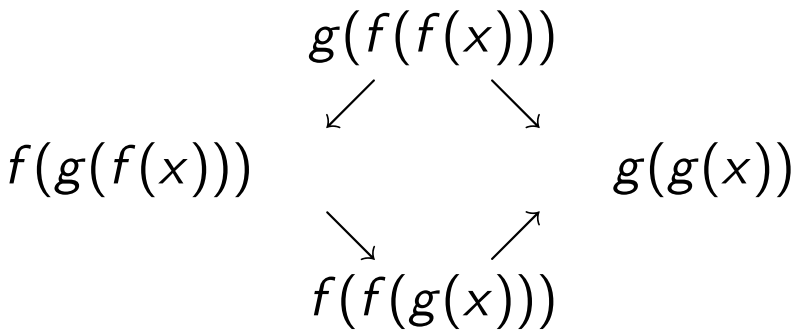
aber nicht $a \downarrow b$. R ist weder terminierend noch links-linear.

Beispiel (Fort.)

- $R = \{f(f(x)) \to g(x)\}$

  $$t_1 = g(f(x)) \leftarrow f(f(f(x))) \to f(g(x)) = t_2$$

  Es gilt nicht, $t_1 \downarrow_R t_2 \rightsquigarrow R$ nicht konfluent.
  Füge Regel $t_1 \to t_2$ zu R hinzu. R_1 ist äquivalent zu R, terminierend und konfluent.



- $R = \{x + 0 \to x, x + s(y) \to s(x + y)\}$,linear ohne krit. Paare $\rightsquigarrow$ confluent.
- $R = \{f(x) \to a, f(x) \to g(f(x)), g(f(x)) \to f(h(x)), g(f(x)) \to b\}$ ist lokal konfluent aber nicht konfluent.

Konfluenz ohne Terminierung

Definition 9.19. *$\epsilon - \epsilon$ - Eigenschaften. Sei* $\xrightarrow{\epsilon} = \xrightarrow{0} \cup \xrightarrow{1}$.

- *R heißt $\epsilon - \epsilon$ abgeschlossen , falls für jedes kritische Paar $(t_1, t_2) \in CP(R)$ es ein t gibt mit $t_1 \xrightarrow[R]{\epsilon} t \xleftarrow[R]{\epsilon} t_2$.*
- *R heißt $\epsilon - \epsilon$ konfluent gdw.* $\xleftarrow[R]{} \circ \xrightarrow[R]{} \subseteq \xrightarrow[R]{\epsilon} \circ \xleftarrow[R]{\epsilon}$

Folgerung 9.20.

- $\rightarrow$ *$\epsilon - \epsilon$ konfluent* $\rightsquigarrow$ $\rightarrow$ *streng konfluent.*
- *R $\epsilon - \epsilon$ abgeschlossen* $\not\Rightarrow$ *R $\epsilon - \epsilon$ konfluent*
  $R = \{f(x,x) \rightarrow a, f(x, g(x)) \rightarrow b, c \rightarrow g(c)\}$. $CP(R) = \emptyset$, d.h. R $\epsilon - \epsilon$ abgeschlossen aber $a \leftarrow f(c,c) \rightarrow f(c, g(c)) \rightarrow b$, d.h. R nicht konfluent $\lightning$.
- *Ist R linear und $\epsilon - \epsilon$ abgeschlossen , dann ist R streng konfluent also konfluent (Zeige R ist $\epsilon - \epsilon$ konfluent).*

Diese Bedingungen sind leider zu einschränkend für die Programmierung.

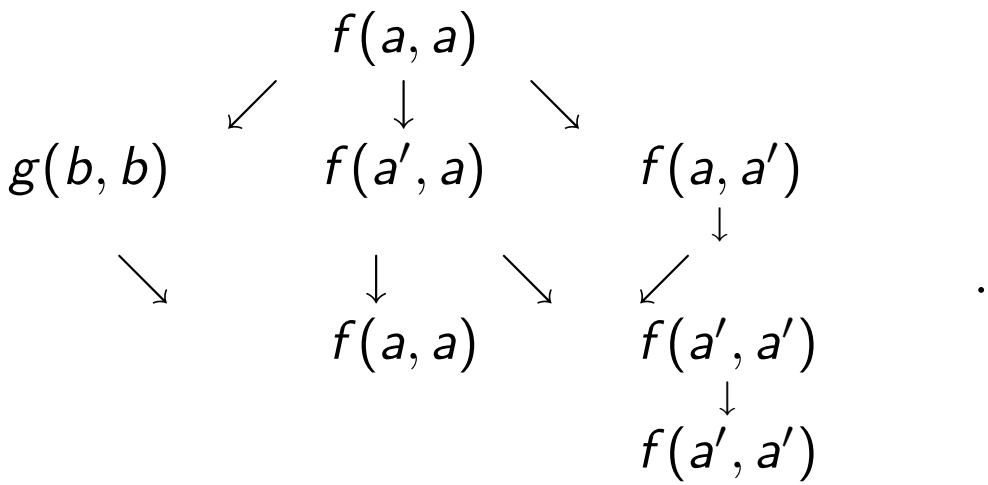
Beispiel

Beispiel 9.21. *R links linear $\epsilon - \epsilon$ abgeschlossen reicht nicht aus:*

$$R = \{f(a,a) \to g(b,b), a \to a', f(a',x) \to f(x,x), f(x,a') \to f(x,x),$$
$$g(b,b) \to f(a,a), b \to b', g(b',x) \to g(x,x), g(x,b') \to g(x,x)\}$$

Es gilt $f(a',a') \xleftrightarrow[R]{*} g(b',b')$ *jedoch nicht* $f(a',a') \downarrow_R g(b',b')$.

R links linear $\epsilon - \epsilon$ abgeschlossen :



Parallel Reduktion

Beachte: Seien $\rightarrow, \Rightarrow$ mit $\overset{*}{\rightarrow} = \overset{*}{\Rightarrow}$. (Oft: $\rightarrow \subseteq \Rightarrow \subseteq \overset{*}{\rightarrow}$).
Dann ist $\rightarrow$ konfluent gdw. $\Rightarrow$ konfluent.

Definition 9.22. *Sei R Regelsystem.*

- *Die Parallelreduktion, $\mapsto_R$, ist definiert durch $t \mapsto_R t'$ gdw. $\exists U \subset O(t) : \forall u_i, u_j (u_i \neq u_j \rightsquigarrow u_i | u_j)$ $\exists l_i \rightarrow r_i \in R, \sigma_i$ mit $t|_{u_i} = \sigma_i(l_i) :: t' = t[\sigma_i(r_i)]_{u_i} (u_i \in U)$ $(t[u_1 \leftarrow \sigma_1(r_1)] \ldots t[u_n \leftarrow \sigma_1(r_n)])$.*
- *Ein kritisches Paar von R : $(\sigma(r_1), \sigma(l_1[r_2]_u)$ ist parallel 0-zusammenführbar falls $\sigma(l_1[r_2]_u) \mapsto_R \sigma(r_1)$.*
- *R ist parallel 0-abgeschlossen falls jedes kritische Paar von R parallel 0-zusammenführbar ist.*

Eigenschaften: $\mapsto_R$ ist stabil und monoton. Es gilt $\overset{*}{\mapsto}_R = \overset{*}{\rightarrow}_R$ und somit ist $\mapsto_R$ konfluent, so auch $\rightarrow_R$.

Parallel Reduktion

Satz 9.23. *Ist R links-linear und parallel 0-abgeschlossen, so ist $\mapsto_R$ streng konfluent, also konfluent, und somit ist auch R konfluent.*

Folgerung 9.24.

- *Erfüllt R die O'Donnel Bedingungen, so ist R konfluent.*
  O'Donnel Bedingungen: R links-linear, $CP(R) = \emptyset$, R links-sequentiell
  (Redexe sind beim Lesen der Terme von links nach rechts eindeutig: $f(g(x,a),y) \to 0, g(b,c) \to 1$ hat diese Eigenschaft nicht. Durch Umgruppieren der Argumente kann oft die Eigenschaft erreicht werden, etwa $f(g(a,x),y) \to 0, g(b,c) \to 1$)
- *Orthogonale Systeme:: R links-linear und $CP(R) = \emptyset$, so R konfluent. (In Lit. auch als reguläre Systeme bezeichnet).*
- *Variationen: R ist stark-abgeschlossen, falls für jedes kritische Paar (s,t) is Terme u, v gibt mit $s \xrightarrow{*} u \xleftarrow{\leq 1} t$ und $s \xrightarrow{\leq 1} v \xleftarrow{*} t$.*
  R linear und stark-abgeschlossen, so R streng-konfluent.

Fogerungen

- Folgt aus $CP(R) = \emptyset$ die Konfluenz? Nein.
  $R = \{f(x,x) \to a, g(x) \to f(x, g(x)), b \to g(b)\}$.
  Betrachte $g(b) \to f(b, g(b)) \to f(g(b), g(b)) \to a$
  "Outermost" Reduktion.
  $g(b) \to g(g(b)) \overset{*}{\to} g(a) \to f(a, g(a))$ nicht zusammenführbar.
- Reguläre Systeme können nicht terminierend sein:
  $\{f(x,b) \to d, a \to b, c \to c\}$. Offenbar $CP = \emptyset$.
  $f(c,a) \to f(c,b) \to d$
  $\downarrow *$
  $f(c,a) \to f(c,b)$. Beachte $f(c,a)$ hat eine Normalform.$\rightsquigarrow$
  Reduktionsstrategien die Normalformberechnend sind oder
  kürzeste Berechnungen liefern.
- Ein Kontext ist ein Term mit "Lücken" $\Box$, z.B.
  $f(g(\Box, s(0)), \Box, h(\Box))$ als "Baummuster" (pattern) zu Regel
  $f(g(x, s(0)), y, h(z)) \to x$. Lücken dürfen beliebig gefüllt werden.

Terminierungs-Kriterien

Satz 9.25. *R ist genau dann terminierend, wenn es eine noethersche Partialordnung $\succ$ auf den Grundtermen $Term(F)$ gibt die monoton ist, so dass gilt $\sigma(l) \succ \sigma(r)$ für jede Regel $l \to r \in R$ und Grundsubstitution σ.*

Beweis: $\curvearrowright$ Definiere $s \succ t$ gdw. $s \xrightarrow{+} t$ $(s, t \in Term(F))$
$\curvearrowleft$ Ang. $\to_R$ nicht terminierend, $t_0 \to t_1 \to \ldots (V(t_i) \subseteq V(t_0))$. Sei σ eine Grundsubstitution mit $V(t_0) \subset D(\sigma)$, dann $\sigma(t_0) \succ \sigma(t_1) \succ \ldots \lightning$.
Problem: Unendlicher Test.

Definition 9.26. *Eine **Reduktionsordnung** ist Partialordnung $\succ$ auf $Term(F, V)$ mit*
(i) $\succ$ ist Noethersch (ii) $\succ$ ist stabil und (iii) $\succ$ ist monoton.

Satz 9.27. *R ist genau dann Noethersch, wenn es eine Reduktionsordnung $\succ$ gibt mit $l \succ r$ für alle $l \to r \in R$*

Terminierungs-Kriterien

Beachte: Es gibt keine totalen Reduktionsordnungen.
$x \succ y ? \rightsquigarrow \sigma(x) \succ \sigma(y)$
$f(x,y) \succ f(y,x)$? Kommutativität kann nicht gerichtet werden.
Beispiele für Reduktionsordnungen:
Knuth-Bendix Ordnungen: Gewicht für jedes Funktionssymbol und Präzedenz auf F.
Rekursive Pfad Ordnungen (RPO): Präzedenz auf F wird rekursiv auf Pfade (Wörter) in den zu vergleichenden Termen fortgesetzt.
Lexikographische Pfad Ordnungen(LPO), Polynominterpretationen, usw.

$f(f(g(x)))$	$=$	$f(h(x))$	$f(f(x))$	$=$	$g(h(g(x)))$	$f(h(x))$	$=$	$h(g(x$
KB	$\rightarrow$	$l(f)=3$	$l(g)=2$	$\rightarrow$	$l(h)=$	1	$\rightarrow$	
RPO	$\leftarrow$	$g > h$	$> f$	$\leftarrow$			$\leftarrow$	

Konfluenz Modulo Äquivalenzrelation (z.B. AC):
$R :: f(x,x) \rightarrow g(x)$ $G :: \{(a,b)\}$ $g(a) \leftarrow f(a,a) \sim f(a,b)$ jedoch nicht $g(a) \downarrow_{\sim} f(a,b)$.

Knuth-Bendix Vervollständigungsverfahren

Eingabe: E Gleichungsmenge, $\succ$ Reduktionsordnung, $R = \emptyset$.

Repeat while E nicht leer
(1) Entferne $t = s$ aus E mit $t \succ s$, $R := R \cup \{t \to s\}$ sonst abort
(2) Bringe rechte Seite der Regeln in Normalform mit R
(3) Erweitere E um alle mit R normalisierten kritischen Paare die $t \to s$ mit R bildet
(4) Entferne alle Regeln aus R, deren linke Seite eine echte Instanz von t enthalten.
(5) Verwende R um beide Seiten von Gleichungen aus E zu Normalisieren. Entferne Identitäten.

Ausgang: Terminierung mit R konvergent , äquivalent zu E. Abbruch (abort), Nicht Terminierung (läuft unendlich lange).

Beispiele für Knuth-Bendix-V

Beispiel 9.28.

