






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.

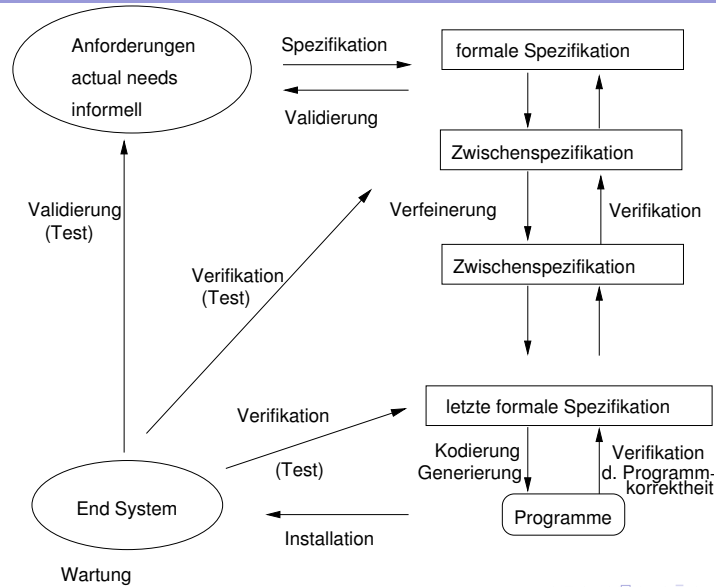
 E. Börger, R. Stärk
Abstract State Machines: A Method for High-Level System Design and Analysis. Springer, 2003.

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



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**:

<p>Modell orientiert (konstruktiv) VDM, Z, B, ASM Konstruktion eines eindeutigen Modells aus vorhandenen Datenstrukturen und Konstruktionsregeln Korrektheitsbegriff</p>	- -	<p>Eigenschaften orientiert (deklarativ) Signatur (Funktionen, Prädikate) Eigenschaften (Formeln Axiome) Modelle algebraische Spezifikation AFFIRM, OBJ, ASF, ...</p>
---	-----	--
- ▶ 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)
- ▶ Kontrollabstraktion (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: $\forall X t_1 = t_2 \quad t_1, t_2 \text{ Terme.}$

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. $t' \in \text{Wert}(\text{spec})$ mit $\text{spec} \models t = t'$.
 - ▶ \rightsquigarrow Funktionale Programmierung: LISP, CAML, ...
 $F(t_1, \dots, t_n) \quad \text{eval}() \rightsquigarrow \text{Wert.}$

Beispiel: Modellbasiert konstruktiv: VDM

Eindeutigkeit, Standard (Notationen), implementierungsunabhängig, formal manipulierbar, abstrakt, strukturiert, expressiv, Konsistenz

Beispiel: Model (zustands)-basierte Spezifikationstechnik VDM

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

Operationen in VDM

VDM-SL: System Zustand, Operationsspezifikation

Format:

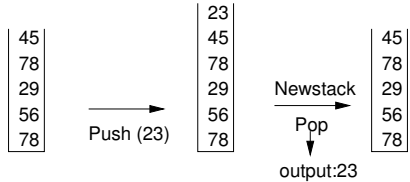
Operation-Identier (Inputparameters) Output Parameters
Pre-Condition
Post-Condition

z. B.

$\text{Int_SQR}(x : \mathbb{N})z : \mathbb{N}$
pre $x \geq 1$
post $(z^2 \leq x) \wedge (x < (z + 1)^2)$

Beispiel VDM: Beschränkter Keller

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



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

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

Beschränkter Keller

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

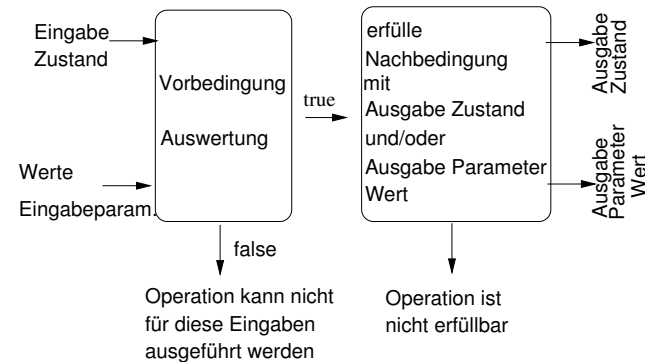
Full() b : \mathbb{B}
 ext rd s : Contents
 rd n : Max_Stack_Size
 pre true
 post b \Leftrightarrow (len s = n)

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

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

\rightsquigarrow Proof-Obligations

Allgemeine Form VDM-Operationen



Allgemeine Form VDM-Operationen

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

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

Falls Zustandsinvarianten vorhanden:

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

bzw.

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

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

Keller algebraisch spezifiziert

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

SPEC STACK
USING NATURAL, BOOLEAN "Namen bekannter SPEC's"
SORT stack "Hauptsorte"
OPS init : \rightarrow stack "Konstante der Sorte stack, leerer Keller"
 push : stack nat \rightarrow stack
 pop : stack \rightarrow stack
 top : stack \rightarrow nat
 is_empty? : stack \rightarrow bool
 stack_error : \rightarrow stack
 nat_error : \rightarrow 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.

Beispiel

Beispiel 3.15. Maximale Interval-Summe.[Gries 1990]. Sei A eine Funktion von $\{0, 1, \dots, n-1\} \rightarrow \mathbb{R}$ und $i, j, k \in \{0, 1, \dots, n\}$. Sei für $i \leq j$ beliebig $S(i, j) \Leftarrow \sum_{i \leq k < j} A(k)$. Insbesondere $S(i, i) = 0$.

Problem: Berechne $S \Leftarrow \max_{i \leq j} S(i, j)$.

Definiere $y(k) \Leftarrow \max_{i \leq j \leq k} S(i, j)$. 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 $x(k) \Leftarrow \max_{i \leq k} S(i, k)$, 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

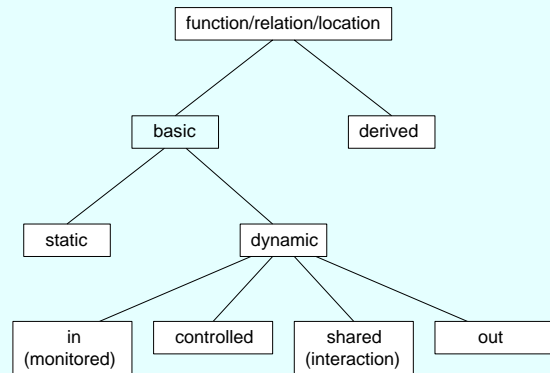
Signatures

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 $f^{\mathfrak{A}}: X^n \rightarrow X$.
- 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 $(a_1, \dots, a_n) \in |\mathfrak{A}|^n$ such that $f^{\mathfrak{A}}(a_1, \dots, a_n) \neq \text{undef}^{\mathfrak{A}}$.
- A *relation* is a function that has the values *true*, *false* or *undef*.
- We write $a \in R$ as an abbreviation for $R(a) = \text{true}$.
- The superuniverse can be divided into *subuniverses* represented by unary relations.

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 $\mathfrak{A} + (U \oplus V) = (\mathfrak{A} + U) + V.$

Terms

Let Σ be a signature.

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

- Variables x, y, z, \dots are terms.
- Constants c of Σ are terms.
- If f is an n -ary function name of $\Sigma, n > 0,$ and t_1, \dots, t_n are terms, then $f(t_1, \dots, t_n)$ 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_x^s we denote the result of replacing the variable x in term t everywhere by the term s (*substitution* of s for x in t).

Part 2

Mathematical Logic

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* $\llbracket t \rrbracket_{\zeta}^{\mathfrak{A}}$ is defined as follows:

- $\llbracket x \rrbracket_{\zeta}^{\mathfrak{A}} = \zeta(x)$
- $\llbracket c \rrbracket_{\zeta}^{\mathfrak{A}} = c^{\mathfrak{A}}$
- $\llbracket f(t_1, \dots, t_n) \rrbracket_{\zeta}^{\mathfrak{A}} = f^{\mathfrak{A}}(\llbracket t_1 \rrbracket_{\zeta}^{\mathfrak{A}}, \dots, \llbracket t_n \rrbracket_{\zeta}^{\mathfrak{A}})$

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 $\neg(s = t)$.

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 $\llbracket t \rrbracket_{\zeta}^{\mathfrak{A}} = \llbracket t \rrbracket_{\eta}^{\mathfrak{A}}$.

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

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

Formulas (continued)

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

Variations of the syntax (continued)

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

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

Beispiel

Beispiel 3.17. *Sortieren linearer Datenstrukturen in-place, one-swap-a-time.*

Sei $a : \text{Index} \rightarrow \text{Value}$

choose $x, y \in \text{Index} : x < y \wedge a(x) > a(y)$
do in-parallel
 $a(x) := a(y)$
 $a(y) := a(x)$

Zwei Arten von Nichtdeterminismus:

“Don't-care” Nichtdeterminismus: Random choice

choose $x \in \{x_1, x_2, \dots, x_n\}$ with $\varphi(x)$ do
 $R(x)$

“Don't-know” Indeterminismus

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

monitored $f : X \rightarrow Y$

Rule declarations

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

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

where

- P is a transition rule and
- the free variables of P are contained in the list x_1, \dots, 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 (continued)

$$\frac{}{\text{yields}(\mathbf{skip}, \mathfrak{A}, \zeta, \emptyset)}$$

$$\frac{\text{yields}(f(s_1, \dots, s_n) := t, \mathfrak{A}, \zeta, \{(l, v)\})}{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A}, \zeta, V)} \quad \text{where } l = (f, ([s_1]_{\zeta}^{\mathfrak{A}}, \dots, [s_n]_{\zeta}^{\mathfrak{A}}))$$

$$\frac{\text{yields}(P, \mathfrak{A}, \zeta, U) \quad \text{yields}(Q, \mathfrak{A}, \zeta, V)}{\text{yields}(P \mathbf{par} Q, \mathfrak{A}, \zeta, U \cup V)} \quad \text{and } v = [t]_{\zeta}^{\mathfrak{A}}$$

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

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

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

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

Semantics of transition rules

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

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

The predicate

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

means:

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

Semantics of transition rules (continued)

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

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

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

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

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

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

Wohlfundierte Induktion

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

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

Korrektheit:

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

FP-Induktion

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

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

Korrektheit: Sei $X \subseteq D$ zulässig.

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

Aufgabe

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

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

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

Betrachte

$$\begin{aligned} F : D &\rightarrow \mathbb{N} \times \mathbb{N} \\ g &\mapsto \begin{cases} \{(0, 1)\} & g = \emptyset \\ \{(x, x \cdot g(x-1)) : x-1 \in \text{dom}(g)\} & \text{sonst} \end{cases} \end{aligned}$$

Aufgabe

Zeigen Sie:

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

Bemerkung:

- ▶ μF kann aufgefasst werden als **Semantik** einer Funktionsdefinition

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

- ▶ 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 $\varrho \equiv (M, \lambda, \sigma)$ mit folgenden Eigenschaften gegeben:

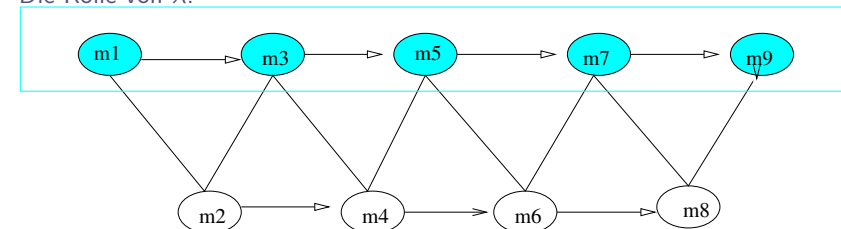
- ▶ 1. M ist eine partiell geordnete Menge von Zügen (moves), wobei jeder Zug nur endlich viele Vorgänger hat.
- ▶ 2. λ ist eine Funktion auf M , die jedem Zug einen Agenten zuordnet, so dass die Züge eines einzelnen Agenten stets linear geordnet sind.
- ▶ 3. σ assoziiert einen Zustand von A mit jedem 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 .

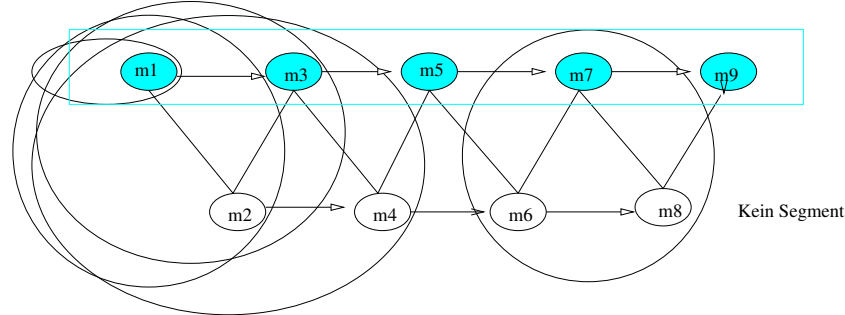
Ein Lauf kann als der gemeinsame Anteil der Historie der gleichen Berechnung aus der Sicht mehrerer Beobachter.