- *WES::* $\Sigma = \{a, b, c\}, E = \{a^2 = \lambda, b^2 = \lambda, ab = c\}$
  $u < v$ *gdw.* $|u| < |v|$ *oder* $|u| = |v|$ *und* $u <_{lex} v$ *mit* $a <_{lex} b <_{lex} c$

  $E_0 = \{a^2 = \lambda, b^2 = \lambda, ab = c\}, R_0 = \emptyset$

  $E_1 = \{b^2 = \lambda, ab = c\}, R_1 = \{a^2 \to \lambda\}, CP_1 = \emptyset$

  $E_2 = \{ab = c\}, R_2 = \{a^2 \to \lambda, b^2 \to \lambda\}, CP_2 = \emptyset$

  $R_3 = \{a^2 \to \lambda, b^2 \to \lambda, ab \to c\}, NCP_3 = \{(b, ac), (a, cb)\}$
  $E_3 = \{b = ac, a = cb\}$

  $R_4 = \{a^2 \to \lambda, b^2 \to \lambda, ab \to c, ac \to b\}, NCP_4 = \emptyset, E_4 = \{a = cb\}$

  $R_5 = \{a^2 \to \lambda, b^2 \to \lambda, ab \to c, ac \to b, cb \to a\}, NCP_5 = \emptyset, E_5 = \emptyset$

Beispiele für Knuth-Bendix-V

- $E = \{ffg(x) = h(x), ff(x) = x, fh(x) = g(x)\} \quad >: KBO(3,2,1)$
  $R_0 = \emptyset, E_0 = E$
  $R_1 = \{ffg(x) \rightarrow h(x)\}, KP_1 = \emptyset. E_1 = \{ff(x) = x, fh(x) = g(x)\}$
  $R_2 = \{ffg(x) \rightarrow h(x), ff(x) \rightarrow x\}, NKP_2 = \{(g(x), h(x))\},$
  $E_2 = \{fh(x) = g(x), g(x) = h(x)\}, R_2 = \{ff(x) \rightarrow x\}$
  $R_3 = \{ff(x) \rightarrow x, fh(x) \rightarrow g(x)\}, NKP_3 = \{(h(x), fg(x))\}, E_3 =$
  $\{g(x) = h(x), h(x) = fg(x)\}$
  $R_4 = \{ff(x) \rightarrow x, fh(x) \rightarrow h(x), g(x) \rightarrow h(x)\}, NKP_3 = \emptyset, E_4 = \emptyset$
- $E = \{fgf(x) = gfg(x)\} \quad >: LL :: f > g$
  $R_0 = \emptyset, E_0 = E$
  $R_1 = \{fgf(x) \rightarrow gfg(x)\}, NKP_1 = \{(gfggf(x), fggfg(x))\}, E_1 =$
  $\{gfggf(x) = fggfg(x)\}$
  $R_1 = \{fgf(x) \rightarrow gfg(x), fggfg(x) \rightarrow gfggf(x)\}, NKP_2 =$
  $\{(gfggfggfg(x), fgggfggfg(x), ..\}...$

Verfeinertes Inferenzsystem

Definition 9.29. *Sei $>$ eine Noethersche PO auf $Term(F, V)$. Das Inferenzsystem $\mathcal{P}_{TES}$ besteht aus folgenden Regeln:*

(1) *Orientieren* $\dfrac{(E \cup \{s \doteq t\}, R)}{(E, R \cup \{s \to t\})}$ *falls* $s > t$

(2) *Generieren* $\dfrac{(E, R)}{(E \cup \{s \doteq t\}, R)}$ *falls* $s \leftarrow_R \circ \to_R t$

(3) *Simplifizieren GL* $\dfrac{(E \cup \{s \doteq t\}, R)}{(E \cup \{u \doteq t\}, R)}$ *falls* $s \to_R u$

(4) *Simplifizieren RS* $\dfrac{(E, R \cup \{s \to t\})}{(E, R \cup \{s \to u\))}$ *falls* $t \to_R u$

(5) *Simplifizieren LS* $\dfrac{(E, R \cup \{s \to t\})}{(E \cup \{u \doteq t\}, R)}$ *falls* $s \to_R u$ *mit* $l \to r$ *und* $s \succ l$ *(SubSumOrd.)*

(6) *Löschen Identitäten*

Gleichungsimplementierungen

Programmierung = Beschreibung von Algorithmen in einem Formalen System

Definition 10.1. *Sei* $f : M_1 \times ... \times M_n \rightsquigarrow M_{n+1}$ *eine (partielle) Funktion.*
Seien $T_i, 1 = 1...n+1$ *entscheidbare Grundtermmengen über* Σ,
$\hat{f}$ *n-stelliges Funktionssymbol,* E *Gleichungsmenge.*
Eine Dateninterpretation $\mathfrak{I}$ *ist eine Funktion* $\mathfrak{I} : T_i \to M_i$.

$\hat{f}$ *implementiert* f *unter der Interpretation* $\mathfrak{I}$ *in* E *gdw*
1) $\mathfrak{I}(T_i) = M_i \qquad (i = 1...n+1)$
2) $f(\mathfrak{I}(t_1), ..., \mathfrak{I}(t_n)) = \mathfrak{I}(t_{n+1})$ *gdw* $\hat{f}(t_1, ..., t_n) =_E t_{n+1}$ $(\forall t_i \in T_i)$

$$\begin{array}{ccc} T_1 \times ... \times T_n & \xrightarrow{\hat{f}} & T_{n+1} \\ \mathfrak{I} \downarrow \qquad\quad \mathfrak{I} \downarrow & & \mathfrak{I} \downarrow \\ M_1 \times ... \times M_n & \xrightarrow{f} & M_{n+1} \end{array}$$

Abkürzung: $(\hat{f}, E, \mathfrak{I})$ *implementiert* f.

Gleichungsimplementierungen

Satz 10.2. *E sei Gleichungs- oder Regelmenge. Bezeichnungen wie eben. Für alle $i = 1, ..., n+1$ gelte*
1) $\mathfrak{I}(T_i) = M_i$
2a) $f(\mathfrak{I}(t_1), ..., \mathfrak{I}(t_n)) = \mathfrak{I}(t_{n+1}) \rightsquigarrow \hat{f}(t_1, ..., t_n) =_E t_{n+1}\ (\forall t_i \in T_i)$

$\hat{f}$ implementiert die totale Funktion f unter $\mathfrak{I}$ in E wenn eine der folgenden Bedingungen gilt:

a) $\forall t, t' \in T_{n+1} : t =_E t' \rightsquigarrow \mathfrak{I}(t) = \mathfrak{I}(t')$
b) E *konfluent und* $\forall t \in T_{n+1} : t \to_E t' \rightsquigarrow t' \in T_{n+1} \wedge \mathfrak{I}(t) = \mathfrak{I}(t')$
c) E *konfluent und* T_{n+1} *enthält nur E-irreduzible Terme.*

Anwendung: $(\hat{f}, E, \mathfrak{I})$ implementiere die totale Funktion f. Wird E um E_0 Erweitert unter Beibehaltung von $\mathfrak{I}$, so gelten weiterhin 1 und 2a. Ist eines der Kriterien a, b, c für $E \cup E_0$ erfüllt, so implementiert auch $(\hat{f}, E \cup E_0, \mathfrak{I})$ die Funktion f. Dies gilt insbesondere wenn $E \cup E_0$ konfluent ist und T_{n+1} enthält nur $E \cup E_0$ irreduzible Terme.

Gleichungsimplementierungen

Satz 10.3. *$(\hat{f}, E, \mathfrak{I})$ implementiere die (partielle) Funktion f. Dann gilt:*

a) $\forall t, t' \in T_{n+1} :: \mathfrak{I}(t) = \mathfrak{I}(t') \wedge \mathfrak{I}(t) \in Bild(f) \rightsquigarrow t =_E t'$
b) Sei E konfluent und T_{n+1} enthalte nur E Normalformen. Dann ist $\mathfrak{I}$ injektiv auf $\{t \in T_{n+1} : \mathfrak{I}(t) \in Bild(f)\}$.

Satz 10.4. *Kriterium zur Implementierung totaler Funktionen. Es gelte*

1) $\mathfrak{I}(T_i) = M_i \quad (i = 1, ..., n+1)$
2) $\forall t, t' \in T_{n+1} :: \mathfrak{I}(t) = \mathfrak{I}(t')$ *gdw* $t =_E t'$
3) $\forall_{1 \leq i \leq n} \quad t_i \in T_i \quad \exists t_{n+1} \in T_{n+1} ::$

$$\hat{f}(t_1, ..., t_n) =_E t_{n+1} \wedge f(\mathfrak{I}(t_1), ...\mathfrak{I}(t_n)) = \mathfrak{I}(t_{n+1})$$

Dann implementiert $\hat{f}$ die Funktion f unter $\mathfrak{I}$ in E und f ist total.

Beachte: Enthält T_{n+1} nur Normalformen und ist E konfluent, so ist 2) erfüllt, wenn $\mathfrak{I}$ auf T_{n+1} injektiv ist.

Gleichungsimplementierungen

Satz 10.5. *$(\hat{f}, E, \mathfrak{I})$ implementiere $f : M_1 \times ... \times M_n \to M_{n+1}$. Seien $S_i = \{t \in T_i :: \exists t_0 \in T_i : t \neq t_0, \mathfrak{I}(t) = \mathfrak{I}(t_0) \;\; t \stackrel{+}{\to}_E t_0\}$ rekursive Mengen.*
Dann implementiert $\hat{f}$ mit Termmengen $T_i' = T_i \backslash S_i$ ebenfalls f unter $\mathfrak{I}|_{T_i'}$ in E.

Man kann also aus T_i Terme streichen die auf andere Terme aus T_i mit gleichem $\mathfrak{I}$-Wert reduzierbar sind. Erlaubt somit Einschränkung auf E-Normalformen.

Folgerung 10.6.

- *Man kann Implementierungen Komponieren.*
- *Erweitert man E um E Folgerungsgleichungen, dann bleibt die Implementierungseigenshaft erhalten.*
  Wichtig für die KB-Vervollständigung da nur E-Folgerungen hinzugenommen werden.

Beispiele: Aussagenlogik, Natürliche Zahlen

Beispiel 10.7. *Konvention: Gleichungen legen die Signatur fest. Gelegentlich variable Stellenzahl und Overloading. Einsortig.*

Boolsche Algebra: Sei $M = \{true, false\}$ *mit* $\wedge, \vee, \neg, \supset, ...$ *Konstanten* tt, ff *Termmenge Bool* $:= \{tt, ff\}$, $\mathfrak{I}(tt) = true, \mathfrak{I}(ff) = false$.
Strategie: Vermeide Regeln mit tt *oder* ff *als linke Seite. Nach Satz 10.1 c) kann man Gleichungen mit diesen Einschränkungen hinzunehmen ohne die Implementierungseigenschaft zu beeinflüßen, sofern Konfluenz erreicht wird. Betrachte folgende Regeln:*

(1) $cond(tt, x, y) \to x$ *(2)* $cond(ff, x, y) \to y$. *(Hilfsfunktion).*
(3) $x \ vel \ y \to cond(x, tt, y)$
$E = \{(1), (2), (3)\}$ *ist konfluent. Es gilt:*
$tt \ vel \ y =_E cond(tt, tt, y) =_E tt$ *d.h.*
$(*_1)$ $tt \ vel \ y = tt$ *und* $(*_2)$ $x \ vel \ tt = cond(x, tt, tt)$
$x \ vel \ tt = tt$ *läßt sich nicht aus* E *ableiten.*
Dennoch implementiert vel die Funktion $\vee$ *mit diesem* E.

Beispiele: Aussagenlogik

Nach Satz 10.4 sind Bedingungen (1), (2), (3) zu zeigen:
$\forall t, t' \in \mathit{Bool}\ \exists \bar{t} \in \mathit{Bool} :: \mathfrak{I}(t) \vee \mathfrak{I}(t') = \mathfrak{I}(\bar{t}) \wedge t\ \mathit{vel}\ t' =_E \bar{t}$

Für $t = tt$ $(*_1)$ und $t = ff$ (2) da $ff\ \mathit{vel}\ t' \to_E \mathit{cond}(ff, tt, t') \to_E t'$
D.h. $x\ \mathit{vel}\ tt \neq_E tt$ aber $tt\ \mathit{vel}\ tt =_E tt$, $ff\ \mathit{vel}\ tt =_E tt$.

MC Carthy's Regeln für *cond*:

(1) $\mathit{cond}(tt, x, y) = x$ (2) $\mathit{cond}(ff, x, y) = y$ (*) $\mathit{cond}(x, tt, tt) = tt$
Beachte: Nicht identisch mit *cond* in Lisp.
Unterschied: Reihenfolge der Auswertung. Betrachte
(**) $\mathit{cond}(x, \mathit{cond}(x, y, z), u) \to \mathit{cond}(x, y, u) \rightsquigarrow$
$E' = \{(1), (2), (3), (*), (**)\}$ ist terminierend und konfluent.
Vereinbarungen: In Gleichungsmengen sind stets (1), (2), (3) und
$x\ et\ y \to \mathit{cond}(x, y, ff)$ enthalten.
Schreibweise: $\mathit{cond}(x, y, z) :: [x \to y, z]$ bzw.
$[x \to y_1, x_2 \to y_2, ..., x_n \to y_n, z]$ für $[x \to [...]..., z]$

Beispiele: Semantische Argumente

Eigenschaften implementierender Funktionen:
$(vel, E, \mathfrak{I})$ implementiert $\vee$ von BOOL.