Die Rolle von λ :



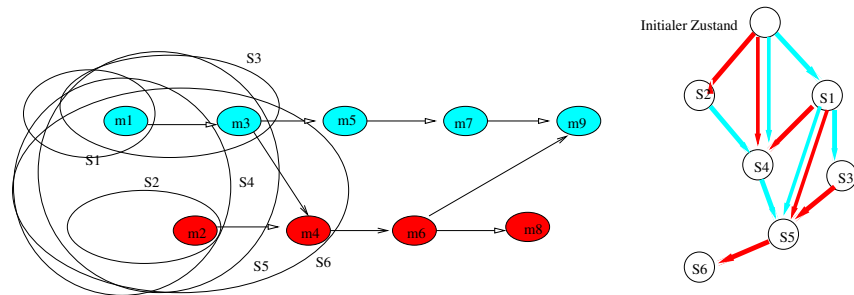
Bemerkung, Beispiel

Die Rolle von σ : Momentaufnahmen sind die initialen Segmente der partiell geordneten Menge M . Jedem initialen Segment wird ein Zustand von A (Interpretation von Σ) zugeordnet, der die Wirkung der Programme der im Segment vorkommenden Agenten wiedergibt.
 ~→ "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:

```

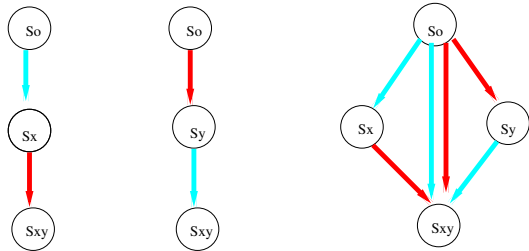
program_d = door := true // move x
program_w = window := true // move y
    
```

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



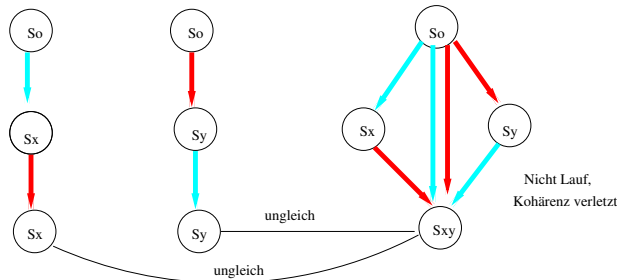
Variante von Einfaches Beispiel

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

```

programd = if ¬window then door := true // move x
programw = if ¬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.

insert : $Item \times ItemList \rightarrow ItemList$
remove : $ItemList \rightarrow (Item \times ItemList)$

- (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

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

Hierbei wird die Zeit von einer diskreten Systemuhr gemessen:

$$monitored\ now : \rightarrow TIME$$

Geldautomat

Übung 4.9. Abstrakte Modellierung eines Geldautomaten: Drei Agenten sind im Modell: GA-Manager, Authentifikations-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.

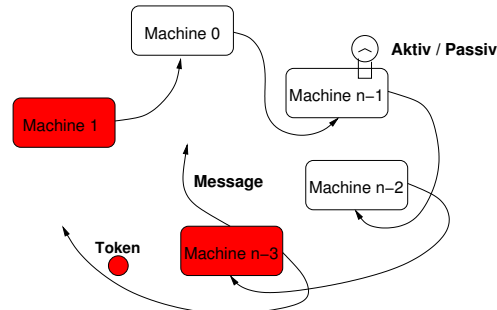
Distributed Termination Detection

Beispiel 4.10. *Modelliere folgendes Terminierungsfeststellungsprotokoll:*

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

$COLOR = \{red, white\}$ $TOKEN = \{redToken, whiteToken\}$
 $MACHINE = \{0, 1, 2, \dots, n-1\}$
 $next : MACHINE \rightarrow MACHINE$
 Z.B. mit $next(0) = n-1, next(n-1) = n-2, \dots, next(1) = 0$

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* =

$$\begin{array}{l}
 \text{ReactOnEvents}(me) \\
 \text{if } \neg \text{Active}(me) \wedge \text{token}(me) \neq \text{undef} \text{ then} \quad \text{Rule 0} \\
 \quad \text{InitializeMachine}(me) \quad \text{Rule 5} \\
 \quad \text{if } \text{color}(me) = \text{red} \text{ then} \\
 \quad \quad \text{Forward}(me, \text{redToken}) \quad \text{Rule 2} \\
 \quad \text{else} \\
 \quad \quad \text{Forward}(me, \text{token}(me)) \quad \text{Rule 2} \\
 \text{if } \neg \text{Active}(me) \wedge \text{token}(me) = \text{undef} \text{ then} \quad \text{Rule 0} \\
 \quad \text{InitializeMachine}(me) \quad \text{Rule 5} \\
 \quad \text{if } \text{color}(me) = \text{red} \text{ then} \\
 \quad \quad \text{Forward}(me, \text{redToken}) \quad \text{Rule 2} \\
 \quad \text{else} \\
 \quad \quad \text{Forward}(me, \text{token}(me)) \quad \text{Rule 2}
 \end{array}$$
- ▶ Mit *InitializeMachine*($m : MACHINE$) =

$$\begin{array}{l}
 \text{token}(m) := \text{undef} \\
 \text{color}(m) := \text{white}
 \end{array}$$

Distributed Termination Detection: Verfahren

Programme

- ▶ *SupervisorMachineProgram* =

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

Distributed Termination Detection

Initiale Zustände

- $\exists m_0 \in MACHINE$

$$\begin{array}{l}
 (\text{program}(m_0) = \text{SupervisorMachineProgram} \wedge \\
 \text{token}(m_0) = \text{redToken} \wedge \\
 (\forall m \in MACHINE)(m \neq m_0 \Rightarrow \\
 \quad (\text{program}(m) = \text{RegularMachineProgram} \wedge \text{token}(m) = \text{undef})))
 \end{array}$$

Umgebungsconstraints

Für alle Läufe und alle Linearisierungen gilt:

- G** $(\forall m \in MACHINE)$

$$\begin{array}{l}
 ((\text{SendMessageEvent}(m) = \text{true} \Rightarrow (\mathbf{P}(\text{Active}(m)) \wedge \text{Active}(m))) \wedge \\
 ((\text{Active}(m) = \text{true} \wedge \mathbf{P}(\neg \text{Active}(m)) \Rightarrow \\
 \quad (\exists m' \in MACHINE) (m' \neq m \wedge \text{SendMessageEvent}(m'))))
 \end{array}$$

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

Distributed Termination Detection

$$(\forall i : t < i < n \text{ Machine}_i \text{ ist passiv}) \vee (\exists j : 0 \leq j \leq t \text{ Machine}_j \text{ ist rot}) \vee (\text{Token ist rot})$$

Korrektheitsargument

Wenn das Token zu Machine_0 gelangt ist $t = 0$ und die Invariante gilt. Falls

$$(\text{Machine}_0 \text{ ist passiv}) \wedge (\text{Machine}_0 \text{ ist weiß}) \wedge (\text{Token ist weiß})$$

so muß

$(\forall i : 0 < i < n \text{ Machine}_i \text{ ist passiv})$ 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 nachrichtnerhaltende 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)
- ⋮

Mehrsortige Algebren

Axiome sind allquantifizierte Gleichungen, d.h.

$$\forall x_1, \dots, x_n : t_1(x_1, \dots, x_n) = t_2(x_1, \dots, x_n) \text{ wobei}$$

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

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

$$\begin{aligned} \text{eq}(0, 0) &= \text{true} & \text{eq}(0, \text{suc}(n)) &= \text{false} \\ \text{eq}(\text{suc}(n), 0) &= \text{false} & \text{eq}(\text{suc}(n_1), \text{suc}(n_2)) &= \text{eq}(n_1, n_2) \end{aligned}$$

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

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

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:

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

$$\text{app}(3.1.\text{nil}, \text{app}(5.\text{nil}, 1.2.3.\text{nil})) \xrightarrow{R} 3.1.5.1.2.3.\text{nil}$$

$$\begin{aligned} \text{rev}(3.1.\text{nil}) &\rightarrow \text{app}(\text{rev}(1.\text{nil}), 3.\text{nil}) \\ &\rightarrow \text{app}(\text{app}(\text{rev}(\text{nil}), 1.\text{nil}), 3.\text{nil}) \\ &\rightarrow \text{app}(\text{app}(\text{nil}, 1.\text{nil}), 3.\text{nil}) \\ &\rightarrow \text{app}(1.\text{nil}, 3.\text{nil}) \end{aligned} \quad \text{Frage: Gilt}$$

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

Mehrsortige Algebren

Manche Gleichungen gelten nicht in allen Modellen von $\text{EQN} = E$. z. B.

$$\begin{aligned} x + y &\neq_E y + x \\ \text{app}(x, \text{app}(y, z)) &\neq_E \text{app}(\text{app}(x, y), z) \\ \text{rev}(\text{rev}(x)) &\neq_E x \end{aligned}$$

die Termpaare sind nicht zusammenführbar.

Unterscheidung:

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

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

$$\text{rev}(\text{rev}(x)) = x \text{ f\u00fcr } x \equiv s^i 0 . s^j 0 . \dots . s^n 0 . \text{nil}$$

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

Striktheit - Positionen-Teilterme

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

b) Teilterme $(t) = \{t_p \mid p \text{ Stelle (Position) in } t, t_p \text{ Teilterm in } p\}$
 Stellen werden durch Folgen über \mathbb{N} dargestellt (Elemente von \mathbb{N}^* , ϵ 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: $O(t) = \{\epsilon\}$ $t_\epsilon \equiv t$
- ▶ $t \equiv f(t_1, \dots, t_n)$ so
 $O(t) = \{i p \mid 1 \leq i \leq n, p \in O(t_i)\} \cup \{\epsilon\}$
 $t_{i p} \equiv t_i|_p$ und $t_\epsilon \equiv t$.

Signaturen - Terme

Beispiel 6.4. $S = (\text{BOOL}, \text{NAT}, \text{LIST})$, $F = \{\text{true}, \text{false}, \dots\}$,
 $\tau : F \rightarrow S^* :: \text{true} \rightarrow \text{BOOL}, \text{eq} : \text{NAT}, \text{NAT} \rightarrow \text{BOOL} \dots$

$$V = \begin{matrix} V_{\text{BOOL}} & \cup & V_{\text{NAT}} & \cup & V_{\text{LIST}} \\ \text{"} & & \text{"} & & \text{"} \\ \{b_i : i \in \mathbb{N}\} & & \{x_i : i \in \mathbb{N}\} & & \{l_j : i \in \mathbb{N}\} \end{matrix}$$

Grundterme:

$\text{true}, \text{false}, \text{eq}(0, \text{succ}(0)) \in \text{Term}_{\text{BOOL}}(S)$
 $0, \text{succ}(0), \text{succ}(0) + (\text{succ}(\text{succ}(0)) + 0) \in \text{Term}_{\text{NAT}}(S)$
 $\text{app}(\text{nil}, \text{succ}(0)).(\text{succ}(\text{succ}(0)).\text{nil}) \in \text{Term}_{\text{LIST}}(S)$
 $0, \text{succ}(0), \text{eq}(\text{true}, \text{false}), \text{rev}(0)$ keine Terme.

Allgemeine Terme:

$\text{eq}(x_1, x_2) \in \text{Term}_{\text{BOOL}}(F, V)$, $\text{succ}(x_1) + (x_2 + \text{succ}(0)) \in \text{Term}_{\text{NAT}}(F, V)$
 $\text{app}(l_1, x_1.l_0) \in \text{Term}_{\text{LIST}}(F, V)$
 $\text{rev}(x_1.l) \in \text{Term}_{\text{LIST}}(F, V)$
 $\text{app}(x_1, l_2)$ kein Term.

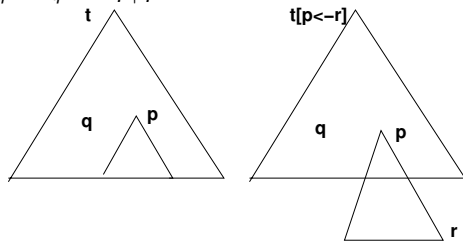
Termersetzung

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

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 .

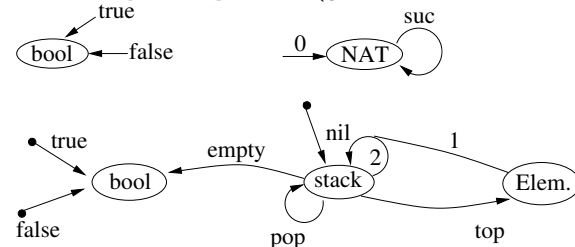
Also $t[p \leftarrow r]_q = t_q$ für $q \mid p$ und



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

Signaturen

Darstellung von Signaturen (graphisch oder normiert)



Notationen:

- $\text{sig} \dots$
- $\text{sorts} \dots$
- $\text{ops} \dots$
- $\text{op} : W \rightarrow S$
- $\text{op}_1, \dots, \text{op}_i : W \rightarrow S$

Kanonische Homomorphismen

Lemma 6.9. \mathcal{A} sig-Algebra, T_{sig}

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

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

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

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

Beweis: Übung macht den Meister!

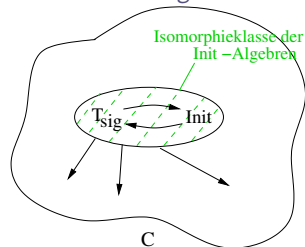
Initiale Algebren

Definition 6.10. *Initiale Algebren:*

Eine sig-Algebra \mathcal{A} heißt *initial in einer Klasse C* von sig-Algebren, wenn für jede sig-Algebra $\mathcal{A}' \in C$ genau ein sig-Homomorphismus $h : \mathcal{A} \rightarrow \mathcal{A}'$ 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

\mathcal{A} sig-Algebra, $h : T_{sig} \rightarrow \mathcal{A}$ Interpretationshomomorphismus.

\mathcal{A} sig-erzeugt (Term-erzeugt) gdw

$\forall s \in S \quad h_s : \text{Term}_s(F) \rightarrow A_s$ 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 \mathcal{A} lassen sich eindeutig zu Homomorphismen von $T_{sig}(V)$ in \mathcal{A} fortsetzen.

Gleichungsspezifikationen

Für Spezifikationsformalismen:

Klassen von Algebren die initialen Algebren zulassen.

\rightsquigarrow Horn (Siehe Literatur)

```
sig INT      sorts int
ops  0 :→ int
     suc : int → int
     pred : int → int
```

Gleichungsspezifikation

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

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

Schreibe: $u = v$

Gleichungssystem E über sig, V : Menge von Gleichungen

b) (Gleichungs)-Spezifikation: $\text{spec} = (\text{sig}, E)$

E Gleichungssystem über $F \cup V$.

Notation

Schlüsselwort eqns

spec INT

sorts int

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

suc, pred: $\text{int} \rightarrow \text{int}$

eqns suc(pred(x)) = x

pred(suc(x)) = x

implizite

All-Quantifikation

oft Deklaration

der Sorten

der Variablen

Semantik::

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

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

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

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

Bewertung $\varphi : \text{Term}(F, V) \rightarrow \mathfrak{A}$ durch

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

$\varphi(f(t_1, \dots, t_n)) = f_{\mathfrak{A}}(\varphi(t_1), \dots, \varphi(t_n))$

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

Beweis!

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

b) $s = t$ Gleichung über sig, V

$\mathfrak{A} \models s = t$: \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 $\mathfrak{A} \models s = t$

d) \mathfrak{A} ist Modell von $\text{spec} = (\text{sig}, E)$

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 :→ nat
     s : nat → nat
     _ + _ : nat, nat → 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) $\mathfrak{C} = (\{\text{true}, \text{false}\}, \hat{0}, \hat{+}, \hat{s})$
 $\hat{0} = \text{false} \quad \hat{s}(\text{true}) = \text{false} \quad \hat{s}(\text{false}) = \text{true}$
 $i \hat{+} j = i \vee j$

Beispiele

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

z.B. $\mathfrak{B} : \varphi(x) = a \quad \varphi(y) = b \quad a, b \in \mathbb{Z}$
 $\varphi(x + 0) = a \hat{+} \hat{0} = a \cdot 1 = a = \varphi(x)$
 $\varphi(x + s(y)) = a \hat{+} \hat{s}(b) = a \cdot (b \cdot 5)$
 $= (a \cdot b) \cdot 5 = \hat{s}(a \hat{+} b)$
 $= \varphi(s(x + y))$

Beispiele

2)