Beh: *vel* ist assoziativ auf *Bool*.
Z.Z. $\forall t_1, t_2, t_3 \in Bool : t_1\ vel\ (t_2\ vel\ t_3) =_E (t_1\ vel\ t_2)\ vel\ t_3$

Es gibt $t, t', T, T' \in Bool$ mit
$\mathfrak{I}(t_2) \vee \mathfrak{I}(t_3) = \mathfrak{I}(t)$ und $\mathfrak{I}(t_1) \vee \mathfrak{I}(t_2) = \mathfrak{I}(t')$ sowie
$\mathfrak{I}(t_1) \vee \mathfrak{I}(t) = \mathfrak{I}(T)$ und $\mathfrak{I}(t') \vee \mathfrak{I}(t_3) = \mathfrak{I}(T')$

Wegen der semantisch gültigen Assoziativität von $\vee$ gilt
$\mathfrak{I}(T) = \mathfrak{I}(t_1) \vee \mathfrak{I}(t_2) \vee \mathfrak{I}(t_3) = \mathfrak{I}(T')$

Da *vel* $\vee$ implementiert folgt nun:
$t_1\ vel\ (t_2\ vel\ t_3) =_E t_1\ vel\ t =_E T =_E T' =_E t'\ vel\ t_3 =_E (t_1\ vel\ t_2)\ vel\ t_3$

Beispiele: Natürliche Zahlen

Funktionssymbole: $\hat{0}, \hat{s}$ Grundterme: $\{\hat{s}^n(\hat{0})\ (n \geq 0)\}$
$\mathfrak{I}$ Interpretation $\mathfrak{I}(\hat{0}) = 0, \mathfrak{I}(\hat{s}) = \lambda x.x + 1$, d.h. $\mathfrak{I}(\hat{s}^n(\hat{0})) = n\ (n \geq 0)$.
Abkürzung: $n \hat{+} 1 := \hat{s}(\hat{n})\ (n \geq 0)$
Zahlenterme. $NAT = \{\hat{n} : n \geq 0\}$ Normalformen (Satz 10.2 c gilt).

Wichtige Hilfsfunktionen auf NAT:

Sei $E = \{is\_null(\hat{0}) \to tt, is\_null(\hat{s}(x)) \to ff\}$.
is_null implementiert das Prädikat $Is\_Null : \mathbb{N} \to \{true, false\}$ Nulltest.
Erweitere E mit (nicht terminierende Regeln)
$\hat{g}(x) \to [is\_null(x) \to \hat{0}, \hat{g}(x)], \qquad \hat{f}(x) \to [is\_null(x) \to \hat{g}(x), \hat{0}]$

Beh: Unter der Standardinterpretation $\mathfrak{I}$ gilt:
$\hat{f}$ implementiert die Nullfunktion $f(x) = 0\ \ (x \in \mathbb{N})$ und
$\hat{g}$ implementiert die Funktion $g(0) = 0$ sonst undefiniert.
Wegen $\hat{f}(\hat{0}) \to [is\_null(\hat{0}) \to \hat{g}(\hat{0}), \hat{0}] \xrightarrow{*} \hat{g}(\hat{0}) \to [...] \xrightarrow{*} \hat{0}$ und
$\hat{f}(\hat{s}(x)) \to [is\_null(\hat{s}(x)) \to \hat{g}(\hat{s}(x)), \hat{0}] \xrightarrow{*} \hat{0}$ folgt dies aus Satz 10.4.

Beispiele: Natürliche Zahlen

Erweiterung von E zu E' mit Regel:

$\hat{f}(x, y) = [is\_null(x) \to y, \hat{0}]$ ($\hat{f}$ überladen).

$\hat{f}$ implementiert die Funktion $F : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$$F(x, y) = \begin{cases} y & x = 0 \\ 0 & x \neq 0 \end{cases} \qquad \begin{array}{l} \hat{f}(\hat{0}, \hat{y}) \xrightarrow{*} \hat{y} \\ \hat{f}(\hat{s}(x), \hat{y}) \xrightarrow{*} \hat{0} \end{array}$$

Es gilt aber
$\hat{f}(x, \hat{g}(x)) =_{E'} [is\_null(x) \to \hat{g}(x), \hat{0}]) =_{E'} \hat{f}(x)$

Es ist aber nicht

$f(n) = F(n, g(n))$ für $n > 0$

Will man alle berechenbaren Funktionen implementieren, so lassen sich die Kleeneschen Rekursionsgleichungen nicht direkt Übertragen, da hierbei die Komposition partieller Funktionen benötigt wird.

Darstellung Primitiv Rekursive Funktionen

Die Klasse $\mathfrak{P}$ enthält die Funktionen
$s = \lambda x.x + 1, \pi_i^n = \lambda x_1, ..., x_n.x_i,$ sowie $c = \lambda x.0$ auf $\mathbb{N}$ und
ist abgeschlossen gegen Einsetzung und primitive Rekursion, d.h.

$f(x_1, ..., x_n) = g(h_1(x_1, ..., x_n), ..., h_r(x_1, ..., x_n))$ bzw.

$f(x_1, ..., x_n, 0) = g(x_1, ..., x_n)$
$f(x_1, ..., x_n, y + 1) = h(x_1, ..., x_n, y, f(x_1, ..., x_n, y))$

Behauptung: $f \in \mathfrak{P}$ ist implementierbar durch $(\hat{f}, E_{\hat{f}}, \mathfrak{I})$

Idee: Zeige für geeignetes $E_{\hat{f}}$:

$\hat{f}(\hat{k_1}, ..., \hat{k_n}) \rightarrow^*_{E_{\hat{f}}} f(k_1, \hat{...}, k_n)$ mit $E_{\hat{f}}$ konfluent und terminierend.

Annahme: *FUNKT* (Signatur) enthalte für $n \in \mathbb{N}$ abzählbar viele Funktionssymbole mit Stelligkeit n.

Darstellung Primitiv Rekursive Funktionen

Satz 10.8. *Zu jeder endlichen Menge $A \subset FUNKT \setminus \{\hat{0}, \hat{s}\}$ die Ausnahmemenge, und jeder Funktion $f : \mathbb{N}^n \to \mathbb{N}, f \in \mathfrak{P}$ gibt es $\hat{f} \in FUNKT$ und $E_{\hat{f}}$ endlich konfluent und terminierend mit $(\hat{f}, E_{\hat{f}}, \mathfrak{I})$ implementiert f und keine der Gleichungen in $E_{\hat{f}}$ enthält Funktionssymbole aus A.*

Beweis: Induktion über Aufbau von $\mathfrak{P}$: $\hat{0}, \hat{s} \notin A$. Setze $A' = A \cup \{\hat{0}, \hat{s}\}$

- $\hat{s}$ implementiert s mit $E_{\hat{s}} = \emptyset$
- $\hat{\pi}_i^n \in FUNKT^n \setminus A'$ implem. π_i^n mit $E_{\hat{\pi}_i^n} = \{\hat{\pi}_i^n(x_1, ..., x_n) \to x_i\}$
- $\hat{c} \in FUNKT^1 \setminus A'$ implementiert c mit $E_{\hat{c}} = \{\hat{c}(x) \to \hat{0}\}$
- Einsetzung: $[\hat{g}, E_{\hat{g}}, A_0]$, $[\hat{h}_i, E_{\hat{h}_i}, A_i]$ mit

$A_i = A_{i-1} \cup \{f \in FUNKT : f \in E_{\hat{h}_{i-1}}\} \setminus \{\hat{0}, \hat{s}\}$. Sei $\hat{f} \in FUNKT \setminus A'_r$ und $E_{\hat{f}} = E_{\hat{g}} \cup \bigcup_1^r E_{\hat{h}_i} \cup \{\hat{f}(x_1, ..., x_n) \to \hat{g}(\hat{h}_1(...), ..., \hat{h}_r(...))\}$

- Primitive Rekursion: Analog mit den üblichen Gleichungen.

Darstellung Primitiv Rekursive Funktionen

Alle Regeln sind links-linear ohne Überlappungen $\rightsquigarrow$ Konfluenz.
Terminierungskriterium: Sei $\mathfrak{J}: \ FUNKT \to (\mathbb{N}^* \to \mathbb{N})$, d.h
$\mathfrak{J}(f): \mathbb{N}^{st(f)} \to \mathbb{N}$, streng monoton in allen Argumenten.
Ist E ein Regelsystem, $l \to r \in E, b: VAR \to \mathbb{N}$ (Belegung), gilt
$\mathfrak{J}[b](l) > \mathfrak{J}[b](r)$, so terminiert E.
Idee: Verwende die Ackermannsche Funktion als Schranke:
$A(0,y) = y+1, A(x+1,0) = A(x,1), A(x+1,y+1) = A(x,A(x+1,y))$
A ist streng monoton,
$A(1,x) = x+2, A(x,y+1) \leq A(x+1,y), A(2,x) = 2x+3$
Zu $n \in \mathbb{N}$ gibt es ein β_n mit $\quad \sum_1^n A(x_i,x) \leq A(\beta_n(x_1,...,x_n),x)$

Definiere $\mathfrak{J}$ durch $\mathfrak{J}(\hat{f})(k_1,...,k_n) = A(p_{\hat{f}}, \sum k_i)$ mit geeigneten $p_{\hat{f}} \in \mathbb{N}$.

- $p_{\hat{s}} := 1 :: \mathfrak{J}[b](\hat{s}(x)) = A(1,b(x)) = b(x)+2 > b(x)+1$
- $p_{\hat{\pi}_i^n} := 1 :: \mathfrak{J}[b](\hat{\pi}_i^n(x_1,...,x_n)) = A(1,\sum_1^n b(x_i)) > b(x_i)$
- $p_{\hat{c}} := 1 :: \mathfrak{J}[b](\hat{c}(x)) = A(1,b(x)) > 0 = \mathfrak{J}[b](\hat{0})$

Darstellung Primitiv Rekursive Funktionen

- Einsetzung: $f(x_1, ..., x_n) = g(h_1(...), ..., h_r(...))$.
  Setze $c^* = \beta_r(p_{\hat{h}_1}, ..., p_{\hat{h}_r})$ und $p_{\hat{f}} := p_{\hat{g}} + c^* + 2$. Überprüfe
  $\mathfrak{J}[b](\hat{f}(x_1, ..., x_n)) > \mathfrak{J}[b](\hat{g}(\hat{h}_1(x_1, ..., x_n), ..., \hat{h}_r(x_1, ..., x_n)))$
- Primitive Rekursion:
  Setze $m = max(p_{\hat{g}}, p_{\hat{f}})$ und $p_{\hat{f}} := m + 3$. Überprüfe
  $\mathfrak{J}[b](\hat{f}(x_1, ..., x_n, 0)) > \mathfrak{J}[b](\hat{g}(x_1, ..., x_n))$ und
  $\mathfrak{J}[b](\hat{f}(x_1, ..., x_n, \hat{s}(y))) > \mathfrak{J}[b](\hat{g}(....))$.
  Verwende $A(m + 3, k + 3) > A(p_{\hat{h}}, k + A(p_{\hat{f}}, k))$
- Durch Induktion über Aufbau von $\mathfrak{P}$ zeige
  $\hat{f}(\hat{k}_1, ..., \hat{k}_n) \rightarrow^*_{E_{\hat{f}}} f(k_1, \hat{...}, k_n)$
- Aus Satz 10.4 folgt die Behauptung.

Darstellung Rekursiver Funktionen

Minimierung:: μ-Operator $\mu_y[g(x_1, ..., x_n, y) = 0] = z$ gdw
i) $g(x_1, ..., x_n, i)$ definiert $\neq 0$ für $0 \leq i < z$ ii) $g(x_1, ..., x_n, z) = 0$

Reguläre Minimierung: μ wird auf totale Funktionen mit
$\forall x_1, ..., x_n \exists y : g(x_1, ..., x_n, y) = 0$

$\mathfrak{R}$ abgeschlossen gegen Ersetzung, Primitive Rekursion und Reguläre Minimierung.

Zeige: Reguläre Minimierung Implementierbar mit Ausnahmemenge A.
$\hat{g}, E_{\hat{g}}$ implementiere g wobei $\hat{g}(\hat{k}_1, ..., \hat{k}_{n+1}) \rightarrow^*_{E_{\hat{g}}} g(k_1, ..., k_{n+1})$
Seien $\hat{f}, \hat{f}^+, \hat{f}^*$ neu setze $E_{\hat{f}} := E_{\hat{g}} \cup \{\hat{f}(x_1, ..., x_n) \rightarrow \hat{f}^*(x_1, ..., x_n, \hat{0}),$
$\hat{f}^*(x_1, ..., x_n, y) \rightarrow \hat{f}^+(\hat{g}(x_1, ..., x_n, y), x_1, ..., x_n, y),$
$\hat{f}^+(\hat{0}, x_1, ..., x_n, y) \rightarrow y, \hat{f}^+(\hat{s}(x), x_1, ..., x_n, y) \rightarrow, \hat{f}^*(x_1, ..., x_n, \hat{s}(y))\}$

Behauptung: $(\hat{f}, E_{\hat{f}})$ implementiert die Minimierung von g.

Darstellung Rekursiver Funktionen

Voraussetzung: Zu $k_1, ..., k_n \in \mathbb{N}$ gibt es kleinstes $k \in \mathbb{N}$ mit $g(k_1, ..., k_n, k) = 0$

Behauptung: Für alle $i \in \mathbb{N}, i \leq k$ gilt $\hat{f}^*(\hat{k}_1, ..., \hat{k}_n, (k \hat{-} i)) \rightarrow^*_{E_{\hat{f}}} \hat{k}$

Bew: Induktion nach i:

- $i = 0 :: \hat{f}^*(\hat{k}_1, ..., \hat{k}_n, \hat{k}) \rightarrow \hat{f}^+(\hat{g}(\hat{k}_1, ..., \hat{k}_n, \hat{k}), \hat{k}_1, ..., \hat{k}_n, \hat{k}) \rightarrow^*_{E_{\hat{g}}}$
  $\hat{f}^+(g(k_1, ..., k_n, k), \hat{k}_1, ..., \hat{k}_n, \hat{k}) \rightarrow \hat{k}$
- $i > 0 :: \hat{f}^*(\hat{k}_1, ..., \hat{k}_n, k - (\hat{i} + 1)) \rightarrow$
  $\hat{f}^+(\hat{g}(\hat{k}_1, ..., \hat{k}_n, k - (\hat{i} + 1)), \hat{k}_1, ..., \hat{k}_n, k - (\hat{i} + 1)) \rightarrow^*_{E_{\hat{g}}}$
  $\hat{f}^+(\hat{s}(\hat{x}), \hat{k}_1, ..., \hat{k}_n, k - (\hat{i} + 1)) \rightarrow \hat{f}^*(\hat{k}_1, ..., \hat{k}_n, \hat{s}(k - (\hat{i} + 1))) =$
  $\hat{f}^*(\hat{k}_1, ..., \hat{k}_n, k \hat{-} i)) \rightarrow^*_{E_{\hat{g}}} \hat{k}$
  Für geeignetes x und Ind. Voraussetzung.
- $E_{\hat{f}}$ ist konfluent und nach Satz 10.4 implementiert $(\hat{f}, E_{\hat{f}})$ die totale Funktion f.
- $E_{\hat{f}}$ ist nicht terminierend.$g(k, m) = \delta_{k,m} \rightsquigarrow \hat{f}^*(\hat{k}, k \hat{+} 1)$ liefert NT-Kette. Lässt sich erreichen.