```
spec LIST(NAT)
use NAT
sorts nat, list
ops nil :→ list
     _ . _ : nat, list → list
     app : list, list → list
eqns app(nil, q2) = q2
     app(x . q1, q2) = x . app(q1, q2)
```

Beispiele

spec-Algebra

$$\begin{aligned}
 \mathfrak{A} & \quad \mathbb{N}, \mathbb{N}^* \\
 \hat{0} &= 0 \quad \hat{+} = + \quad \hat{s} = +1 \\
 \hat{nil} &= e \quad (\text{leeres Wort}) \\
 \hat{\cdot}(i, z) &= i z \\
 \hat{app}(z_1, z_2) &= z_1 z_2 \text{ (Konkatenation)}
 \end{aligned}$$

Beispiele

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

	1	2	3
A_{int}	\mathbb{Z}	\mathbb{N}	$\{\text{true}, \text{false}\}$
$0_{\mathfrak{A}_i}$	0	0	true
$\text{suc}_{\mathfrak{A}_i}$	$\text{suc}_{\mathbb{Z}}$	$\text{suc}_{\mathbb{N}}$	$\begin{cases} \text{true} \rightarrow \text{false} \\ \text{false} \rightarrow \text{true} \end{cases}$
$\text{pred}_{\mathfrak{A}_i}$	$\text{pred}_{\mathbb{Z}}$	$\begin{cases} n+1 \rightarrow n \\ 0 \rightarrow 0 \end{cases}$	$\begin{cases} \text{true} \rightarrow \text{false} \\ \text{false} \rightarrow \text{true} \end{cases}$
	+	-	+

Beispiele

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

Substitution

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

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

Fortsetzung zu Homomorphismus $\sigma : \text{Term}(F, V) \rightarrow \text{Term}(F, V)$

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

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

Lose Semantik

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

Gesucht: Charakterisierungen der Gleichungen die in $ALG(spec)$ bzw. $ALG_{TE}(spec)$ (Term erzeugt).

- a) *Semantische Gleichheit:* $E \models s = t$
- b) *Operationale Gleichheit:* $t_1 \stackrel{E}{\equiv} t_2$ gdw
 Es gibt $p \in O(t_1)$, $s = t \in E$, Substitution σ mit
 $t_1|_p \equiv \sigma(s)$, $t_2 \equiv t_1[\sigma(t)]_p$ ($t_1[p \leftarrow \sigma(t)]$)
 oder
 $t_1|_p \equiv \sigma(t)$, $t_2 \equiv t_1[\sigma(s)]_p$
 $t_1 \stackrel{*}{=} t_2$ gdw $t_1 \stackrel{E}{\equiv} t_2$

Formalisierung von: Ersetze Gleiches \leftrightarrow Gleiches

Gleichheitskalkül

- c) **Gleichheitskalkül:** Inferenzregeln (deduktiv)

Reflexivität $\frac{}{t = t}$

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

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

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

(oft auch mit Substitution σ)

Gleichheitskalkül

$E \vdash s = t$ 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) $t_1 = t_2 \dots$ aus vorangehenden Gleichungen durch Anwendung einer der Inferenzregeln.

Eigenschaften und Beispiele

Folgerung 6.16. *Eigenschaften und Beispiele*

- a) Gilt $E \models s = t$ oder $s \stackrel{E}{=} t$ oder $E \vdash s = t$

- i) Ist σ eine Substitution, so auch

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

- ii) $r \in \text{Term}(F, V)$, $p \in O(r)$, $r|_p, s, t \in \text{Term}_{s'}(F, V)$ so

$E \models r[s]_p = r[t]_p / r[s]_p \stackrel{E}{=} r[t]_p / E \vdash r[s]_p = r[t]_p$
Monotonie Eigenschaft

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

Satz von Birkhoff

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

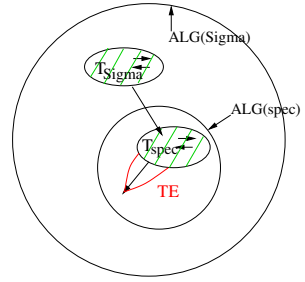
$$E \models s = t \text{ gdw } E \vdash s = t \quad (\text{d. h. } s =_E t)$$

Definition 6.18. Initiale Semantik
 Sei $spec = (sig, E)$ die Algebra $T_{sig}/=_E$ (*Quotiententalgebra*)
 ($=_E$ 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) Quotiententalgebra $T_{sig}/=_E$ zu.
 Schreibe: T_{spec} oder $I(E)$



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

Quotiententalgebren

Quotiententalgebren sind ADT.

Beispiel 7.1. (Fortsetzung) $spec = INT$

A_{int}^i	\mathbb{Z}	$\{true, false\}$	$\{1\}^+ \cup \{0\}^+ \cup \{z\}$
0_{A^i}	0	true	z
suc_{A^i}	$suc_{\mathbb{Z}}$	not	...
$pred_{A^i}$	$pred_{\mathbb{Z}}$	not	...

$$T_{INT}/=E \quad [0] \mapsto true \quad [suc^{2n}(0)] \mapsto true$$

$$[suc^{2n+1}(0)] \mapsto false \quad [pred^{2n+1}(0)] \mapsto false$$

$$[pred^{2n}(0)] \mapsto true$$

Initiale Algebra

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

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

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

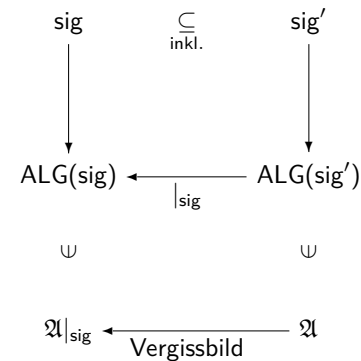
Definition 7.7. *Einschränkungen/Vergißbilder*

- 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: $\mathfrak{A}|_{sig}$ ist sig - Algebra.
 $\mathfrak{A}|_{sig}$ heißt auch *Vergißbild* von \mathfrak{A} (bzgl. sig).

Einschränkungen/Vergißbilder

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



Einschränkungen/Vergißbilder

- b) Eine Spezifikation $spec = (sig', E)$ mit $sig \subseteq sig'$ ist *korrekt bzgl. sig-Algebra \mathfrak{A}* gdw
- $$(T_{spec})|_{sig} \cong \mathfrak{A}$$
- (Der Hom. von $(T_{spec})|_{sig}$ nach \mathfrak{A} ist Bijektion).
 c) Eine Spezifikation $spec' = (sig', E')$ *implementiert eine Spezifikation $spec = (sig, E)$* gdw
- $$sig \subseteq sig' \text{ und } (T_{spec'})|_{sig} \cong T_{spec}$$

Beachte:

- ▶ Konsistenz-Begriff für =-Spezifikation nicht notwendig. (Modelle existieren immer!).
- ▶ Allgemeiner Implementierungsbegriff $(CI(spec) \subseteq CI(spec'))$ reduziert sich immer auf =.
 "vollständige,, Theorien.

Parametrisierung

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

D. h. $\text{Formal}=(\text{sig}_F, E_F)$, $\text{Body}=(\text{sig}_B, E_B)$, wobei $\text{sig}_F \subseteq \text{sig}_B$ $E_F \subseteq E_B$.

Notation: $\text{Body}[\text{Formal}]$

Syntaktisch: $\text{Body} = \text{Formal} + (\text{sig}', E')$ 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 $(T_{\text{spec}})_{\text{elem}} = \emptyset$
 sorts elem
 ops next : elem \rightarrow elem

spec STRING[ELEM] $(T_{\text{spec}})_{\text{string}} = \{\{\text{empty}\}\}$
 use ELEM
 sorts string
 ops empty : \rightarrow string
unit : elem \rightarrow string
concat : string, string \rightarrow string
ladd : elem, string \rightarrow string
radd : string, elem \rightarrow string

Beispiel (Forts.)

eqns concat(s, empty) = s
concat(empty, s) = s
concat(concat(s₁, s₂), s₃) = concat(s₁, concat(s₂, s₃))
ladd(e, s) = concat(unit(e), s)
radd(s, e) = concat(s, unit(e))

Parameterübergabe: ELEM \rightarrow NAT

STRING[ELEM] \rightarrow STRING[NAT]

Zuordnung: formale Parameter \rightarrow aktuelle Parameter

$S_F \rightarrow S_A$
 $Op \rightarrow Op_A$

Zuordnung der Sorten und Funktionen, Semantik?

Signaturmorphismen - Parameterübergabe

Definition 7.13.

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

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

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

im Bsp. $g : \text{elem} \rightarrow \text{nat}$ $h :: \text{next} \rightarrow \text{suc}$
oder $\sigma : \text{sig}_{\text{BOOL}} \rightarrow \text{sig}_{\text{NAT}}$ mit
 $g :: \text{bool} \rightarrow \text{nat}$ $h :: \text{not} \rightarrow \text{suc}$
 $h :: \text{true} \rightarrow 0$ $\text{and} \rightarrow \text{plus}$
 $\text{false} \rightarrow 0$ $\text{or} \rightarrow \text{times}$

Signaturmorphisimen - Parameterübergabe

- b) spec = Body[Formal] Parameterspezifikation
 Eine **Parameterübergabe** ist ein Signaturmorphismus
 $\sigma : sig(\text{Formal}) \rightarrow sig(\text{Actual})$ 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 $op : s_1 \dots s_n \rightarrow s_0 \in \text{Body}$, das nicht in Formal auftritt jedes $s_i, s_i \in \text{Formal}$ durch $\sigma(s_i)$.
 - 3) Ersetze in jeder Gleichung $L = R$ aus Body, die nicht in Formal ist, jedes $op \in \text{Formal}$ durch $\sigma(op)$.
 - 4) Fasse jede Variable einer Sorte s mit $s \in \text{Formal}$ als Variable der Sorte $\sigma(s)$ auf.
 - 5) Vermeide Namenskonflikte zwischen Actual und Body/Formal.

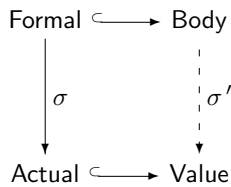
Parameterübergabe

Bezeichnung:

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

Man erhält somit für $\sigma : sig(\text{Formal}) \rightarrow sig(\text{Actual})$ einen Signaturmorphismus

$$\sigma' : sig(\text{Body}[\text{Formal}]) \rightarrow sig(\text{Body}[\text{Actual}, \sigma])$$



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

x' ist dabei eine **Umbenennung**, wenn Konflikte.

Signaturmorphisimen (Forts.)

Definition 7.14. Sei $\sigma : sig' \rightarrow sig$ Signaturmorphismus.

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

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

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

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

Beispiel

Beispiel 7.15. $\mathfrak{A} = T_{\text{NAT}}$ (Mit 0, suc, plus, times)

$$sig' = sig(\text{BOOL}) \quad sig = sig(\text{NAT})$$

$\sigma : sig' \rightarrow sig$ wie oben definiert.

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

$$\begin{aligned} \text{true}_{(T_{\text{NAT}})|_\sigma} &= \sigma(\text{true})_{T_{\text{NAT}}} = [0] \\ \text{false}_{(T_{\text{NAT}})|_\sigma} &= \sigma(\text{false})_{T_{\text{NAT}}} = [0] \\ \text{not}_{(T_{\text{NAT}})|_\sigma} &= \sigma(\text{not})_{T_{\text{NAT}}} = \text{suc}_{T_{\text{NAT}}} \\ \text{and}_{(T_{\text{NAT}})|_\sigma} &= \sigma(\text{and})_{T_{\text{NAT}}} = \text{plus}_{T_{\text{NAT}}} \\ \text{or}_{(T_{\text{NAT}})|_\sigma} &= \sigma(\text{or})_{T_{\text{NAT}}} = \text{times}_{T_{\text{NAT}}} \end{aligned}$$

Vergiβbilder für Homomorphismen

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

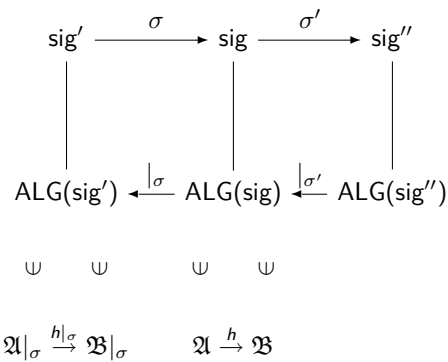
$h|_\sigma := \{h_{\sigma(s)} \mid s \in S'\}$, wobei $sig' = (S', F', \tau')$, ein sig'-Homomorphismus:

$$(h|_\sigma)_s = h_{\sigma(s)} : \begin{matrix} A_{\sigma(s)} & \rightarrow & B_{\sigma(s)} \\ \parallel & & \parallel \\ (A|_\sigma)_s & \rightarrow & (B|_\sigma)_s \end{matrix}$$

$h|_\sigma$ heiβt Vergiβbild von h entlang σ

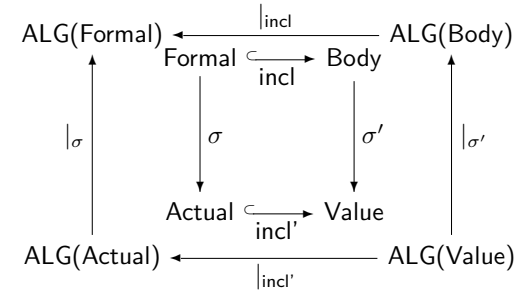
Vergiβbilder

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



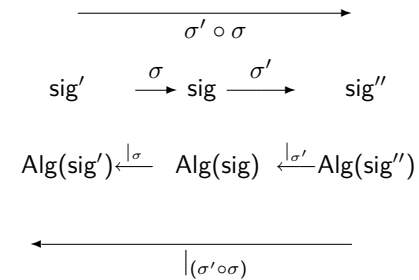
Verträglichkeit mit Identität, Komposition und Homomorphismen.

Parameterspezifikation

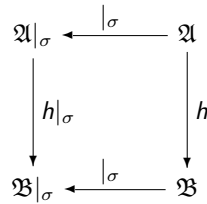


Parameterspezifikation

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



Parameterspezifikation



Semantik der Parameterübergabe (nur Signatur)

Definition 7.17. Sei $Body[Formal]$ Parameterspezifikation.
 $\sigma : Formal \rightarrow Actual$ Signaturmorphismus.
 Semantik der Parameterübergabe $[Actual, \sigma]$.
 Zuordnung: $\sigma : Formal \rightarrow Actual$
 \downarrow
 initiale Semantik von Value. D. h.
 $T_{Body[Actual, \sigma]}$
 Betrachte: $S :: (T_{Actual}, \sigma) \mapsto T_{Body[Actual, \sigma]}$
 Abbildung zwischen init Algebren. Kann aufgefasst werden als Zuordnung
 zwischen Formal Algebren \rightarrow Body-Algebren.

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

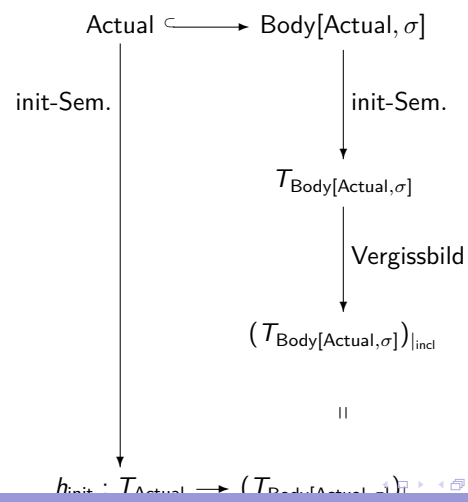


Abbildung zwischen init Algebren

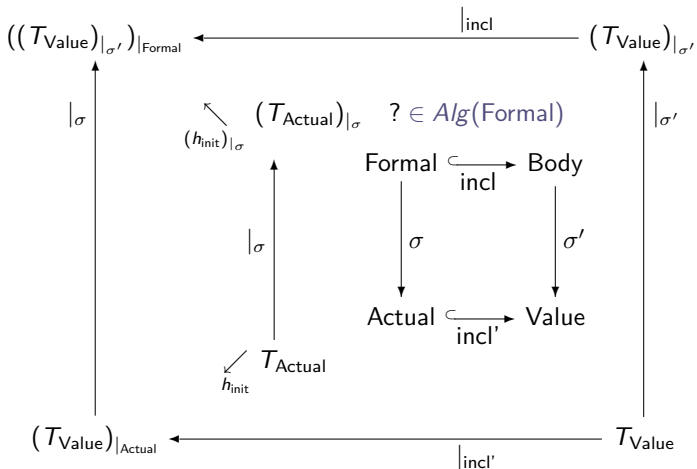


Abbildung zwischen init Algebren

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

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

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

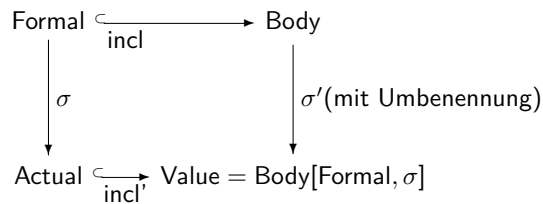
$a|_{\mathfrak{A}|_{\sigma}} = 0 \neq 1 = b|_{\mathfrak{A}|_{\sigma}}$

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

Parameterübergabe (Actual, σ)

Body[Formal]

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



Vor: $\text{sig}(\text{Actual})$ und $\text{sig}(\text{Value})$ strikt.

Parameterübergabe (Actual, σ)

Vergißbilder: $|_{\sigma} : \text{Alg}(\text{sig}) \rightarrow \text{Alg}(\text{sig}')$

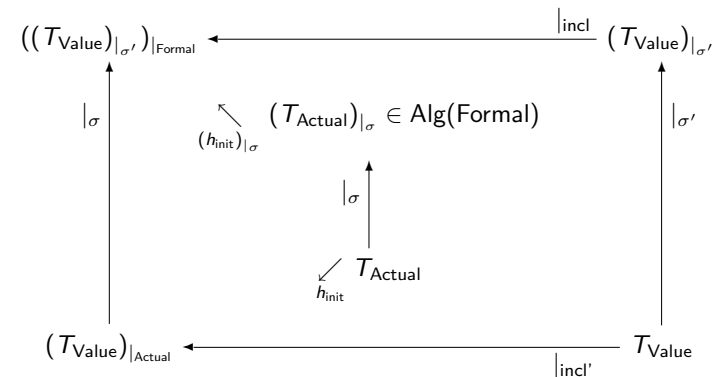
$\mathfrak{A}|_{\sigma}$ für $\sigma : \text{sig}' \rightarrow \text{sig}$

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

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

sig'-Homomorphismus

Parameterübergabe (Actual, σ)



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

Spezifikationsmorphismus

Definition 7.18. Seien $spec' = (sig', E')$, $spec = (sig, E)$ (allg.)

Spezifikationen.

Ein Signaturmorphismus $\sigma : sig' \rightarrow sig$ heißt **Spezifikationsmorphismus**, falls für alle $s = t \in E'$ gilt $\sigma(s) = \sigma(t) \in Th(E)$.

Schreibe: $\sigma : spec' \rightarrow spec$

Fakt: Für $\mathfrak{A} \in Alg(spec)$ gilt $\mathfrak{A}|_{\sigma} \in Alg(spec')$

D. h. $|_{\sigma} : Alg(spec) \rightarrow Alg(spec')$!

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 $\sigma : Formal \rightarrow Actual$ Spezifikationsmorphismus.

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

- verlange auch h_{init} Bijektion.

Syntaktische Einschränkungen die dies garantieren.

Spezifikationssprachen

CLEAR, Act-one, -Cip-C, Affirm, ASL, Aspik, OBJ, ASF, \rightsquigarrow neuere
+

Sprachen: - Spectrum, - Troll, -Maude

Beispiel

Beispiel 7.19.

Formal :: {

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

}

Beispiel (Forts.)

- spec LIST[ELEMENT]
- use ELEMENT
- sorts list
- ops $nil : \rightarrow list$
 $. : elem, list \rightarrow list$
 $insert : elem, list \rightarrow list$
 $case : bool, list, list \rightarrow list$
 $sorted : list \rightarrow bool$

Beispiel (Forts.)

```

eqns case(true, l1, l2) = l1
      case(false, l1, l2) = l2

insert(x, nil) = x.nil
insert(x, y.l) = case(x ≤ y, insert(x, insert(y, l)), insert(y, insert(x, l)))

sorted(nil) = true
sorted(x.nil) = true
sorted(x.y.l) = if x ≤ y then sorted(y, l) else false
  
```

Eigenschaft: sorted(insert(x, l)) = true

Beispiel (Forts.)

```

ACTUAL ≡ BOOL
σ : elem → bool, bool → bool
    . ≤ . → impl
  
```

Die Gleichungen von ELEMENT sind in $Th(BOOL)$

↔ Spezifikationsmorphismus

Beispiel (Forts.)