Darstellung Partiell Rekursiver Funktionen

Problem: Rekursionsgleichungen (Kleenesche Normalform) lassen sich nicht direkt verwenden. Argumente müssen "Zahl" als Wert besitzen. (Siehe Beispiel). Manche Argumente lassen sich noch retten:

Beispiel 10.9. $f(x,y) = g(h_1(x,y), h_2(x,y), h_3(x,y))$. g, h_1, h_2, h_3 *seien durch Gleichungsmengen als partielle Funktionen implementierbar.*

Behauptung: f ist implementierbar. Seien dazu $\hat{f}, \hat{f}_1, \hat{f}_2$ neu und setze:

$$\hat{f}(x,y) = \hat{f}_1(\hat{h}_1(x,y), \hat{h}_2(x,y), \hat{h}_3(x,y), \hat{f}_2(\hat{h}_1(x,y)), \hat{f}_2(\hat{h}_2(x,y)), \hat{f}_2(\hat{h}_3(x,y)))$$

$$\hat{f}_1(x_1,x_2,x_3,\hat{0},\hat{0},\hat{0}) = \hat{g}(x_1,x_2,x_3), \quad \hat{f}_2(\hat{0}) = \hat{0}, \quad \hat{f}_2(\hat{s}(x)) = \hat{f}_2(x)$$

$(\hat{f}, E_{\hat{g}}, E_{\hat{h}_1}, E_{\hat{h}_2}, E_{\hat{h}_3} \cup \mathit{REST})$ *implementiert f.*

Satz 10.4 kann nicht angewendet werden.

$(\hat{f}, E_{\hat{g}}, E_{\hat{h}_1}, E_{\hat{h}_2}, E_{\hat{h}_3} \cup \ REST)$ implementiert f.

Wende Definition 10.1 an:
$\curvearrowright$ Für Zahlenterme sei $f(\mathfrak{I}(t_1), \mathfrak{I}(t_2)) = \mathfrak{I}(t)$. Es gibt Zahlenterme T_i $(i = 1, 2, 3)$ mit
$g(\mathfrak{I}(T_1), \mathfrak{I}(T_2), \mathfrak{I}(T_3)) = \mathfrak{I}(t)$ und $h_i(\mathfrak{I}(t_1), \mathfrak{I}(t_2)) = \mathfrak{I}(T_i)$.
Voraussetzung: $\hat{g}(T_1, T_2, T_3) =_{E_{\hat{f}}} t$ und $\hat{h}_i(t_1, t_2) =_{E_{\hat{f}}} T_i (i = 1, 2, 3)$ Da die T_i Zahlenterme: $\hat{f}_2(T_i) =_{E_{\hat{f}}} \hat{0}$ d.h. $\hat{f}_2(\hat{h}_i(t_1, t_2)) =_{E_{\hat{f}}} \hat{0}$ $(i = 1, 2, 3)$.
Also
$\hat{f}(t_1, t_2) =_{E_{\hat{f}}} \hat{f}_1(T_1, T_2, T_3, \hat{0}, \hat{0}, \hat{0}) \rightsquigarrow \hat{f}(t_1, t_2) =_{E_{\hat{f}}} t (=_{E_{\hat{f}}} \hat{g}(T_1, T_2, T_3)$
$\curvearrowleft$ Für Zahlenterme t_1, t_2, t gelte $\hat{f}(t_1, t_2) =_{E_{\hat{f}}} t$, dann
$\hat{f}_1(\hat{h}_1(t_1, t_2), \hat{h}_2(t_1, t_2), \hat{h}_3(t_1, t_2), \hat{f}_2(\hat{h}_1(t_1, t_2), ...) =_{E_{\hat{f}}} t$. Wäre für ein $i = 1, 2, 3$ $\hat{f}_2(\hat{h}_i(t_1, t_2))$ nicht $E_{\hat{f}}$ gleich $\hat{0}$, so sind in der $E_{\hat{f}}$ Äquivalenzklasse nur $\hat{f}_1$ Terme. Es git also Zahlenterme T_1, T_2, T_3 mit $\hat{h}_i(t_1, t_2) =_{E_{\hat{f}}} = T_i$ $(i = 1, 2, 3)$ (Sonst nur $\hat{f}_2$ Terme äquivalent zu $\hat{f}_2(\hat{h}_i(t_1, t_2))$. Voraussetzung:
$\rightsquigarrow h_i(\mathfrak{I}(T_1), \mathfrak{I}(T_2)) = \mathfrak{I}(T_i), \qquad g(\mathfrak{I}(T_1), \mathfrak{I}(T_2), \mathfrak{I}(T_3)) = \mathfrak{I}(t)$

$\mathfrak{R}_p$ und Normierte Register Maschinen

Definition 10.10. *Programmterme für RM:* P_n $(n \in \mathbb{N})$ *Sei* $0 \leq i \leq n$
Funktionssymbole: a_i, s_i *Konstanten* $, \circ$ *2-stellig* $, W^i$ *1-stellig*
Gewünschte Interpretation:
a_i :: *Inhalt Register i um 1 Erhöhen.*
s_i :: *Inhalt Register i um 1 Erniedrigen* $(\dot{-}1)$
$\circ(M_1, M_2)$ :: *Verkettung* $M_1 M_2$ *(Erst* M_1*, dann* M_2*)*
$W^i(M)$ :: *Solange Inhalt von Register i nicht 0, führe M aus Abk:* $(M)_i$

Beachte: $P_n \subseteq P_m$ für $n \leq m$

Semantik durch partielle Funktionen: $M_e : P_n \times \mathbb{N}^n \longrightarrow \mathbb{N}^n$

- $M_e(a_i, \langle x_1, ..., x_n \rangle) = \langle ...x_{i-1}, x_i + 1, x_{i+1}... \rangle$ $(s_i :: x_i \dot{-} 1)$
- $M_e(M_1 M_2, \langle x_1, ..., x_n \rangle) = M_e(M_2, M_e(M_1, \langle x_1, ..., x_n \rangle))$
- $M_e((M)_i, \langle x_1, ..., x_n \rangle) = \begin{cases} \langle x_1, ..., x_n \rangle & x_i = 0 \\ M_e((M)_i, M_e(M, \langle x_1, ..., x_n \rangle)) & \text{sonst} \end{cases}$

Implementierung normierter Register Maschinen

Lemma 10.11. *M_e kann durch Gleichungssystem implementiert werden.*

Beweis: Sei tup_n n-stelliges Funktionssymbol. Für $t_i \in \mathbb{N}$ $(0 < i \leq n)$ sei $\langle t_1, ..., t_n \rangle$ Interpretation von $tup_n(\hat{t}_1, ..., \hat{t}_n)$. Programmterme werden durch sich selbst interpretiert (es sind ja Terme). Für $m \geq n$ ::

$$\begin{array}{ll} P_n & tup_m(\hat{t}_1, ..., \hat{t}_m) \quad \text{Syntaktische Ebene} \\ \mathfrak{I} \downarrow & \mathfrak{I} \downarrow \\ P_n & \langle t_1, ..., t_m \rangle \quad \text{Interpretation} \end{array}$$

Sei *eval* 2-stelliges Funktionssymbol zur Implementierung von M_e und $i \leq n$. Definiere $E_n := \{$

$$eval(a_i,\ tup_n(x_1, ..., x_n)) \to tup_n(x_1, ..., x_{i-1}, \hat{s}(x_i), x_{i+1}, ..., x_n)$$
$$eval(s_i,\ tup_n(..., x_{i-1}, \hat{0}, x_{i+1}...)) \to tup_n(..., x_{i-1}, \hat{0}, x_{i+1}...)$$
$$eval(s_i,\ tup_n(..., x_{i-1}, \hat{s}(x), x_{i+1}...)) \to tup_n(..., x_{i-1}, x, x_{i+1}...)$$
$$eval(x_1x_2, t) \to eval(x_2,\ eval(x_1, t))$$
$$eval((x)_i,\ tup_n(..., x_{i-1}, \hat{0}, x_{i+1}...)) \to tup_n(..., x_{i-1}, \hat{0}, x_{i+1}...)$$
$$eval((x)_i,\ tup_n(..., x_{i-1}, \hat{s}(y), x_{i+1}...)) \to$$
$$eval((x)_i, eval(x,\ tup_n(..., x_{i-1}, \hat{s}(y), x_{i+1}...)))\}$$

$(eval, E_n, \mathfrak{I})$ implementiert M_e

Betrachte Programmterme die höchstens Register mit $1 \leq i \leq n$ enthalten.

- E_n ist konfluent (links-linear, ohne kritische Paare).
- Satz 10.4 nicht anwendbar, da M_e nicht total. Bedingungen der Definition 10.1 überprüfen.

(1) $\mathfrak{I}(T_i) = M_i$ nach Definition.
(2) $M_e(p, \langle k_1, ..., k_n \rangle) = \langle m_1, ..., m_n \rangle$ gdw
$eval(p,\ tup_n(\hat{k}_1, ..., \hat{k}_n)) =_{E_n} tup_n(\hat{m}_1, ..., \hat{m}_n)$
$\curvearrowright$ Aus der Def. von M_e bzw. E_n. Induktion Aufbau von p.
$\curvearrowleft$ Induktion über Aufbau von p ::
1. $p = a_i(s_i) :: \hat{k}_j = \hat{m}_j (j \neq i), \hat{s}(\hat{k}_i) = \hat{m}_i$ bzw. $\hat{k}_i = \hat{m}_i = \hat{0}$
$(\hat{k}_i = \hat{s}(\hat{m}_i))$ für s_i
2.Sei $p = p_1 p_2$ und
$eval(p_2, eval(p_1,\ tup_n(\hat{k}_1, ..., \hat{k}_n))) \xrightarrow{*}_{E_n} tup_n(\hat{m}_1, ..., \hat{m}_n)$
Wegen der Regeln in E_n gilt:

$(eval, E_n, \mathfrak{I})$ implementiert M_e

Es gibt $i_1, ..., i_n \in \mathbb{N}$ mit $eval(p_1, tup_n(\hat{k}_1, ..., \hat{k}_n)) \xrightarrow{*}_{E_n} tup_n(\hat{i}_1, ..., \hat{i}_n)$
also auch
$eval(p_2, tup_n(\hat{i}_1, ..., \hat{i}_n)) \xrightarrow{*}_{E_n} tup_n(\hat{m}_1, ..., \hat{m}_n)$
Nach Induktion Voraussetzung (2-Mal) gilt die Behauptung.
3. Sei $p = (p_1)_i$. Es gilt:
$eval((p_1)_i, tup_n(\hat{k}_1, ..., \hat{k}_n)) \xrightarrow{*}_{E_n} tup_n(\hat{m}_1, ..., \hat{m}_n)$
Es gibt endliche Folge $(t_j)_{1 \le j \le l}$ mit
$t_1 = eval((p_1)_i, tup_n(\hat{k}_1, ..., \hat{k}_n)), \quad t_j \to t_{j+1}, \quad t_l = tup_n(\hat{m}_1, ..., \hat{m}_n)$
Es gibt eine Teilfolge $(T_j)_{1 \le j \le m}$ der Gestalt $eval((p_1)_i, tup_n(\hat{i}_{1,j}, ..., \hat{i}_{n,j}))$
In T_m ist $i_{i,m} = 0$, d.h. $i_{1,m} = m_1, ..., i_{i,m} = 0 = m_i, ..., i_{n,m} = m_n$.
Für $j < m$ gilt stets $i_{i,j} \neq 0$ und
$eval(p_1, tup_n(\hat{i}_{1,j}, ..., \hat{i}_{n,j}) \xrightarrow{*}_{E_n} tup_n(\hat{i}_{1,j+1}, ..., \hat{i}_{n,j+1})$. Induktion
Voraussetzung liefert:
$M_e(p_1, \langle i_{1,j}, ..., i_{n,j} \rangle) = \langle i_{1,j+1}, ..., i_{n,j+1} \rangle$ für $j = 1, ..., m$. Dann aber
$M_e((p_1)_i, \langle i_{1,j}, ..., i_{n,j} \rangle) = \langle m_1, ..., m_n \rangle \quad (1 \le j < m)$

Implementierung von $\mathfrak{R}_p$

Für $f \in \mathfrak{R}_p^{n,1}$ gibt es $r \in \mathbb{N}$, Programmterm p mit höchstens r-Registern $(n+1 \leq r)$, so dass für alle $k_1, ..., k_n, k \in \mathbb{N}$ gilt:
$f(k_1, ..., k_n) = k$ gdw $\forall m \geq 0$

$$eval(p, tup_{r+m}(\hat{k}_1, ..., \hat{k}_n, \hat{0}, \hat{0}, ..., \hat{0}, \hat{x}_1, ..., \hat{x}_m)) =_{E_{r+m}} tup_{r+m}(\hat{k}_1, ..., \hat{k}_n, \hat{k}, \hat{0}, ..., \hat{0}, \hat{x}_1, ..., \hat{x}_m) \quad \text{gdw}$$

$$eval(p, tup_r(\hat{k}_1, ..., \hat{k}_n, \hat{0}, \hat{0}, ..., \hat{0})) =_{E_r} tup_r(\hat{k}_1, ..., \hat{k}_n, \hat{k}, \hat{0}, ..., \hat{0})$$

Beachte: $E_r \sqsubset E_{r+m}$ via $tup_r(...) \blacktriangleright tup_{r+m}(..., \hat{0}, ..., \hat{0})$.

Seien $\hat{f}, \hat{R}$ neue Funktionssymbole, p Programm für f. Erweitere E_r um $\hat{f}(y_1, ..., y_n) \to \hat{R}(eval(p, tup_r(y_1, ..., y_n), \hat{0}, ..., \hat{0}))$ und $\hat{R}(tup_r(y_1, ..., y_r)) = y_{n+1}$ zu $E_{ext(f)}$.

Satz 10.12. *$f \in \mathfrak{R}_p^{n,1}$ wird von $(\hat{f}, E_{ext(f)}, \mathfrak{I})$ implementiert.*

Nicht berechenbare Funktionen

Sei E rekursiv, T_i rekursiv. Dann ist das Prädikat

$$P(t_1, ..., t_n, t_{n+1}) \text{ gdw } \hat{f}(t_1, ..., t_n) =_E t_{n+1}$$

r.a. Prädikat auf $T_1 \times ... \times T_n \times T_{n+1}$
Implementiert $\hat{f}$ die Funktion f, so stellt P den Graphen der Funktion f dar $\rightsquigarrow$ $f \in \mathfrak{R}_p$.
Kleenescher Normalformsatz: $f(x_1, ..., x_n) = U(\mu_y[T_n(p, x_1, ..., x_n, y) = 0])$

Sei h totale nicht rekursive Funktion definiert durch:

$$h(x) = \begin{cases} \mu_y[T_1(x, x, y) = 0] & \text{falls } \exists y : T_1(x, x, y) = 0 \\ 0 & \text{sonst} \end{cases}$$

h wird eindeutig durch folgende Prädikate festgelegt:
(1) $(T_1(x, x, y) = 0 \wedge \forall z(z < y \rightsquigarrow T_1(x, x, z) \neq 0)) \rightsquigarrow h(x) = y$
(2) $(\forall z(z < y \wedge T_1(x, x, z) \neq 0)) \rightsquigarrow (h(x) = 0 \vee h(x) \geq y)$
Ersetzt man $h(x)$ durch u, so sind dies prim. rek. Prädikate in x, y, u.

Nicht berechenbare Funktionen

Es gibt primitiv rekursive Funktionen P_1, P_2 in x, y, u, so dass

$$(1') \quad P_1(x, y, h(x)) = 0 \text{ und } (2') \quad P_2(x, y, h(x)) = 0$$

(1) und (2) darstellen.
Es gibt Gleichungssystem E und Funktionssymbole $\hat{P}_1, \hat{P}_2$ die P_1, P_2 unter der standard Interpretation implementieren.
(Als prim. rek. Funktionen in den Var. x, y, u)
Sei $\hat{h}$ frisch. Füge zu E die Gleichungen

$$\hat{P}_1(x, y, \hat{h}(x)) = \hat{0} \quad \text{und} \quad \hat{P}_2(x, y, \hat{h}(x)) = \hat{0}$$

hinzu. Das Gleichungssystem ist Konsistent (es gibt Modelle) und $\hat{h}$ wird durch die Funktion h auf den natürlichen Zahlen interpretiert.$\rightsquigarrow$
Man kann also mit einer endlichen Gleichungsmenge implizit nicht rekursive Funktionen spezifizieren, wenn man beliebige Modelle als Interpretationen zulässt.
Durch nicht rekursive Gleichungsmengen kann man beliebige Funktionen als Interpretation eines konfluenten, terminierendes Grundsystem erhalten:
$E = \{\hat{h}(\hat{t}) = \hat{t}' : t, t' \in \mathbb{N}, h(t) = t'\}$ (Regelanwendung nicht effektiv).

Berechenbare Algebren

Definition 10.13.

- *Eine sig-Algebra $\mathfrak{A}$ ist rekursiv (effektiv, berechenbar), falls ihre Trägermengen rekursiv und alle Operationen rekursive Funktionen sind.*
- *Eine Spezifikation $spec = (sig, E)$ ist rekursiv, falls T_{spec} rekursiv ist.*

Beispiel 10.14. *Sei $sig = (\{nat, gerade\}, odd :\rightarrow gerade, 0 :\rightarrow nat, s : nat \rightarrow nat, red : nat \rightarrow gerade)$.*
Als sig-Algebra $\mathfrak{A}$ wähle: $A_{gerade} = \{2n : n \in \mathbb{N}\} \cup \{1\}, A_{nat} = \mathbb{N}$ mit

$$red \text{ als } \lambda x.\mathit{if}\ x \text{ gerade } \mathit{then}\ x\ \mathit{else}\ 1, \qquad s \text{ Nachfolger}$$

Behauptung: Es gibt keine endliche (init-Algebra) Spezifikation für $\mathfrak{A}$

- *Keine Gleichungen der Sorte nat.*
- *$odd, red(s^n(0)), red(s^n(x))$ $(n \geq 0)$ Terme der Sorte gerade. Keine Gleichungen der Form $red(s^n(x)) = red(s^m(x)$ $(n \neq m)$ möglich.*
- *Unendlich viele Grundgleichungen nötig.*

Berechenbare Algebren

Lösung: Anreicherung der Signatur mit:
$gerade : nat \rightarrow nat$ und $cond : nat\ gerade\ gerade \rightarrow gerade$ mit Interpretation

$$\lambda x.\ if\ x\ gerade\ then\ 0\ else\ 1, \qquad \lambda x, y, z.\ if\ x = 0\ then\ y\ else\ z$$

Gleichungen:
$gerade(0) = 0, gerade(s(0)) = s(0), gerade(s(s(x)) = gerade(x)$
$cond(0, y, z) = y, cond(s(x), y, z) = z$
$red(x) = cond(gerade(x), red(x), odd)$

Alternative: Bedingte Gleichungen:
$red(s(0)) = odd,\ red(s(s(x)) = odd\ if\ red(x) = odd$

Bedingte Gleichungssysteme (Termersetzungssysteme) sind offenbar Ausdruckstärker als reine Gleichungssysteme. Sie definieren ebenfalls Reduktionsrelationen. Konfluenz- und Terminierungskriterien lassen sich angeben. Negative Gleichungen in den Bedingungen führen zu Problemen mit der Initialen Semantik (keine Horn-Klausel Spezifikationen).

Berechenbare Algebren: Ergebnisse

Satz 10.15. *Sei $\mathfrak{A}$ eine rekursive termerzeugte sig- Algebra. Dann gibt es eine endliche Anreicherung sig' von sig und eine endliche Spezifikation $spec' = (sig', E)$ mit $T_{spec'}|_{sig} \cong \mathfrak{A}$.*

Satz 10.16. *Sei $\mathfrak{A}$ eine termerzeugte sig- Algebra. Dann sind äquivalent:*

- *$\mathfrak{A}$ ist rekursiv.*
- *Es gibt eine endliche Anreicherung (ohne neue Sorten) sig' von sig und ein endliches konvergentes Regelsystem R, so dass $\mathfrak{A} \cong T_{spec'}|_{sig}$ für $spec' = (sig', R)$*

Siehe Bergstra, Tucker: Characterization of Computable Data Types (Math. Center Amsterdam 79).

Achtung: Gilt nicht wenn man sich nur auf einstellige Funktionssymbole einschränkt.

Reduktionsstrategien für Ersetzungssysteme

Grundlegende Implementierungsprobleme für funktionale Programmiersprachen.
Welche Reduktionsstrategien garantieren die Berechnung von Normalformen falls diese existieren. Sei R TES, $t \in Term(\Sigma)$.
Ang. es gibt $\bar{t}$ irreduzibel mit $t \rightarrow_R^* \bar{t}$.

- Welche Auswahl der Redexe garantiert eine "Berechnung" von $\bar{t}$.
- Welche Auswahl der Redexe liefern "kürzeste" Ableitungsketten.
- Sei R terminierend. Gibt es eine Reduktionsstrategie die stets die kürzesten Ableitungsketten liefert. Was kostet sie?

Für $SKI-$Kalkül und $\lambda-$Kalkül ist die Left-Most-Outermost Strategie (normale Strategie) normalisierend, d.h. sie berechnet eine Normalform eines Terms wenn sie existiert. Sie liefert nicht die kürzesten Ableitungsketten. Es gilt jedoch: Ist $t \stackrel{k}{\rightarrow} \bar{t}$ eine kürzeste Ableitungskette, so $t \rightarrow_{LMOM}^{\leq 2^k} \bar{t}$. Durch Structure-Sharing-Methoden kann die Schranke für LMOM kleiner gewählt werden.

Funktionale Berechnungsmodelle

- Partiell rekursive Funktionen (Ausgangsfunktionen + Operatoren)
- Termersetzungssysteme (Algebraische Spezifikation)
- λ-Kalkül und Kombinatorenkalkül
- Graphersetzungssysteme (Implementierung + Effizienz)

Zentraler Begriff: Applikation:
Ausdrücke stellen Funktionen dar, Applikation (Anwendung) von Funktionen auf Funktionen $\rightsquigarrow$ Selbstanwendungsproblem

Siehe etwa Barendregt: Functional Programming and λ-Calculus Handbook of Theoretical Computer Science.

λ-Kalkül und Kombinatorenkalkül: Informell

Grundoperationen:

- Applikation:: $F.A$ auch (FA)
  F als Programmterm wird auf A als Argumentterm angewendet.
- Abstraktion:: $\lambda x.M$
  Bezeichnet eine Funktion die x nach M abbildet, hierbei kann M von x abhängen.
- Beispiel: $(\lambda x.2 * x + 1).3$ sollte als Ergebnis $2 * 3 + 1$ also 7 liefern.
- β-Regel:: $(\lambda x.M[x])N = M[x := N]$
  "Freie" Vorkommen von x in M durch N ersetzen. β-Konversion

$$(yx(\lambda x.x))[x := N] \equiv (yN(\lambda x.x))$$

Hierbei bleiben freie Vorkommen von Variablen in N frei (u.u. durch Umbenennung)

$$(\lambda x.y)[y := xx] \equiv \lambda z.xx\ z \text{ "neu"}$$

λ-Kalkül und Kombinatorenkalkül: Informell

- α-Regel:: $\lambda x.M = \lambda y.M[x := y]$ mit y "neu"
  $\lambda x.x = \lambda y.y$. Gleiche Wirkung als "Funktionen" α-Konversion
- Menge der λ- Terme in C und V::
  $$\Lambda(C, V) = C|V|(\Lambda\Lambda)|(\lambda V.\Lambda)$$
- Die Menge der freien Variablen von M:: $FV(M)$
- M heisst abgeschlossen (Kombinator) falls $FV(M) = \emptyset$
- Standard Kombinatoren:: $I \equiv \lambda x.x$ $K \equiv \lambda xy.x$
  $B \equiv \lambda xyz.x(yz)$ $K_* \equiv \lambda xy.y$ $S \equiv \lambda xyz.xz(yz)$
- Folgende Gleichungen gelten:
  $IM = M$ $KMN = M$ $K_*MN = N$ $SMNL = ML(NL)$
  $BLMN = L(M(N))$
- Fixpunktsatz:: $\forall F \exists X$ $FX = X$ mit $X \equiv WW$ und
  $W \equiv \lambda x.F(xx)$

λ-Kalkül und Kombinatorenkalkül: Informell

- Darstellbarkeit von Funktionen, Zahlen $c_n \equiv \lambda fx.f^n(x)$
  F Kombinator stellt f dar gdw. $Fz_{n1}...z_{nk} = z_{f(n1,...,nk)}$
- f ist partiell rekursiv gdw. f wird von Kombinator dargestellt.
- Satz von Scott: Sei $A \subset \Lambda$, A nicht trivial und abgeschlossen unter $=$, dann ist A nicht rekursiv entscheidbar.
- β-Reduktion:: $(\lambda x.M)N \rightarrow_\beta M[x := N]$
- $NF =$ Menge der Terme mit einer Normalform ist nicht rekursiv.
- $(\lambda x.xx)y$ nicht in Normalform, yy ist Normalform
- $(\lambda x.xx)(\lambda x.xx)$ hat keine Normalform.
- Church Rosser Satz:: $\rightarrow_\beta$ ist konfluent
- Satz von Curry Hat M eine Normalform so gilt $M \rightarrow_l^* N$, d.h. Leftmost Reduktion ist Normalisierend.

Reduktionsstrategien für Ersetzungssysteme

Definition 11.1. *Sei R ein TES.*

- *Eine Einschritt-Reduktionsstrategie $\mathfrak{S}$ für R ist eine Abbildung $\mathfrak{S} : Term(R, V) \to Term(R, V)$ mit $t = \mathfrak{S}(t)$ falls t in Normalform und $t \to_R \mathfrak{S}(t)$ sonst.*
- *$\mathfrak{S}$ ist eine Mehrschritt-Reduktionsstrategie für R wenn $t = \mathfrak{S}(t)$ falls t in Normalform und $t \overset{+}{\to}_R \mathfrak{S}(t)$ sonst.*
- *Eine Reduktionsstrategie $\mathfrak{S}$ heißt normalisierend für R, falls für jeden Term t der eine Normalform hat die Folge $(\mathfrak{S}^n(t))_{n\geq 0}$ eine Normalform enthält. (Insbesondere endlich ist).*
- *Eine Reduktionsstrategie $\mathfrak{S}$ heißt cofinal für R, falls für jedes t und $r \in \Delta^*(t)$ es ein $n \in \mathbb{N}$ gibt mit $r \overset{*}{\to}_R \mathfrak{S}^n(t)$.*

Cofinale Reduktionsstrategien sind die Besten im folgenden Sinn: Sie liefern maximalen Informationsgewinn.
Normalformen haben stets maximale Information.