```

ACTUAL ≡ NAT
σ : bool → nat      elem → nat
      true → suc(0)  nicht erlaubt
      false → 0
      not → suc
      or → plus
      and → times
      . ≤ . → ...
  
```

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

Abstrakte Reduktionssysteme: Begriffswelt

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

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

Wichtige Zusammenhänge

Lemma 8.5. \rightarrow konfluent gdw \rightarrow Church-Rosser.

Satz 8.6. (Newmann Lemma) Sei \rightarrow noethersch, dann \rightarrow konfluent gdw \rightarrow lokal konfluent.

Folgerung 8.7.

- a) \rightarrow konfluent und $x \xrightarrow{*} y$.
 - i) Ist y irreduzibel, so $x \xrightarrow{*} y$.
 Insbesondere, wenn x, y irreduzibel, so $x = y$.
 - ii) $x \xrightarrow{*} y$ gdw $\Delta^*(x) \cap \Delta^*(y) \neq \emptyset$.
 - iii) Hat x eine NF, so ist sie eindeutig.
 - iv) Ist \rightarrow noethersch, so hat jedes $x \in U$ genau eine NF:
 Schreibweise $x \downarrow$
- b) (U, \rightarrow) hat jedes $x \in U$ genau eine NF, so ist \rightarrow konfluent, i. Allg. nicht noethersch.

Konvergente Reduktionssysteme

Definition 8.8. (U, \rightarrow) konvergent gdw \rightarrow noethersch und konfluent.

Wichtig da: $x \xrightarrow{*} y$ gdw $x \downarrow = y \downarrow$

\rightarrow effektiv so Entscheidungsverfahren für WP:

Für Programmierung: $x \xrightarrow{*} x \downarrow, f(t_1, \dots, t_n) \xrightarrow{*}$ „Wert“

i. Allg. Unentscheidbare Eigenschaften

Terminierung und Konfluenz

Hinreichende Bedingungen/Techniken

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

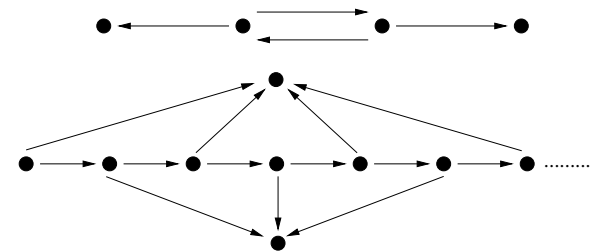
Beispiel 8.10. Oft $(\mathbb{N}, >), (\Sigma^*, >)$
 $w \in \Sigma^*, |w|$ Länge, $|w|_a$ a-Länge $a \in \Sigma$.
 WF-Partialordnungen auf Σ^*

- ▶ $x > y$ 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!

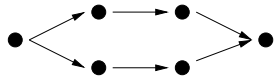


Abschwächung der Terminierung

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

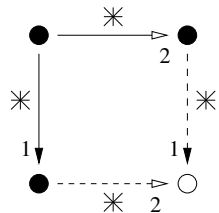
▷ einseitige Lokalisierung der Konfluenz ◁

Satz 8.12. Ist \rightarrow streng konfluent, so ist \rightarrow konfluent.

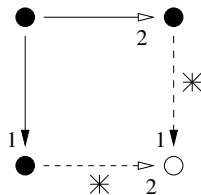


Zusammenhang von Relationen

Definition 8.13. Zwei Relationen $\rightarrow_1, \rightarrow_2$ auf U kommutieren, falls gilt $1^* \circ \rightarrow_2 \subseteq \rightarrow_2 \circ 1^*$.
 Sie **kommutieren lokal** falls gilt $1 \leftarrow \circ \rightarrow_2 \subseteq \rightarrow_2 \circ 1^* \leftarrow$.



kommutierend



lokal kommutierend

Zusammenhang von Relationen

Lemma 8.14. Sei $\rightarrow = \rightarrow_1 \cup \rightarrow_2$

- (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) $x + 0 = x, \quad x + s(y) = s(x + y)$
- (b) $x + y = y + x, \quad (x + y) + z = x + (y + z)$

▷ *Problem: permutative Regeln (b)* ◁

Nicht-Orientierbarkeit

Definition 8.15. (U, \rightarrow, \vdash) mit \rightarrow Relation, \vdash symmetrische Relation.
 Sei $\vdash = \leftrightarrow \cup \vdash, \quad \sim = \vdash^*, \quad \approx = \vdash^*$,
 $\rightarrow_{\sim} = \sim \circ \rightarrow \circ \sim, \quad \downarrow_{\sim} = \rightarrow^* \circ \sim \circ \rightarrow^*$.

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

- \rightarrow heißt **Church-Rosser modulo \sim** : gdw $\approx \subseteq \downarrow_{\sim}$
- \rightarrow heißt **lokal konfluent modulo \sim** : gdw $\leftarrow \circ \rightarrow \subseteq \downarrow_{\sim}$
- \rightarrow heißt **lokal kohärent modulo \sim** : gdw $\leftarrow \circ \vdash \subseteq \downarrow_{\sim}$

Nicht-Orientierbarkeit- Reduktion Modulo \vdash

Satz 8.16. Sei \rightarrow_{\sim} terminierend. Dann ist \rightarrow 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) $\xrightarrow{*} = \sim$ mit $\sim = \vdash^*$

Idee: Approximation von \rightarrow durch Transformationen
 $(\vdash, \emptyset) = (\vdash_0, \rightarrow_0) \vdash (\vdash_1, \rightarrow_1) \vdash (\vdash_2, \rightarrow_2) \vdash \dots$

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) $u \rightarrow_{i+1} v$, falls $u > v$ und $u \vdash_i v$
- (2) $u \vdash_{i+1} v$, falls $u \leftarrow w \rightarrow_i v$
- (3) verkleinere $u \vdash_i v$ zu $u \vdash_{i+1} w$, falls $v \rightarrow_i w$

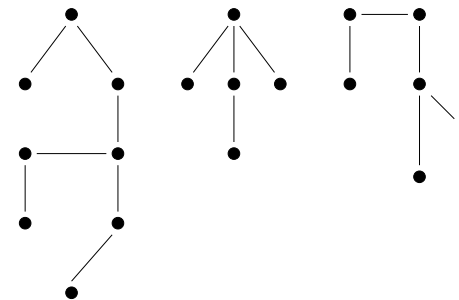
Ziel: Grenzsysteem

$$\rightarrow = \rightarrow_{\infty} = \bigcup \{ \rightarrow_i \mid i \in \mathbb{N} \} \text{ mit } \vdash_{\infty} = \emptyset$$

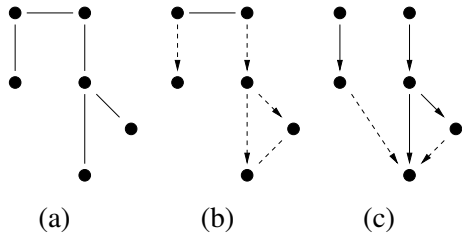
Also:

- $\rightarrow_{\infty} \subseteq >$, d. h. noethersch
- $\xrightarrow{*} = \sim$
- \rightarrow_{∞} 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**

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

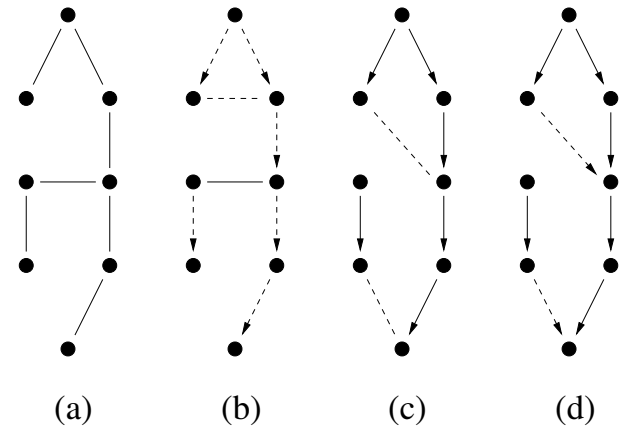
$(\vdash, \rightarrow) \vdash_{\mathcal{P}} (\vdash', \rightarrow')$ falls (\vdash, \rightarrow) mit \mathcal{P} in einem Schritt in (\vdash', \rightarrow') überführt werden kann.

$\vdash_{\mathcal{P}}^*$ 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

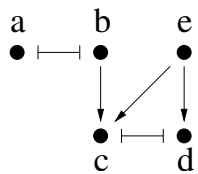
- Lemma 8.18.** Sei $(\vdash, \rightarrow) \vdash_{\mathcal{P}} (\vdash', \rightarrow')$
- (a) Ist $\rightarrow \subseteq >$, so gilt auch $\rightarrow' \subseteq >$
 - (b) Es gilt $(\vdash \cup \leftrightarrow)^* = (\vdash' \cup \leftrightarrow')^*$

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 $\xrightarrow{*} \circ \xleftarrow{*}$ Beweise in der Ordnung minimal sein.

Eigenschaften des Inferenzsystems

Definition 8.19. Sei (\vdash, \rightarrow) gegeben und $>$ eine Noethersche PO auf U . Sei weiter $(\vdash \cup \leftrightarrow)^* = \sim$.
 Ein **Beweis** für $u \sim v$ ist eine Folge $u_0 *_{i_1} u_1 *_{i_2} \dots *_{i_n} u_n$ mit $*_{i_j} \in \{\vdash, \leftarrow, \rightarrow\}$, $u_i \in U$, $u_0 = u$, $u_n = v$ 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 $a \sim e$:
 $P_1(a, e) = a \vdash b \rightarrow c \vdash d \leftarrow e$
 $P_2(a, e) = a \vdash b \rightarrow c \leftarrow e$

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
 $A \in Mult(U) : \text{gdw } A : U \rightarrow \mathbb{N}$ 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

$$c(u * v) = \begin{cases} \{u\} & \text{falls } u \rightarrow v \\ \{v\} & \text{falls } u \leftarrow v \\ \{u, v\} & \text{falls } u \vdash v \end{cases}$$

Erweitere diese Komplexität auf „zusammengesetzte“ Beweise durch

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

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

Definiere Ordnung auf Beweisen durch
 $P >_P Q$: gdw $c(P) \gg c(Q)$

Konstruktion der Beweisordnung

Es gilt: $>_P$ ist Noethersche Beweisordnung!

Welche Beweisschritte sind groß bzw. klein

Betrachte dazu:

(a) $P_1 = x \leftarrow u \rightarrow y, P_2 = x \vdash y$
 $c(P_1) = \{\{u\}, \{u\}\} \gg \gg \{\{x, y\}\} = c(P_2)$
 $\rightsquigarrow P_1 >_P P_2$

analog

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

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

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

Faire Ableitungen in \mathcal{P}

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

$$\vdash^\infty = \bigcup_{i \geq 0} \bigcap_{j \geq i} \vdash_j \text{ und } \rightarrow^\infty = \bigcup_{i \geq 0} \rightarrow_i.$$

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

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

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

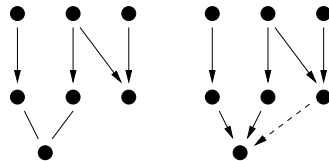
(a) Zu jedem Beweis P in $(\vdash_i, \rightarrow_i)$ gibt es einen äquivalenten Beweis P' in $(\vdash_{i+1}, \rightarrow_{i+1})$ mit $P \geq_P P'$.

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

Hauptergebniss

Satz 8.25. Sei $(\vdash_i, \rightarrow_i)_{i \in \mathbb{N}}$ eine faire \mathcal{P} -Ableitung und $\rightarrow = \rightarrow_\infty$

- (a) Gilt $u \sim v$, so existiert ein $i \in \mathbb{N}$ mit $u \xrightarrow{*}_i \circ \xrightarrow{*}_i v$
- (b) \rightarrow ist konvergent, und es gilt $\xleftrightarrow{*} = \sim$



Terersetzungssysteme

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
 $rep : Term(sig) \rightarrow Term(sig)$ gibt, mit $rep(t) \in [t]$ eindeutiger
 Repräsentant in seiner Äquivalenzklasse.

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

$$f(x_1, \dots, x_n) : ((T_{spec})_{s_1} \times \dots \times (T_{spec})_{s_n}) \rightarrow (T_{spec})_s$$

$$f([r_1], \dots, [r_n]) := [rep(f(rep(r_1), \dots, (rep(r_n)))]$$

Terersetzungssysteme

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: $l \rightarrow r$
- ▶ Ein **Regelsystem** R ist eine Menge von Regeln.
 R definiert eine **Reduktionsrelation** \rightarrow_R auf $Term(F, V)$ durch:
 $t_1 \rightarrow_R t_2$ gdw $\exists l \rightarrow r \in R, p \in O(t_1), \sigma$ Substitution :
 $t_1|_p = \sigma(l) \wedge t_2 = t_1[\sigma(r)]_p$
- ▶ $(Term(F, V), \rightarrow_R)$ sei das von R definierte **Reduktionssystem** (**Terersetzungssystem**).

Terersetzungssysteme

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

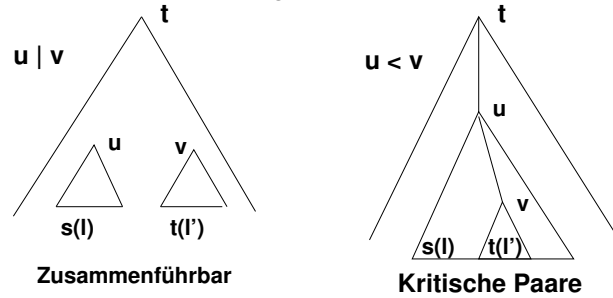
Beachte:

- ▶ 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 Terersetzung. D.h. Aus $s \rightarrow_R t$ folgt, $\sigma(s) \rightarrow_R \sigma(t)$ und $u[s]_p \rightarrow_R u[t]_p$
- ▶ Insbesondere $=_R = \xrightarrow{*}_R$

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 $t_1 \downarrow t_2$.

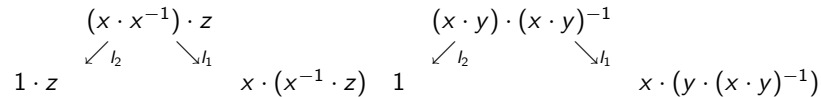


Kritische Paare

Betrachte Gruppenaxiome:

$$\underbrace{(x' \cdot y')}_{h_1} \cdot z \rightarrow x' \cdot (y' \cdot z) \text{ und } x \cdot \underbrace{x^{-1}}_{h_2} \rightarrow 1.$$

“Überlappungen” (Superpositionen)



- ▶ $h_1|_1$ ist “unifizierbar” mit h_2 mit Substitution $\sigma :: \{x' \leftarrow x, y' \leftarrow x^{-1}, x \leftarrow x\} \rightsquigarrow \sigma(h_1|_1) = \sigma(h_2)$
- ▶ h_1 “unifizierbar” mit h_2 mit Substitution $\sigma :: \{x' \leftarrow x, y' \leftarrow y, z \leftarrow (x \cdot y)^{-1}, x \leftarrow x \cdot y\} \rightsquigarrow \sigma(h_1) = \sigma(h_2)$

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 $(t \preceq s \wedge \neg(s \preceq t))$
 \preceq ist noethersche Partialordnung auf $Term(F, V)$ *Beweis!*

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

Verträglichkeitseigenschaften:

- $t|_u = t' \rightsquigarrow \sigma(t)|_u = \sigma(t')$
- $t|_u = x \in V \rightsquigarrow \sigma(t)|_{uv} = \sigma(x)|_v$ ($v \in O(\sigma(x))$)
- $\sigma(t)[\sigma(t')]_u = \sigma(t[t']_u)$ 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, Allgemeinsten Unifikator

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

- ▶ $\sigma \preceq \tau$ (V') gdw $\exists \rho$ Substitution : $\rho \circ \sigma|_{V'} = \tau|_{V'}$
Sprechweise: σ ist *allgemeiner* als τ auf V'
- ▶ $\sigma \approx \tau$ (V') gdw $\sigma \preceq \tau(V') \wedge \tau \preceq \sigma(V')$
- ▶ $\sigma \prec \tau$ (V') gdw $\tau \preceq \sigma$ (V') $\wedge \neg(\sigma \preceq \tau$ (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 $E = \{s_i \stackrel{?}{=} t_i : i = 1, \dots, n\}$ von Gleichungen.
- ▶ σ heißt **Lösung** (oder **Unifikator**) falls $\sigma(s_i) = \sigma(t_i)$ für $i = 1, \dots, n$.
- ▶ Gilt $\tau \succeq \sigma(\text{Var}(E))$ für jede Lösung τ von E , so $\text{mgu}(E) := \sigma$ **allgemeinste Lösung** oder **allgemeinster Unifikator**.
- ▶ $\text{Sol}(E)$ sei die Menge der Lösungen von E . E und E' sind **äquivalent**, falls $\text{Sol}(E) = \text{Sol}(E')$.
- ▶ E' ist in **gelöster Form**, falls $E' = \{x_j \stackrel{?}{=} t_j : x_i \neq x_j \ (i \neq j), x_i \notin \text{Var}(t_j) \ (1 \leq i \leq j \leq m)\}$
- ▶ E' ist **gelöste Form** zu E , falls E' in gelöster Form und äquivalent zu E mit $\text{Var}(E') \subseteq \text{Var}(E)$.

Beispiele

Beispiel 9.10. Betrachte

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

Inferenzsystem für die Unifikation

Definition 9.11. Kalkül UNIFY. Sei $\sigma = \text{Bindungsmenge}$.

- (1) **Löschen** $\frac{(E \cup \{s \stackrel{?}{=} s\}, \sigma)}{(E, \sigma)}$
- (2) **Split (Zerlegen)** $\frac{(E \cup \{f(s_1, \dots, s_m) \stackrel{?}{=} g(t_1, \dots, t_n)\}, \sigma)}{\frac{1}{2} \text{ (Unlösbar)}} \text{ falls } f \neq g$
 $\frac{(E \cup \{f(s_1, \dots, s_m) \stackrel{?}{=} f(t_1, \dots, t_m)\}, \sigma)}{(E \cup \{s_i \stackrel{?}{=} t_i : i = 1, \dots, m\}, \sigma)}$
- (3) **Merge (Lösen)** $\frac{(E \cup \{x = t\}, \sigma)}{(\tau(E), \sigma \cup \tau)} \text{ falls } x \notin \text{Var}(t), \tau = \{x \stackrel{?}{=} t\}$
 "occur check" $\frac{(E \cup \{x = t\}, \sigma)}{\frac{1}{2} \text{ (Unlösbar)}} \text{ falls } x \in \text{Var}(t) \wedge x \neq t$

Unifikationsalgorithmen

Unifikationsalgorithmen basierend auf diesem Kalkül starten stets mit $(E_0, S_0) := (E, \emptyset)$ liefern eine Folge $(E_0, S_0) \vdash_{UNIFY} \dots \vdash_{UNIFY} (E_n, S_n)$ Sie sind **erfolgreich** falls sie mit $E_n = \emptyset$, **unerfolgreich** falls sie mit $S_n = \frac{1}{2}$ enden. S_n beschreibt eine Substitution σ die $\text{Sol}(S_n)$ und somit auch $\text{Sol}(E)$ erfasst.

Lemma 9.12. Korrektheit. Jede Folge $(E_0, S_0) \vdash_{UNIFY} \dots \vdash_{UNIFY} (E_n, S_n)$ bricht ab: Entweder mit $\frac{1}{2}$ (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!!):
 $s \stackrel{?}{=} f(x_1, \dots, x_n) \quad t \stackrel{?}{=} f(g(x_0, x_0), \dots, g(x_{n-1}, x_{n-1}))$
 $S = \{x_i \stackrel{?}{=} g(x_{i-1}, x_{i-1}) : i = 1, \dots, n\}$ und
 $S_1 = \{x_{i+1} \stackrel{?}{=} t_i : t_0 = g(x_0, x_0), t_{i+1} = g(t_i, t_i) \ i = 0, \dots, n-1\}$
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 $l_1 \rightarrow r_1, l_2 \rightarrow r_2 \in R$ mit $V(l_1) \cap V(l_2) = \emptyset$ (Umbenennung von Variablen falls nötig, u.U. $l_1 \approx l_2$ bzw. $l_1 \rightarrow r_1 \approx l_2 \rightarrow r_2$).

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 $l_2 \rightarrow r_2$ in $l_1 \rightarrow r_1$ und $(\sigma(r_1), \sigma(l_1[r_2]_u))$ ist das zugehörige **kritische Paar** zu $l_1 \rightarrow r_1, l_2 \rightarrow r_2, u \in O(l_1)$, sofern $\sigma(r_1) \neq \sigma(l_1[r_2]_u)$.

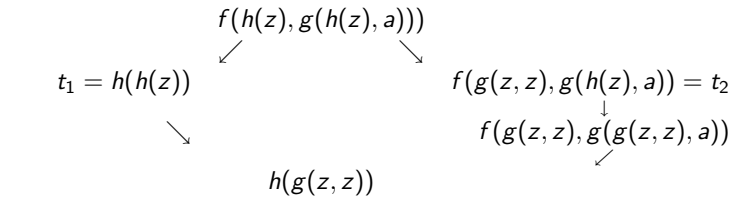
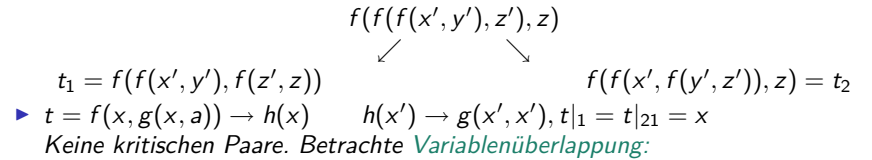
$CP(R)$ sei die Menge aller kritischen Paare die man mit Regeln aus R bilden kann.

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

Beispiele

Beispiel 9.15. Betrachte

► $f(f(x, y), z) \rightarrow f(x, f(y, z)) \quad f(f(x', y'), z') \rightarrow f(x', f(y', z'))$
Unifizierbar mit $x \leftarrow f(x', y'), y \leftarrow z'$



Eigenschaften

► Seien σ, τ Substitutionen, $x \in V, \sigma(y) = \tau(y)$ für $y \neq x$ und $\sigma(x) \rightarrow_R \tau(x)$. Dann gilt für jeden Term t :

$$\sigma(t) \xrightarrow{*}_R \tau(t)$$

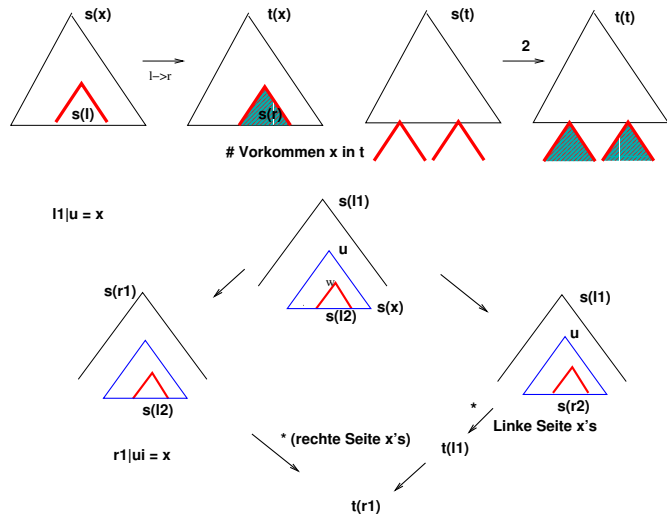
► Seien $l_1 \rightarrow r_1, l_2 \rightarrow r_2$ Regeln, $u \in O(l_1), l_1|_u = x \in V$. Sei $\sigma(x)|_w = \sigma(l_2)$, d.h. $\sigma(l_2)$ wird durch $\sigma(x)$ eingeführt. Dann gilt $t_1 \downarrow_R t_2$ für

$$t_1 := \sigma(r_1) \leftarrow \sigma(l_1) \rightarrow \sigma(l_1)[\sigma(r_2)]_{uw} =: t_2$$

Lemma 9.16. Critical-Pair Lemma von Knuth/Bendix
Sei R ein Regelsystem. Dann gilt:

Aus $t_1 \leftarrow_R t \rightarrow_R t_2$ folgt $t_1 \downarrow_R t_2$ oder $t_1 \leftrightarrow_{CP(R)} t_2$.

Beweise



Konfluenztest

Satz 9.17. Hauptergebnis: Sei R ein Regelsystem.

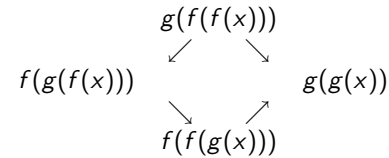
- ▶ R ist genau dann lokal konfluent, wenn alle Paare $(t_1, t_2) \in CP(R)$ zusammenführbar sind.
- ▶ Ist R terminierend, so gilt:
 R konfluent gdw. $(t_1, t_2) \in CP(R) \rightsquigarrow t_1 \downarrow t_2$.
- ▶ Sei R linear (d.h. für $l, r \in l \rightarrow r \in R$ kommen Variablen höchstens einmal vor). Gilt $CP(R) = \emptyset$, so ist R konfluent.

Beispiel 9.18.

- ▶ Sei $R = \{f(x, x) \rightarrow a, f(x, s(x)) \rightarrow b, a \rightarrow s(a)\}$.
 R ist lokal konfluent, aber nicht konfluent:
 $a \leftarrow f(a, a) \rightarrow f(a, s(a)) \rightarrow b$
 aber nicht $a \downarrow b$. R ist weder terminierend noch links-linear.

Beispiel (Fort.)

- ▶ $R = \{f(f(x)) \rightarrow g(x)\}$
 $t_1 = g(f(x)) \leftarrow f(f(f(x))) \rightarrow f(g(x)) = t_2$
 Es gilt nicht, $t_1 \downarrow_R t_2 \rightsquigarrow R$ nicht konfluent.
 Füge Regel $t_1 \rightarrow t_2$ zu R hinzu. R_1 ist äquivalent zu R , terminierend und konfluent.



- ▶ $R = \{x + 0 \rightarrow x, x + s(y) \rightarrow s(x + y)\}$, linear ohne krit. Paare \rightsquigarrow konfluent.
- ▶ $R = \{f(x) \rightarrow a, f(x) \rightarrow g(f(x)), g(f(x)) \rightarrow f(h(x)), g(f(x)) \rightarrow b\}$
 ist lokal konfluent aber nicht konfluent.

Konfluenz ohne Terminierung

Definition 9.19. $\epsilon - \epsilon$ -Eigenschaften. Sei $\overset{\epsilon}{\rightarrow} = \overset{0}{\rightarrow} \cup \overset{1}{\rightarrow}$.

- ▶ R heißt $\epsilon - \epsilon$ abgeschlossen, falls für jedes kritische Paar $(t_1, t_2) \in CP(R)$ es ein t gibt mit $t_1 \overset{\epsilon}{\rightarrow}_R t \overset{\epsilon}{\leftarrow}_R t_2$.
- ▶ R heißt $\epsilon - \epsilon$ konfluent gdw. $\overset{\epsilon}{\leftarrow}_R \circ \overset{\epsilon}{\rightarrow}_R \subseteq \overset{\epsilon}{\rightarrow}_R \circ \overset{\epsilon}{\leftarrow}_R$

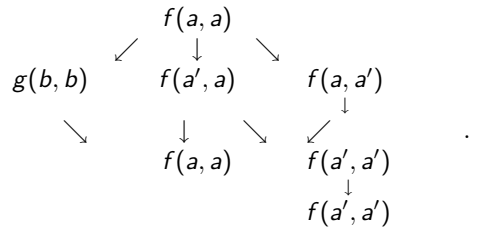
Folgerung 9.20.

- ▶ $\rightarrow \epsilon - \epsilon$ konfluent $\rightsquigarrow \rightarrow$ streng konfluent.