Bekannte Reduktionsstrategien

Definition 11.2. *Reduktionsstrategien:*

- *Leftmost-Innermost (Call-by-Value). Einschritt-RS, reduziert wird Redex der am weitesten links im Term vorkommt und keinen echten Redex enthält.*
- *Paralell-Innermost. Mehrschritt-RS.* $PI(t) = \overline{t}$, *wobei* $t \mapsto \overline{t}$ *(Alle innermost Redexe werden reduziert).*
- *Leftmost-Outermost (Call-by-Name). Einschritt-RS.*
- *Parallel-Outermost. Mehrschritt-RS.* $PO(t) = \overline{t}$, *wobei* $t \mapsto \overline{t}$ *(Alle disjunkte outermost Redexe werden reduziert).*
- *Fair-LMOM. Ein Left-Most Outermost Redex in einer Red-Folge wird irgendwann reduziert. (Ein LMOR bleibt bei einer solchen Strategie nicht unreduziert). (Lazy Strategie).*

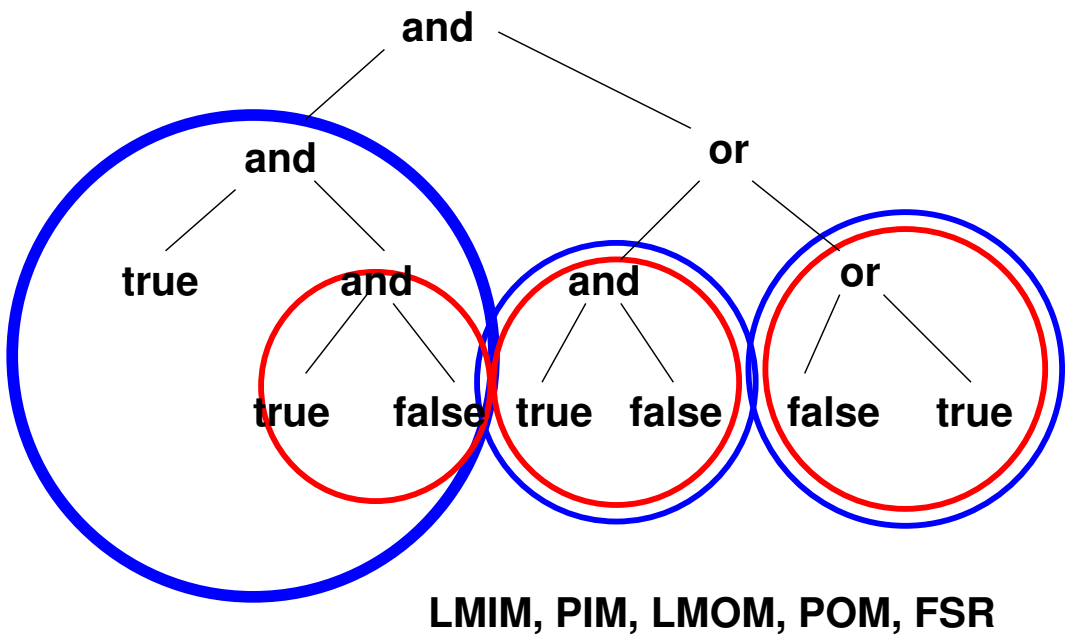
Bekannte Reduktionsstrategien

- Full-Substitution-rule. (Nur für orthogonale Systeme).
  Mehrschritt-RS. $GK(t) :: t \xrightarrow{+} GK(t)$ alle Redexe in t werden reduziert, falls nicht Disjunkt, so werden die Reste der Redexe danach noch reduziert.
- Call-By-Need. Einschritt-RS. Es wird stets ein notwendiger Redex reduziert. Ein Redex in t ist notwendig, wenn er zur Berechnung der Normalform reduziert werden muss. (Nur für bestimmte TES z.B. LMOM für SKI Kalkül)
  Problem: Wie entscheidet man ob ein Redex notwendig ist.
- Variable-Delay-Strategy: Einschritt-RS. Reduziere Redex der nicht als Redex in Variableninstanz eines anderen Redexes vorkommt.

Beispiele

Beispiel 11.3. *:*

- $and(true, x) \to x$, $and(false, x) \to false$,
  $or(true, x) \to true$, $or(false, x) \to x$
  Orthogonal, stark links sequentiell (Konstanten vor Variablen).



Beispiele

- $\Sigma = \{0, s, p, if0, F\}, R = \{p(0) \to 0, p(s(x)) \to x, if0(0, x, y) \to x, if0(s(z), x, y) \to y, F(x, y) \to if0(x, 0, F(p(x), F(x, y)))\}$
  Links -linear, ohne Überlappungen.(Orthogonal).
  $F(0,0) \to if0(0, 0, F(p(0), F(0,0))) \overset{OM}{\to} 0$
  $\downarrow PIM$
  $if0(0, 0, F(0, if0(0, 0, F(p(0), F(0,0)))))$
  Keine IM-Strategie ist für beliebige orthogonale Systeme normalisierend, erst recht nicht cofinal.
- FSR (Full-Substitution-Rule): Wähle alle Redexe im Term und reduziere sie von innermost zu outermost (beachte Redexe werden nicht zerstört). Cofinal für orthogonale Systeme.
- $\Sigma = \{a, b, c, d_i : i \in \mathbb{N}\}$
  $R := \{a \to b, d_k(x) \to d_{k+1}(x), c(d_k(b)) \to b$
  konfluent (links linear parallel 0-abgeschlossen).
  $c(d_0(a)) \to_1 c(d_1(a)) \to_1 \ldots$. nicht normalisierend (POM).
  $c(d_0(a)) \to_{1,1} c(d_0(b)) \to_0 b$

Beispiele

- $\Sigma = \{a, b_i, c, d : i \in \mathbb{N}\}$. Nicht konfluentes WES:
  $R = \{ab_0c \to acb_0, ab_0d \to ad, c \to d, cb_i \to d, b_i \to b_{i+1}(i \geq 1)\}$
  $ab_0c \to_{11} ab_0d \to ad$
  $ab_0c \to_0 acb_0 \to_{11} acb_1 \to adb_1 \to \ldots$
- $\Sigma = \{f, a, b, c, d\}$ $R = \{f(x, b) \to d, a \to b, c \to c\}$ Orthogonal.
  LMOM muss nicht normalisierend sein:
  $f(c, a) \to f(c, a) \to \ldots.$ aber $f(c, a) \to f(c, b) \to d$
- $f(a, f(x, y)) \to f(x, f(x, f(b, b)))$ links linear mit Überlappungen.
  $f(a, f(a, f(b, b))) \to_{OUT} f(a, f(a, f(b, b))) \to_{OUT} \ldots.$
  $\downarrow INN$
  $f(a, f(b, f(b, f(b, b)))) \to f(b, f(b, f(b, b)))$
- $R = \{f(g(x), c) \to h(x, d), b \to c\}$
  $f(g(f(a, f(a, \underline{b}))), c) \to_{VD} h(f(a, f(a, \underline{b})), d) \to_{VD}$
  $h(f(a, f(a, c)), d)$

Strategien für orthogonale Systeme

Satz 11.4. *Für orthogonale Systeme gilt:*

- *Full-Substitution-Rule ist eine cofinale Reduktionstrategie.*
- *POM ist eine normalisierende Reduktionstrategie.*
- *LMOM ist normalisierend für λ-Kalkül und CL-Kalkül.*
- *Jede Strategie die Fair-Outermost ist, ist normalisierend.*

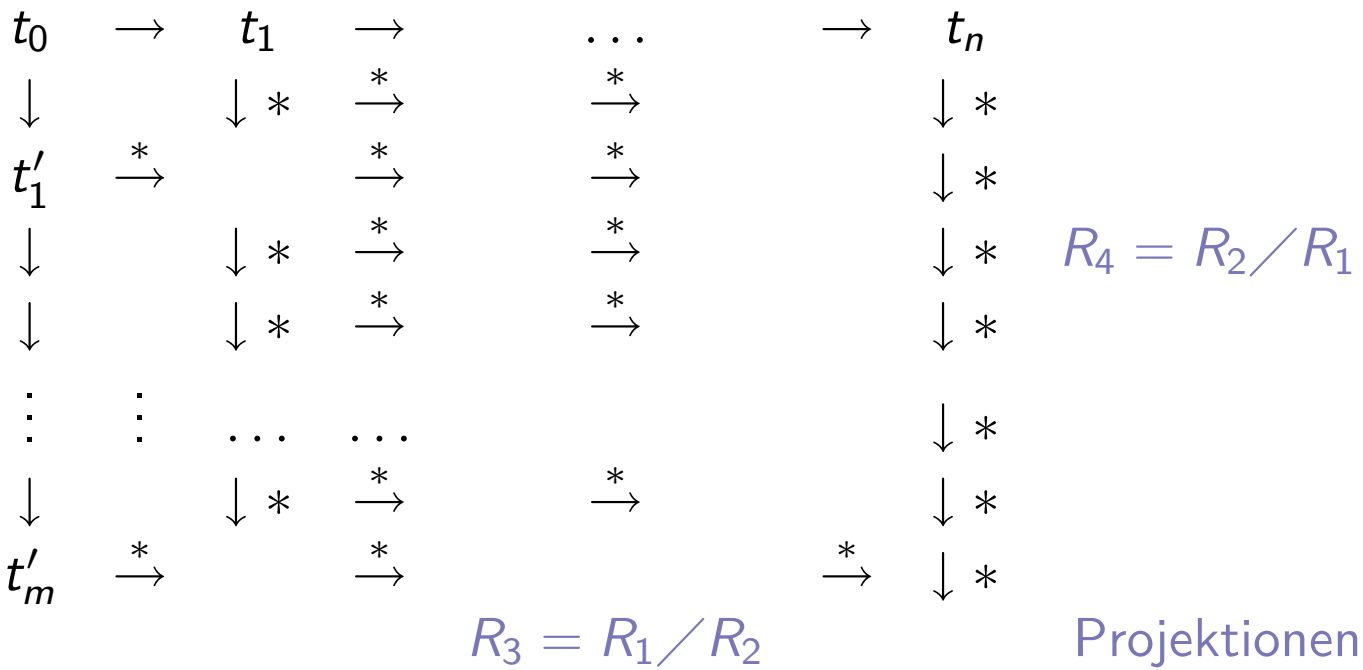
Hilfmittel: Elementare Reduktionsdiagramme und Reduktionsdiagramme:

$$\begin{array}{ccc} Sab(Ic) & \to & a(Ic)(b(Ic)) \\ \downarrow & & \downarrow \\ & & ac(b(Ic)) \\ & & \downarrow \\ Sabc & \to & ac(bc) \end{array} \qquad \begin{array}{ccc} Ka(Ib) & \to & Kab \\ \downarrow & & \downarrow \\ a & \to_{\emptyset} & a \end{array}$$

$$\begin{array}{ccc} Ia & \to & a \\ \downarrow_{\emptyset} & & \downarrow_{\emptyset} \\ Ia & \to & a \end{array} \qquad \begin{array}{ccc} Ia & \to & a \\ \downarrow & & \downarrow_{\emptyset} \\ a & \to_{\emptyset} & a \end{array} \qquad \begin{array}{ccc} a & \to & a \\ a & & a \end{array}$$

Zusammensetzung E-Reduktionsdiagramme

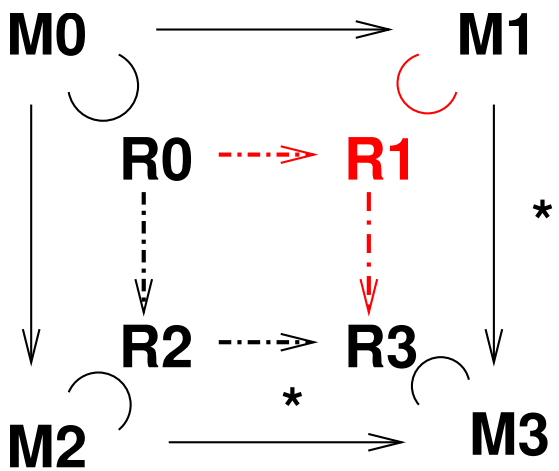
Reduktionsdiagramme und Projektionen:



Seien $R_1 :: t \xrightarrow{+} t'$ und $R_2 :: t \xrightarrow{+} t'$ zwei Reduktionsfolgen von t nach t'.
Sie sind äquivalent $R_1 \cong R_2$ gdw $R_1 / R_2 = R_2 / R_1 = \emptyset$.

Strategien für orthogonale Systeme

Lemma 11.5. *Sei D ein elementares Reduktionsdiagramm für orthogonale Systeme, $R_i \subseteq M_i$ $(i = 0, 2, 3)$ Redexe mit $R_0 - . - . \rightarrow R_2 - . - . \rightarrow R_3$ d.h R_2 ist Rest von R_0 und R_3 ist Rest von R_2. Dann gibt es einen eindeutigen Redex $R_1 \subseteq M_1$ mit $R_0 - . - . \rightarrow R_1 - . - . \rightarrow R_3$, d.h.*



Strategien für orthogonale Systeme

Definition 11.6. *Sei Π ein Prädikat auf Termpaaren M, R so dass $R \subseteq M$ und R ist Redex (z.B. LMOM, LMIM,...).*

i) Π hat Eigenschaft I wenn für das Vorliegen von D wie im Lemma gilt $\Pi(M_0, R_0) \wedge \Pi(M_2, R_2) \wedge \Pi(M_3, R_3) \rightsquigarrow \Pi(M_1, R_1)$

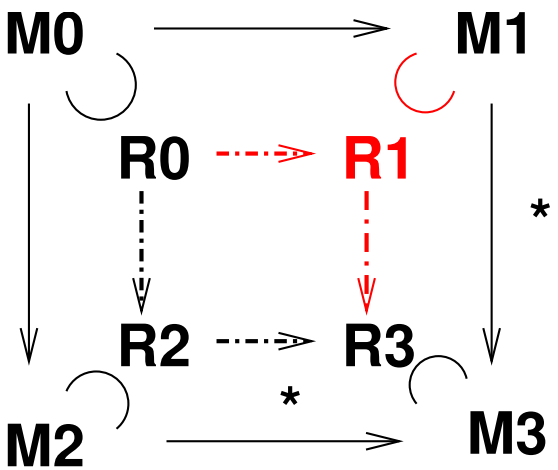
ii) Π hat Eigenschaft II falls in jedem Reduktionsschritt $M \rightarrow^R M'$ mit $\neg\Pi(M, R)$, jeder Redex $S' \subseteq M'$ mit $\Pi(M', S')$ einen Vorgänger-Redex $S \subseteq M$ mit $\Pi(M, S)$ hat.
(D.h. $\neg\Pi$ Schritte produzieren keine neuen Π-Redexe).