- ▶ $R \epsilon - \epsilon$ abgeschlossen $\nRightarrow R \epsilon - \epsilon$ 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 \nmid .
- ▶ 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:
 $R = \{f(a, a) \rightarrow g(b, b), a \rightarrow a', f(a', x) \rightarrow f(x, x), f(x, a') \rightarrow f(x, x),$
 $g(b, b) \rightarrow f(a, a), b \rightarrow b', g(b', x) \rightarrow g(x, x), g(x, b') \rightarrow g(x, x)\}$
 Es gilt $f(a', a') \xrightarrow{*}_R g(b', b')$ jedoch nicht $f(a', a') \downarrow_R g(b', b')$.
 R links linear $\epsilon - \epsilon$ abgeschlossen :



Parallel Reduktion

Beachte: Seien \rightarrow, \Rightarrow mit $\xrightarrow{*} = \Rightarrow^*$. (Oft: $\rightarrow \subseteq \Rightarrow \subseteq \xrightarrow{*}$).
 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 $t|_{u_i} = \sigma_i(l_i) :: t' = t[\sigma_i(r_i)]_{u_i} (u_i \in U) (t[u_1 \leftarrow \sigma_1(r_1)] \dots t[u_n \leftarrow \sigma_n(r_n)])$.
- ▶ 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 $\mapsto_R^* = \xrightarrow{*}_R$ 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:
 $f(g(x, a), y) \rightarrow 0, g(b, c) \rightarrow 1$ hat diese Eigenschaft nicht. Durch Umgruppieren der Argumente kann oft die Eigenschaft erreicht werden, etwa $f(g(a, x), y) \rightarrow 0, g(b, c) \rightarrow 1$)
- ▶ **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 $s \xrightarrow{*}_R u \xrightarrow{\leq 1}_R t$ und $s \xrightarrow{\leq 1}_R v \xrightarrow{*}_R t$.
 R linear und stark-abgeschlossen, so R streng-konfluent.

Fogerungen

- ▶ Folgt aus $CP(R) = \emptyset$ die Konfluenz? **Nein**.
 $R = \{f(x, x) \rightarrow a, g(x) \rightarrow f(x, g(x)), b \rightarrow g(b)\}$.
 Betrachte $g(b) \rightarrow f(b, g(b)) \rightarrow f(g(b), g(b)) \rightarrow a$
 "Outermost" Reduktion.
 $g(b) \rightarrow g(g(b)) \xrightarrow{*}_R g(a) \rightarrow f(a, g(a))$ nicht zusammenführbar.
- ▶ Reguläre Systeme können **nicht terminierend** sein:
 $\{f(x, b) \rightarrow d, a \rightarrow b, c \rightarrow c\}$. Offenbar $CP = \emptyset$.
 $f(c, a) \rightarrow f(c, b) \rightarrow d$
 \downarrow^*
 $f(c, a) \rightarrow f(c, b)$. Beachte $f(c, a)$ hat eine Normalform. \rightsquigarrow
 Reduktionsstrategien die **Normalformberechnend** sind oder **kürzeste Berechnungen** liefern.
- ▶ Ein **Kontext** ist ein Term mit "Lücken" \square , z.B.
 $f(g(\square, s(0)), \square, h(\square))$ als "Baumuster" (pattern) zu Regel
 $f(g(x, s(0)), y, h(z)) \rightarrow x$. Lücken dürfen beliebig gefüllt werden.

Terminierungs-Kriterien

Satz 9.25. R ist genau dann terminierend, wenn es eine noethersche Partialordnung $>$ auf den Grundtermen $Term(F)$ gibt die monoton ist, so dass gilt $\sigma(l) > \sigma(r)$ für jede Regel $l \rightarrow r \in R$ und Grundsubstitution σ .

Beweis: \curvearrowright Definiere $s > t$ gdw. $s \xrightarrow{+} t$ ($s, t \in Term(F)$)
 \curvearrowleft Ang. \rightarrow_R nicht terminierend, $t_0 \rightarrow t_1 \rightarrow \dots (V(t_i) \subseteq V(t_0))$. Sei σ eine Grundsubstitution mit $V(t_0) \subseteq D(\sigma)$, dann $\sigma(t_0) > \sigma(t_1) > \dots \not\downarrow$.

Problem: Unendlicher Test.

Definition 9.26. Eine **Reduktionsordnung** ist Partialordnung $>$ auf $Term(F, V)$ mit
 (i) $>$ ist Noethersch (ii) $>$ ist stabil und (iii) $>$ ist monoton.

Satz 9.27. R ist genau dann Noethersch, wenn es eine Reduktionsordnung $>$ gibt mit $l > r$ für alle $l \rightarrow r \in R$

Terminierungs-Kriterien

Beachte: Es gibt keine totalen Reduktionsordnungen.

$x > y?$ $\rightsquigarrow \sigma(x) > \sigma(y)$
 $f(x, y) > f(y, x)$? 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.

$$\begin{array}{ccccccccc}
 f(f(g(x))) & = & f(h(x)) & f(f(x)) & = & g(h(g(x))) & f(h(x)) & = & h(g(x)) \\
 \text{KB} & \rightarrow & l(f) = 3 & l(g) = 2 & \rightarrow & l(h) = 1 & & \rightarrow & \\
 \text{RPO} & \leftarrow & g > h & > f & \leftarrow & & & \leftarrow &
 \end{array}$$

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, $>$ Reduktionsordnung, $R = \emptyset$.

- Repeat** while E nicht leer
- (1) Entferne $t = s$ aus E mit $t > s$, $R := R \cup \{t \rightarrow s\}$ sonst abort
 - (2) Bringe rechte Seite der Regeln in Normalform mit R
 - (3) Erweitere E um alle mit R normalisierten kritischen Paare die $t \rightarrow s$ 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.

$\triangleright WES :: \Sigma = \{a, b, c\}, E = \{a^2 = \lambda, b^2 = \lambda, ab = c\}$
 $u < v$ gdw. $|u| < |v|$ oder $|u| = |v|$ und $u <_{lex} v$ mit $a <_{lex} b <_{lex} c$

$E_0 = \{a^2 = \lambda, b^2 = \lambda, ab = c\}, R_0 = \emptyset$
 $E_1 = \{b^2 = \lambda, ab = c\}, R_1 = \{a^2 \rightarrow \lambda\}, CP_1 = \emptyset$
 $E_2 = \{ab = c\}, R_2 = \{a^2 \rightarrow \lambda, b^2 \rightarrow \lambda\}, CP_2 = \emptyset$
 $R_3 = \{a^2 \rightarrow \lambda, b^2 \rightarrow \lambda, ab \rightarrow c\}, NCP_3 = \{(b, ac), (a, cb)\}$
 $E_3 = \{b = ac, a = cb\}$
 $R_4 = \{a^2 \rightarrow \lambda, b^2 \rightarrow \lambda, ab \rightarrow c, ac \rightarrow b\}, NCP_4 = \emptyset, E_4 = \{a = cb\}$
 $R_5 = \{a^2 \rightarrow \lambda, b^2 \rightarrow \lambda, ab \rightarrow c, ac \rightarrow b, cb \rightarrow a\}, NCP_5 = \emptyset, E_5 = \emptyset$

Darstellung Rekursiver Funktionen

Voraussetzung: Zu $k_1, \dots, k_n \in \mathbb{N}$ gibt es kleinstes $k \in \mathbb{N}$ mit $g(k_1, \dots, k_n, k) = 0$

Behauptung: Für alle $i \in \mathbb{N}, i \leq k$ gilt $\hat{f}^*(\hat{k}_1, \dots, \hat{k}_n, (k - i)) \rightarrow_{E_{\hat{f}}} \hat{k}$

Bew: Induktion nach i :

- $i = 0$:: $\hat{f}^*(\hat{k}_1, \dots, \hat{k}_n, \hat{k}) \rightarrow \hat{f}^+(\hat{g}(\hat{k}_1, \dots, \hat{k}_n, \hat{k}), \hat{k}_1, \dots, \hat{k}_n, \hat{k}) \rightarrow_{E_{\hat{g}}} \hat{f}^+(g(k_1, \dots, k_n, k), \hat{k}_1, \dots, \hat{k}_n, \hat{k}) \rightarrow \hat{k}$
- $i > 0$:: $\hat{f}^*(\hat{k}_1, \dots, \hat{k}_n, k - (i + 1)) \rightarrow \hat{f}^+(\hat{g}(\hat{k}_1, \dots, \hat{k}_n, k - (i + 1)), \hat{k}_1, \dots, \hat{k}_n, k - (i + 1)) \rightarrow_{E_{\hat{g}}} \hat{f}^+(\hat{s}(\hat{x}), \hat{k}_1, \dots, \hat{k}_n, k - (i + 1)) \rightarrow \hat{f}^*(\hat{k}_1, \dots, \hat{k}_n, \hat{s}(k - (i + 1))) \rightarrow_{E_{\hat{f}}} \hat{k}$
 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. $g(k, m) = \delta_{k,m} \rightsquigarrow \hat{f}^*(\hat{k}, k + 1)$ 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:

$$\begin{aligned} \hat{f}(x, y) &= \hat{f}_1(\hat{h}_1(x, y), \hat{h}_2(x, y), \hat{h}_3(x, y), \hat{f}_2(\hat{h}_1(x, y)), \hat{f}_2(\hat{h}_2(x, y)), \hat{f}_2(\hat{h}_3(x, y))) \\ \hat{f}_1(x_1, x_2, x_3, \hat{0}, \hat{0}, \hat{0}) &= \hat{g}(x_1, x_2, x_3), \quad \hat{f}_2(\hat{0}) = \hat{0}, \quad \hat{f}_2(\hat{s}(x)) = \hat{f}_2(x) \end{aligned}$$

$(\hat{f}, E_{\hat{g}}, E_{\hat{h}_1}, E_{\hat{h}_2}, E_{\hat{h}_3} \cup REST)$ 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 $f(\mathcal{J}(t_1), \mathcal{J}(t_2)) = \mathcal{J}(t)$. Es gibt Zahlenterme T_i ($i = 1, 2, 3$) mit

$g(\mathcal{J}(T_1), \mathcal{J}(T_2), \mathcal{J}(T_3)) = \mathcal{J}(t)$ und $h_i(\mathcal{J}(t_1), \mathcal{J}(t_2)) = \mathcal{J}(T_i)$.

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

$\hat{f}(t_1, t_2) =_{E_{\hat{f}}} \hat{f}_1(T_1, T_2, T_3, \hat{0}, \hat{0}) \rightsquigarrow \hat{f}(t_1, t_2) =_{E_{\hat{f}}} t (=_{E_{\hat{f}}} \hat{g}(T_1, T_2, T_3))$

\curvearrowleft Für Zahlenterme t_1, t_2, t gelte $\hat{f}(t_1, t_2) =_{E_{\hat{f}}} t$, dann

$\hat{f}_1(\hat{h}_1(t_1, t_2), \hat{h}_2(t_1, t_2), \hat{h}_3(t_1, t_2), \hat{f}_2(\hat{h}_1(t_1, t_2)), \dots) =_{E_{\hat{f}}} t$. 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 gibt 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:

$$\rightsquigarrow h_i(\mathcal{J}(T_1), \mathcal{J}(T_2)) = \mathcal{J}(T_i), \quad g(\mathcal{J}(T_1), \mathcal{J}(T_2), \mathcal{J}(T_3)) = \mathcal{J}(t)$$

\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 (-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: $M_e : P_n \times \mathbb{N}^n \rightarrow \mathbb{N}^n$

- $M_e(a_i, \langle x_1, \dots, x_n \rangle) = \langle \dots, x_{i-1}, x_i + 1, x_{i+1}, \dots \rangle$ (s_i :: $x_i - 1$)
- $M_e(M_1 M_2, \langle x_1, \dots, x_n \rangle) = M_e(M_2, M_e(M_1, \langle x_1, \dots, x_n \rangle))$
- $M_e((M)_i, \langle x_1, \dots, x_n \rangle) = \begin{cases} \langle x_1, \dots, x_n \rangle & x_i = 0 \\ M_e((M)_i, M_e(M, \langle x_1, \dots, x_n \rangle)) & \text{sonst} \end{cases}$

Nicht berechenbare Funktionen

Sei E rekursiv, T_i rekursiv. Dann ist das Prädikat

$$P(t_1, \dots, t_n, t_{n+1}) \text{ gdw } \hat{f}(t_1, \dots, t_n) =_E t_{n+1}$$

r.a. Prädikat auf $T_1 \times \dots \times T_n \times T_{n+1}$