Lemma 11.7. *Trennbarkeit von Entwicklungen. Π habe Eigenschaft II. dann kann jede Entwicklung $R :: M_0 \rightarrow \ldots \rightarrow M_n$ aufgeteilt werden in einen Π-Anteil gefolgt von einem $\neg\Pi$-Anteil.*
Genauer: es gibt Reduktionsfolgen $R_\Pi :: M_0 = N_0 \rightarrow^{R_0} \ldots \rightarrow^{R_{k-1}} N_k$ mit $\Pi(N_i, R_i)$ $(i < k)$ und $R_{\neg\Pi} :: N_k \rightarrow^{R_k} \ldots \rightarrow^{R_{k+l-1}} N_{k+l}$ mit $\neg\Pi(N_j, R_j)$ $(k \leq j < k + l)$ und R ist äquivalent zu $R_\Pi \times R_{\neg\Pi}$.

Beispiele

Beispiel 11.8.

- $\Pi(M, R)$ *gdw* R *ist Redex in* M*. I und II gelten.*
- $\Pi(M, R)$ *gdw* R *ist Outermostredex in* M*. dann gelten Eigenschaften I und II: Zu I*



R_0, R_2, R_3 Outermostredexe
Sei S_i der Redex in $M_0 \to M_i$
Ang. nicht OM $\rightsquigarrow$ In M_1 wird durch die Reduktion von S_1 ein Redex (P) erzeugt, der R_1 überdeckt.

In $M_1 \to> M_3$ wird R_1 wieder outermost. D.h. P wird reduziert: In $M_1 \to> M_3$ werden jedoch nur Reste von S_2 reduziert und P ist nicht Rest da neu eingeführt.↯. II ist klar.

- $\Pi(M, R)$ *gdw* R *ist left-most Redex in* M*. I gilt. II nicht immer:* $F(x, b) \to d, a \to b, c \to c :: F(c, a) \to F(c, b)$

Nachfolger von Redexe (Reste)

Definition 11.9. *Spuren in Reduktionsfolgen:*

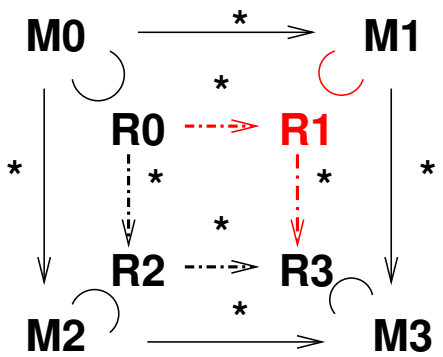
- *Sei* $\mathfrak{R} :: M_0 \to M_1 \to \ldots$ *Reduktionsfolge. Sei* M_j *fest und* $L_i \subseteq M_i$ $(i \geq j)$ *(sofern* M_i *existent) Redexe mit* $L_j - . - . \to L_{j+1} - . - . \to \ldots$.
  Die Folge $\mathfrak{L} = (L_{j+i})_{i \geq 0}$ *ist eine Spur von Nachfolgern (Reste) von Redexe in* M_j.
- $\mathfrak{L}$ *heißt* Π*-Spur, falls* $\forall i \geq j \;\; \Pi(M_i, L_i)$.
- *Sei* R *eine Reduktionsfolge,* Π *Prädikat.* R *ist* Π*-fair, falls* R *keine unendlichen* Π*-Spuren enthält.*

Ergebnisse aus Bergstra, Klop :: Conditional Rewrite Rules: Confluence and Termination. JCSS 32 (1986)

Eigenschaften von Spuren

Lemma 11.10. *Sei Π ein Prädikat mit Eigenschaft I.*

- *Sei $\mathfrak{D}$ ein Reduktionsdiagramm mit $R_i \subseteq M_i, R_0 - . - . \to> R_2 - . - . \to> R_3$ ist Π Spur.*



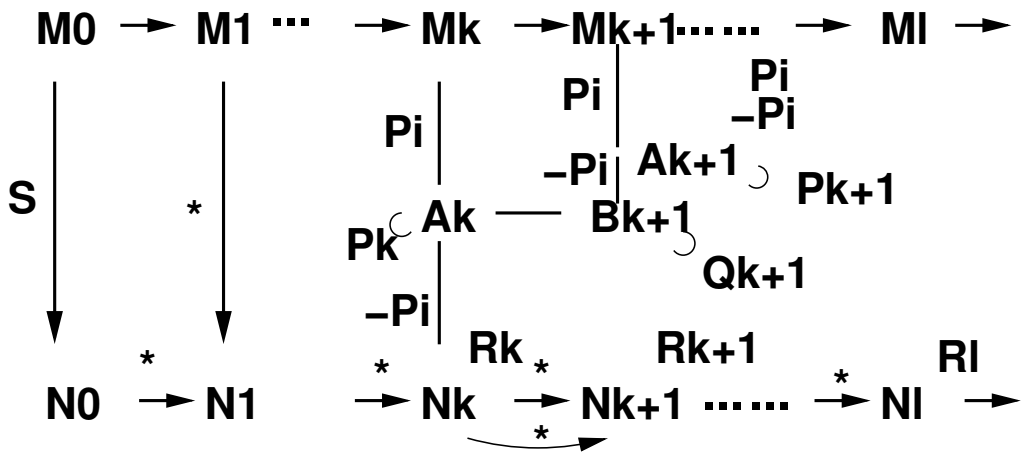
Dann ist $R_0 - . - . \to> R_1 - . - . \to> R_3$ über M_1 ebenfalls eine Π Spur

- *Sind $\mathfrak{R}, \mathfrak{R}'$ äquivalente Reduktionsfolgen von M_0 nach M. $S \subseteq M_0, S' \subseteq M$ Redexe, so dass eine Π-Spur $S - . - . \to> S'$ via $\mathfrak{R}$ existiert. Dann gibt es eine eindeutige Π-Spur $S - . - . \to> S'$ via $\mathfrak{R}'$.*

Hauptsatz von O'Donnell 77

Satz 11.11. *Sei* Π *Prädikat mit Eigenschaften I,II. Dann ist die Klasse der* Π*-fairen Reduktionsfolgen abgeschlossen gegenüber Projektionen.*

Beweisidee:



$\mathfrak{R} :: M_0 \to \ldots$ sei Π-fair und $\mathfrak{R}' :: N_0 \stackrel{*}{\to}$ eine Projektion.$(R_{k+i})\Pi$ Spur. $\forall k \exists A_k . M_k \stackrel{\Pi}{\to}> A_k \stackrel{\neg\Pi}{\to}> N_k$ äquivalent zur vollständigen Entwicklung $M_k \to> N_k$. Bei dieser Umordnung sind die beiden Ableitungen zwischen N_k und N_{k+1} äquivalent. Insbesondere bleiben Π-Spuren erhalten. Ergibt eine Stufenform: $A_k - B_{k+1} - A_{k+1} - B_{k+2} - \ldots$.

Beweis Hauptsatz

Diese Stufen treffen $\mathfrak{R}$ nach endlich vielen Stufen etwa in M_l. Sonst würde $\mathfrak{R}$ eine unendliche Spur von S Reste mit Eigenschaft Π enthalten.

Angenommen $\mathfrak{R}'$ ist nicht Π fair. Es enthalte eine unendliche Spur $R_k, ..., R_{k+1}...$ die von N_k aus startet.

Es gibt Π-Vorgänger $P_k \subseteq A_k$ vom Π-Redex $R_k \subseteq N_k$, d.h mit $\Pi(A_k, P_k)$. Dann kann die Π-Spur $P_k - . - . \rightarrow > R_k - . - . \rightarrow > R_{k+1}$ via B_{k+1} geliftet werden zu Π-Spur $P_k - . - . \rightarrow > Q_{k+1} - . - . \rightarrow > R_{k+1}$.

Iteriert man diese Konstruktion bis M_l, so erhält man einen Redex P_l der Vorgänger von R_l ist mit $\Pi(M_l, P_l)$. Diese Argument kann nun mir R_{l+1} fortgesetzt werden.
Somit ist $\mathfrak{R}$ nicht Π-fair.$\lightning$.

Folgerungen

Lemma 11.12. *Sei $\mathfrak{R} :: M_0 \to M_1 \to \ldots$ eine unendliche Reduktionsfolge mit unendlich vielen outermost Redex-Reduktionen. $S \subseteq M_0$ ein Redex. Dann ist auch $\mathfrak{R}' = \mathfrak{R} / \{S\}$ unendlich.*

Beweis: Angenommen $\mathfrak{R}'$ sei endlich der Länge k. Sei $l \geq k$ und R_l der Redex bei der Reduktion von $M_l \to M_{l+1}$

- Ist R_l outermost, so kann $M'_l \xrightarrow{*} M'_{l+1}$ nur dann leer sein, wenn R_l eines der Reste von S die in $\mathfrak{R}_l$ reduziert worden ist. Somit hat $\mathfrak{R}_{l+1}$ ein Schritt weniger als $\mathfrak{R}_l$.
- Andernfalls ist R_l echt enthalten in Rest von S welcher in $\mathfrak{R}_l$ reduziert wird.

Da aber $\mathfrak{R}$ unendlich viele outermost Redex-Reduktionen enthält muss $\mathfrak{R}_q$ leer werden. Somit stimmt $\mathfrak{R}'$ mit $\mathfrak{R}$ ab einer Stelle überein ist somit auch unendlich.

Folgerungen für orthogonale Systeme

Satz 11.13. *Sei $\Pi(M, R)$ gdw R ist outermost Redex in M.*

- *Die fairen outermost Reduktionsfolgen sind terminierend, wenn sie von einem Term mit einer Normalform starten.*
- *Parallel-Outermost ist normalisierend für orthogonale Systeme.*

Beweis: Hat t eine Normalform, so gibt es keine unendliche Π-faire Reduktionsfolge die mit t startet.
Sei $\mathfrak{R} :: t \to t_1 \to .... \to$ unendlich Π-fair und $\mathfrak{R}' :: t \to t'_1 \to ... \to \bar{t}$ Normalform.
$\mathfrak{R}$ enthält unendlich viele outermost Reduktionsschritte (sonst wäre sie nicht Π-fair). Dann ist aber auch $\mathfrak{R}/\mathfrak{R}'$ unendlich. $\lightning$.

Beachte: Der Satz gilt nicht für LMOM-Strategie: Eigenschaft II ist nicht erfüllt. Betrachte hierfür $a \to b, c \to c, f(x, b) \to d$.

Folgerungen für orthogonale Systeme

Definition 11.14. *Sei R orthogonal, $l \to r \in R$ heißt linksnormal, falls in l alle Funktionssymbole von den Variablen vorkommen. R ist linksnormal, falls alle Regeln in R linksnormal sind.*

Folgerung 11.15. *Sei R linksnormal dann gilt:*

- *Faire leftmost Reduktionsfolgen sind terminierend für Terme mit Normalformen.*
- *Die LMOM-Strategie ist normalisierend.*

Beweis: Sei $\Pi(M, L)$ gdw L ist LMO-Redex in M. Dann gelten Eigenschaften I und II. Für II benötigt man linksnormal.
Nach Satz 11.2 sind Π-faire Reduktionsfolgen abgeschlossen unter Projektionen. Aus Lemma 11.4 folgt wie eben die Behauptung.

Zusammenfassung

Eine Strategie heißt ewig falls sie unendliche Reduktionsfolgen erzeugen kann.

Strategie	Orthogonale	LN-Ortogonal	Orthogonal-NE
LMIM	*e*	*e*	*e n*
PIM	*e*	*e*	*e n*
LMOM		*n*	*e n*
POM	*n*	*n*	*e n*
FSR	*n c*	*n c*	*e n c*

Klassifikation von TES nach Vorkommen von Variablen

Definition 11.16. *Sei R TES, $Var(r) \subseteq Var(l)$ für $l \to r \in R, x \in Var(l)$.*

- *R heißt variablenreduzierend, falls für alle $l \to r \in R, |l|_x > |r|_x$*
  R heißt variablenerhaltend, falls für alle $l \to r \in R, |l|_x = |r|_x$
  R heißt variablenvermehrend, falls für alle $l \to r \in R, |l|_x \leq |r|_x$
- *Sei $D[t, t']$ eine Ableitung von t nach t'. $|D[t, t']|$ sei die Länge der Reduktionsfolge. $D[t, t']$ ist optimal falls sie minimale Länge unter allen Ableitungen von t nach t' hat.*

Lemma 11.17. *Sei R orthogonal variablenerhaltend. Dann bleibt in jeder Reduktionsfolge jeder Redex erhalten, es sei denn er wird reduziert. Jede Ableitungsfolge ist optimal.*

Beweis: Austauschtechnik: Reste bleiben Reste soweit sie nicht reduziert werden, d.h. Reduktionsschritte können ausgetauscht werden.

Beispiele

Beispiel 11.18. *Längen von Ableitungen:*

- *Variablenerhaltend:*
  $R :: f(x,y) \to g(h(x),y), g(x,y) \to l(x,y), a \to c, b \to d.$
  Betrachte den Term $f(a,b)$ *und seine Ableitungen.*
  Alle Ableitungsfolgen sind gleich lang.
- *Variablenvermehrend (non erasing):*
  $R :: f(x,b) \to g(x,x), a \to b, c \to d.$
  Betrachte den Term $f(c,a)$ *und seine Ableitungen.*
  Innermost Ableitungsfolgen sind kürzer.

Weitere Ergebnisse

Lemma 11.19. *Sei R überlappungsfrei, variablenvermehrend. Dann bleibt ein innermost Redex solange erhalten bis er reduziert wird.*

Satz 11.20. *Sei R orthogonal variablenvermehrend (ne). Sei $D[t, t']$ eine Ableitungsfolge von t zu seiner Normalform t' die nicht innermost ist. Dann gibt es eine innermost Ableitung $D'[t, t']$ mit $|D'| \leq |D|$.*

Beweis: Sei $L(D)$ = Länge der Ableitung von der ersten nicht innermost Reduktion in D nach t'.
Induktion nach $L(D) :: t \to t_1 \to \ldots \to t_i \xrightarrow{S} \ldots \to t_j \xrightarrow{*} t'$.
Sei i diese Stelle.
S ist nicht innermost und t_i enthält innermost Redex S_i der später reduziert werden muss, etwa bei Reduktion von t_j. Betrachte die

Reduktionsfolge $D' :: t \to t_1 \to \ldots \to t_i \xrightarrow{S_i} t'_{i+1} \xrightarrow{S} \ldots t'_j \xrightarrow{<}{*} t'$
$|D'| \leq |D|, L(D') < L(D) \rightsquigarrow$ es gibt Ableitung D' mit $L(D') = 0$.

Weitere Ergebnisse

Satz 11.21. *Sei R überlappungsfrei, variablenvermehrend. Je zwei innermost Ableitungen zu einer Normalform sind gleich lang.*

Klar da innermost Redexe disjunkt sind und erhalten bleiben solange sie nicht reduziert werden.
Folgerung: R orthogonal variablenvermehrend. Dann sind innermost Ableitungen optimal. Insbesondere ist LMIM optimal für solche Systeme.

Beispiel 11.22. *Gibt es mehrere outermost Redexe so hängt die Länge der Ableitungsfolgen von der Wahl der Redexe ab. Betrachte:*
$f(x,c) \to d, a \to d, b \to c$ *und die Ableitungen:*
$f(\underline{a},b) \to f(d,\underline{b}) \to \underline{f(d,c)} \to d$ *bzw.* $f(a,\underline{b}) \to \underline{f(a,c)} \to d$

$\rightsquigarrow$ *Strategie Variablenverzögerung. Bleibt ein outermost Redex nach einer Reduktion nicht outermost, so liegt er unterhalb einer Variablen eines bei der Reduktion entstandenen Redexes. Löscht diese Regel gerade diese Variable, so muss der Redex nicht reduziert werden.*

Weitere Ergebnisse

Satz 11.23. *Sei R überlappungsfrei.*

- *Sei D eine outermost Ableitung und L ein nicht-variablen outermost Redex in D. Dann bleibt L ein nicht-variablen outermost Redex bis er reduziert wird.*
- *Sei R linear. Zu jeder outermost Ableitung $D[t, t']$, t' Normalform, gibt es eine variablenverzögernden Ableitung $D[t, t']$ mit $|D'| \leq |D|$. Somit sind variablenverzögernden Ableitungen optimal.*

Satz 11.24. *Ke Li. Folgendes Problem ist NP-vollständig:*

Eingabe: *Ein konvergentes TES R, Term t und $D[t, t\downarrow]$.*

Frage: *Gibt es eine Ableitung $D'[t, t\downarrow]$ mit $|D'| < |D|$.*

Beweisidee: Reduziere 3-SAT auf dieses Problem.

Berechenbare Strategien

Definition 11.25. *Eine Reduktionsstrategie $\mathfrak{S}$ ist berechenbar, falls die Abbildung $\mathfrak{S} : Term \to Term$ mit $t \xrightarrow{*} \mathfrak{S}(t)$ rekursiv ist.*

Beachte: Die Strategien LMIM, PIM, LMOM, POM, FSR sind polynomial berechenbar.

Frage: Gibt es eine einschritt berechenbare normalisierende Strategie für orthogonale Systeme ?.

Beispiel 11.26.

- *(Berry) CL-Kalkül erweitert um Regeln $FABx \to C, FBxA \to C, FxAB \to C$ ist orthogonal, nicht links-normal. Welches Argument wählt man zur Reduktion von FMNL aus? Jedes Argument kann nach A bzw B ausgewertet werden, dies ist aber unentscheidbar in CL.*
- *Betrachte $or(true, x) \to true, or(x, true) \to true$ + CL. Parallel Auswertung scheint notwendig zu sein!*

Berechenbare Strategien: Gegenbeispiel

Beispiel 11.27. *Signatur: 0-stellig:* $S, K, S', K', C, 0, 1$
1-stellig: A, *activate* *2-stellig:* ap, ap' *3-stellig:* B
Regeln:
$ap(ap(ap(S,x),y),z) \to ap(ap(x,y),ap(y,z))$
$ap(ap(K,x),y) \to x$
$activate(S') \to S, \quad activate(K') \to K$
$activate(ap'(x,y)) \to ap(activate(x), activate(y))$
$A(x) \to B(0,x,activate(x)), \quad A(x) \to B(1,x,activate(x))$
$B(0,x,S) \to C, \quad B(1,x,k) \to C, \quad B(x,y,z) \to A(y)$
Terme: *Ausgehend von Terme der Form* $A(t)$ *wobei* t *aus* S', K' *und* ap' *gebildet wird.*
Behauptung: *R ist konfluent, hat aber keine berechenbare einschritt Strategie die normalisierend ist.*

Eine sequentielle Strategie für paror Systeme

Beispiel 11.28. *Seien $f, g : \mathbb{N}^+ \to \mathbb{N}$ rekursive Funktionen. Definiere Reduktionssystem R auf $\mathbb{N} \times \mathbb{N}$ mit Regeln:*

- $(x, y) \to (f(x), y)$ *falls* $x, y > 0$
- $(x, y) \to (x, g(y))$ *falls* $x, y > 0$
- $(x, 0) \to (0, 0)$ *falls* $x > 0$
- $(0, y) \to (0, 0)$ *falls* $y > 0$

Offenbar ist R konfluent. Einzige Normalform ist $(0, 0)$ und für $x, y > 0$, (x, y) hat Normalform gdw. $\exists n. f^n(x) = 0 \vee g^n(x) = 0$.
Eine einschritt Reduktionsstrategie muss wählen zwischen der Anwendung von f bzw. g auf erste bzw. zweite Argument.
Eine solche Reduktionsstrategie kann nicht erst Nullstellen von $f^n(x)$ bzw. $g^n(y)$ berechnen um das gewünschte Argument zu wählen. Man würde erwarten, dass es geeignete Funktionen f und g gibt, für die es keine berechenbare Einschrittstrategie gibt. Dies ist aber nicht der Fall.

Eine sequentielle Strategie für paror Systeme

Es gibt eine effektive Einschrittreduktionsstrategie die normalisierend ist.

Lemma 11.29. *Sei $(x, y) \in \mathbb{N} \times \mathbb{N}$. Dann gilt:*

- *$x < y$:: Für ein n gilt entweder $f^n(x) = 0$, oder $f^n(x) \geq y$ oder es gibt $i < n$ mit $f^n(x) = f^i(x) \neq 0$. Wähle n minimal mit diese Eigenschaft. Die Alternativen schließen sich aus. Trifft eine der beiden ersten so $\mathfrak{S}(x, y) = L$ sonst R*
- *$x \geq y$:: Für ein n gilt entweder $g^n(y) = 0$, oder $g^n(y) > x$ oder es gibt $i < n$ mit $g^n(y) = g^i(y) \neq 0$. Wähle n minimal mit diese Eigenschaft. Die Alternativen schließen sich aus. Trifft eine der beiden ersten so $\mathfrak{S}(x, y) = R$ sonst L*
- *Behauptung: $\mathfrak{S}$ ist eine berechenbare Einschrittreduktionsstrategie für R die normalisierend ist. (Beweis: Übung)*

Berechenbare Strategien

Satz 11.30. *Kennaway (Annals of Pure and Applied Logic 43(89))*
Für jedes orthogonale System gibt es eine berechenbare sequentielle normalisierende Reduktionsstrategie.

Definition 11.31. *Standard Reduktionsfolgen*
Sei $\mathfrak{R} :: t_0 \to t_1 \to \ldots$ eine Reduktionsfolge im TES R. Markiere in jedem Schritt in $\mathfrak{R}$ alle Top-Symbole von Redexe die links vom reduziertem Redex vorkommen. $\mathfrak{R}$ ist eine standard Reduktionsfolge falls kein Redex mit markiertem Top-Symbol reduziert wird.

Satz 11.32. *Standarisierungssatz für links-normale orthogonale TES.*
Sei R LNO.
Gilt $t \xrightarrow{*} s$, so gibt es eine standard Reduktion in R mit $t \xrightarrow{*}_{ST} s$.
Insbesondere ist LMOM normalisierend.

Sequentielle Orthogonale TES

Beispiel 11.33. *Für applikative TES:: $PxQ \to xx, R \to S, Ix \to x$*
Betrachte $\mathfrak{R} :: PR(\underline{IQ}) \to \underline{PRQ} \to \underline{R}R \to SR$
Es gibt keine standard Reduktion von $PR(IQ)$ nach SR

Fakt: λ-Kakül und CL-Kalkül sind sequentiell, d.h. es wird stets Redex reduziert der notwendig zur Berechnung der Normalform ist.

Definition 11.34. *R sei Orthogonal, $t \in Term(R)$ mit Normalform $t\downarrow$.*
*Ein Redex $s \subseteq t$ ist ein **notwendiger** Redex, falls in jeder Reduktionsfolge $t \to ... \to t\downarrow$ ein Rest von s reduziert wird.*

Sequentielle Orthogonale TES: Call-by-need

Satz 11.35. *Huet- Levy (1979) Sei R Orthogonal*

- *Sei t mit einer Normalform aber reduzibel , so enthält t einen notwendigen Redex*
- *"Call-by-need" Strategie (notwendige Redexe) ist normalisierend*
- *Faire notwendige-Redexe Reduktionsfolgen sind terminierend*

Lemma 11.36. *Sei R orthogonal, $t \in Term(R)$, s, s' Redexe in t mit $s \subseteq s'$. Ist s notwendig, so auch s'.*
Insbesondere:: Ist t nicht in Normalform, so ist ein outermost Redex notwendig.

Sei $C[..., ..., ...]$ ein Kontext mit n-Plätze (Löcher), σ eine Substitution der Redexe $s_1, ..., s_n$ in Plätze $1, ..., n$. Das Lemma impliziert folgende Eigenschaft:
$\forall C[..., ..., ...]$ in Normalform, $\forall \sigma \exists i.s_i$ notwendig in $C[s_1, ..., s_n]$.
Welches s_i notwendig ist, hängt von σ ab.

Sequentielle Orthogonale TES

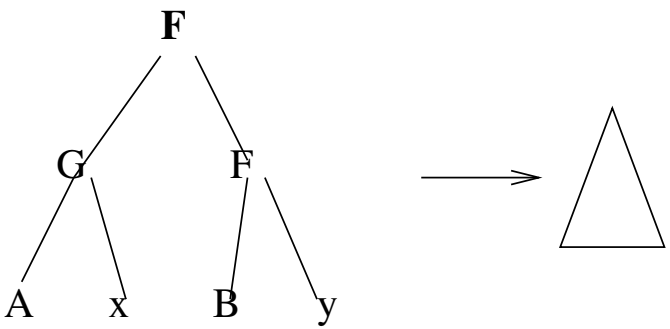
Definition 11.37. *Sei R orthogonal.*

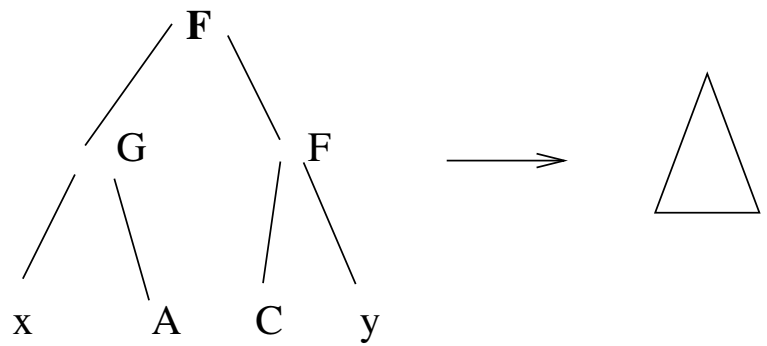
- *R ist sequentiell* gdw. $\forall C[\ldots,\ldots,\ldots]$ in normalform $\exists i \forall \sigma.s_i$ ist notwendig in $C[s_1,\ldots,s_n]$*
  Leider ist diese Eigenschaft unentscheidbar
- *Sei $C[\ldots]$ Kontext. Die Reduktionsrelation $\rightarrow_?$ (mögliche Reduktion) ist definiert durch*

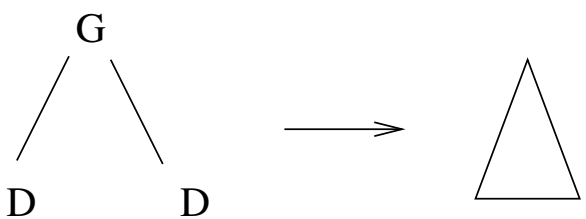
  $$C[s] \rightarrow_? C[r] \text{ für jeden Redex } s \text{ und beliebigen Term } r$$

  $\rightarrow_?^*$ und Reste analog definiert.
- *Ein Redex s in t heißt **stark notwendig** wenn in jeder Reduktionsfolge $t \rightarrow_? \ldots \rightarrow_? t'$, wobei t' eine Normalform ist, ein Nachfolger von s reduziert wird.*
- *R ist **stark sequentiell** falls $\forall C[\ldots,\ldots,..]$ in Normalform $\exists i \forall \sigma.s_i$ stark notwendig.*

Beispiel







Ist nicht stark sequentiell F(G(1,2),F(G(3,4),5))

Starke Sequentialität

Lemma 11.38. *Sei R orthogonal.*

- *Die Eigenschaft Starke-Sequentialität ist entscheidbar. Der notwendige Index i ist berechenbar.*
  Beweis: Siehe Huet-Levy
- *Call-by-need ist effektive Einschrittstrategie für solche Systeme*