Implementiert \hat{f} die Funktion f , so stellt P den Graphen der Funktion f dar $\rightsquigarrow f \in \mathfrak{R}_p$.

Kleenescher Normalformsatz: $f(x_1, \dots, x_n) = U(\mu_y [T_n(p, x_1, \dots, x_n, y) = 0])$

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

$$(1') P_1(x, y, h(x)) = 0 \text{ und } (2') P_2(x, y, h(x)) = 0$$

(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

$$\hat{P}_1(x, y, \hat{h}(x)) = \hat{0} \text{ und } \hat{P}_2(x, y, \hat{h}(x)) = \hat{0}$$

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

$$red \text{ als } \lambda x. \text{if } x \text{ gerade then } x \text{ else } 1, \quad s \text{ Nachfolger}$$

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 $cond : nat \text{ gerade } gerade \rightarrow gerade$ mit Interpretation

$$\lambda x. \text{if } x \text{ gerade then } 0 \text{ else } 1, \quad \lambda x, y, z. \text{if } x = 0 \text{ then } y \text{ else } z$$

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:

$$red(s(0)) = odd, red(s(s(x))) = odd \text{ if } red(x) = odd$$

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 \mathcal{A} eine rekursive termerzeugte sig- Algebra. Dann gibt es eine endliche Anreicherung sig' von sig und eine endliche Spezifikation $spec' = (sig', E)$ mit $T_{spec'}|_{sig} \cong \mathcal{A}$.

Satz 10.16. Sei \mathcal{A} eine termerzeugte sig- Algebra. Dann sind äquivalent:

- ▶ \mathcal{A} ist rekursiv.
- ▶ Es gibt eine endliche Anreicherung (ohne neue Sorten) sig' von sig und ein endliches konvergentes Regelsystem R , so dass $\mathcal{A} \cong T_{spec'}|_{sig}$ 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 $t \xrightarrow{*}_R \bar{t}$.

- ▶ 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 λ -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 $t \xrightarrow{k} \bar{t}$ eine kürzeste Ableitungskette, so $t \xrightarrow{\leq k}_{LMOM} \bar{t}$. Durch Structure-Sharing-Methoden kann die Schranke für LMOM kleiner gewählt werden.

Funktionale Berechnungsmodelle

- ▶ Partiiell 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**

$$(yx(\lambda x.x))[x := M] \equiv (yN(\lambda x.x))$$

Hierbei bleiben freie Vorkommen von Variablen in N frei (u.u. durch Umbenennung)

$$(\lambda x.y)[y := xx] \equiv \lambda z.xx \text{ z "neu"}$$

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 ::
 $\Lambda(C, V) = C \setminus V \mid (\Lambda \Lambda) \mid (\lambda V.\Lambda)$
- ▶ 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. $Fz_{n1} \dots z_{nk} = z_{f(n1, \dots, nk)}$
- ▶ 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_β ist konfluent
- ▶ **Satz von Curry** Hat M eine Normalform so gilt $M \rightarrow_j^* N$, d.h. Leftmost Reduktion ist Normalisierend.

Reduktionsstrategien für Ersetzungssysteme

Definition 11.1. Sei R ein TES.

- ▶ Eine **Einschritt-Reduktionsstrategie** \mathcal{G} für R ist eine Abbildung $\mathcal{G} : \text{Term}(R, V) \rightarrow \text{Term}(R, V)$ mit $t = \mathcal{G}(t)$ falls t in Normalform und $t \rightarrow_R \mathcal{G}(t)$ sonst.
- ▶ \mathcal{G} ist eine **Mehrschritt-Reduktionsstrategie** für R wenn $t = \mathcal{G}(t)$ falls t in Normalform und $t \xrightarrow{+}_R \mathcal{G}(t)$ sonst.
- ▶ Eine Reduktionsstrategie \mathcal{G} heißt **normalisierend** für R , falls für jeden Term t der eine Normalform hat die Folge $(\mathcal{G}^n(t))_{n \geq 0}$ eine Normalform enthält. (Insbesondere endlich ist).
- ▶ Eine Reduktionsstrategie \mathcal{G} heißt **cofinal** für R , falls für jedes t und $r \in \Delta^*(t)$ es ein $n \in \mathbb{N}$ gibt mit $r \xrightarrow{*}_R \mathcal{G}^n(t)$.

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. $PI(t) = \bar{t}$, wobei $t \mapsto \bar{t}$ (Alle innermost Redexe werden reduziert).
- ▶ **Leftmost-Outermost (Call-by-Name)**. Einschritt-RS.
- ▶ **Parallel-Outermost**. Mehrschritt-RS. $PO(t) = \bar{t}$, wobei $t \mapsto \bar{t}$ (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).

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.

Hilfsmittel: Elementare Reduktionsdiagramme und Reduktionsdiagramme:

$$\begin{array}{ccc}
 Sab(lc) & \rightarrow & a(lc)(b(lc)) \\
 \downarrow & & \downarrow \\
 Sabc & \rightarrow & ac(bc) \\
 \\
 la & \rightarrow & a \\
 \downarrow_{\emptyset} & & \downarrow_{\emptyset} \\
 la & \rightarrow & a
 \end{array}
 \quad
 \begin{array}{ccc}
 Ka(lb) & \rightarrow & Kab \\
 \downarrow & & \downarrow \\
 a & \rightarrow_{\emptyset} & a
 \end{array}$$

Zusammensetzung E-Reduktionsdiagramme

Reduktionsdiagramme und Projektionen:

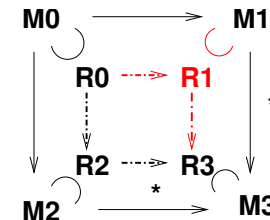
$$\begin{array}{ccccccc}
 t_0 & \rightarrow & t_1 & \rightarrow & \dots & \rightarrow & t_n \\
 \downarrow & & \downarrow * & \rightarrow * & \rightarrow * & & \downarrow * \\
 t'_1 & \rightarrow * & & \rightarrow * & \rightarrow * & & \downarrow * \\
 \downarrow & & \downarrow * & \rightarrow * & \rightarrow * & & \downarrow * \\
 \vdots & & \vdots & \dots & \dots & & \downarrow * \\
 \downarrow & & \downarrow * & \rightarrow * & \rightarrow * & & \downarrow * \\
 t'_m & \rightarrow * & & \rightarrow * & \rightarrow * & & \downarrow *
 \end{array}$$

$R_4 = R_2 / R_1$ $R_3 = R_1 / R_2$ Projektionen

Seien $R_1 :: t \xrightarrow{\pm} t'$ und $R_2 :: t \xrightarrow{\pm} t'$ 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 Term paaren M, R so dass $R \subseteq M$ und R ist Redex (z.B. LMOM, LMIM, ...).

- Π hat Eigenschaft I wenn für das Vorliegen von D wie im Lemma gilt $\Pi(M_0, R_0) \wedge \Pi(M_2, R_2) \wedge \Pi(M_3, R_3) \rightsquigarrow \Pi(M_1, R_1)$
- 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).

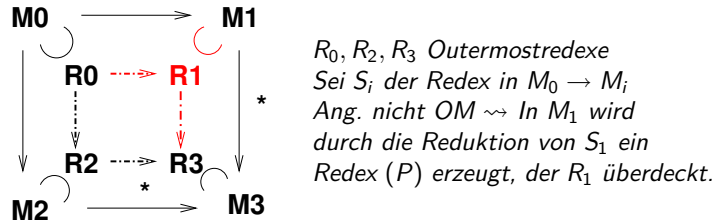
Lemma 11.7. Trennbarkeit von Entwicklungen. Π habe Eigenschaft II. dann kann jede Entwicklung $R :: M_0 \rightarrow \dots \rightarrow M_n$ aufgeteilt werden in einen Π -Anteil gefolgt von einem $\neg \Pi$ -Anteil.

Genauer: es gibt Reduktionsfolgen $R_{\Pi} :: M_0 = N_0 \rightarrow^{R_0} \dots \rightarrow^{R_{k-1}} N_k$ mit $\Pi(N_i, R_i)$ ($i < k$) und $R_{\neg \Pi} :: N_k \rightarrow^{R_k} \dots \rightarrow^{R_{k+l-1}} N_{k+l}$ 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



In $M_1 \rightarrow M_3$ wird R_1 wieder outermost. D.h. P wird reduziert: In $M_1 \rightarrow M_3$ 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: $F(x, b) \rightarrow d, a \rightarrow b, c \rightarrow c :: F(c, a) \rightarrow F(c, b)$

Nachfolger von Redexe (Reste)

Definition 11.9. Spuren in Reduktionsfolgen:

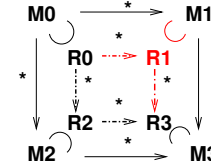
- ▶ Sei $\mathfrak{R} :: M_0 \rightarrow M_1 \rightarrow \dots$ Reduktionsfolge. Sei M_j fest und $L_i \subseteq M_i$ ($i \geq j$) (sofern M_i existent) Redexe mit $L_j \rightarrow \dots \rightarrow L_{j+1} \rightarrow \dots$
 Die Folge $\mathcal{L} = (L_{j+i})_{i \geq 0}$ ist eine **Spur** von Nachfolgern (Reste) von Redexe in M_j .
- ▶ \mathcal{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 \mathcal{D} ein Reduktionsdiagramm mit $R_i \subseteq M_i, R_0 \rightarrow \dots \rightarrow R_2 \rightarrow \dots \rightarrow R_3$ ist Π Spur.



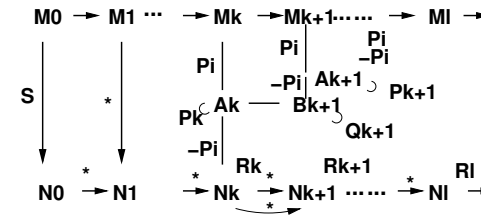
Dann ist $R_0 \rightarrow \dots \rightarrow R_1 \rightarrow \dots \rightarrow R_3$ über M_1 ebenfalls eine Π Spur

- ▶ 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 $S \rightarrow \dots \rightarrow S'$ via \mathfrak{R} existiert. Dann gibt es eine eindeutige Π -Spur $S \rightarrow \dots \rightarrow S'$ 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:



$\mathfrak{R} :: M_0 \rightarrow \dots$ sei Π -fair und $\mathfrak{R}' :: N_0 \xrightarrow{*}$ eine Projektion. $(R_{k+i}) \Pi$ Spur.
 $\forall k \exists A_k. M_k \xrightarrow{\Pi} A_k \xrightarrow{\neg \Pi} N_k$ äquivalent zur vollständigen Entwicklung $M_k \rightarrow N_k$. Bei dieser Umordnung sind die beiden Ableitungen zwischen N_k und N_{k+1} äquivalent. Insbesondere bleiben Π -Spuren erhalten.
 Ergibt eine Stufenform: $A_k - B_{k+1} - A_{k+1} - B_{k+2} - \dots$

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 $R_k, \dots, R_{k+1} \dots$ 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 $P_k \rightarrow \dots \rightarrow R_k \rightarrow \dots \rightarrow R_{k+1}$ via B_{k+1} geliftet werden zu Π -Spur $P_k \rightarrow \dots \rightarrow Q_{k+1} \rightarrow \dots \rightarrow R_{k+1}$.

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. ζ .

Folgerungen

Lemma 11.12. Sei $\mathfrak{R} :: M_0 \rightarrow M_1 \rightarrow \dots$ 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 $M_l \rightarrow M_{l+1}$

- 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 $\mathfrak{R} :: t \rightarrow t_1 \rightarrow \dots \rightarrow$ unendlich Π -fair und $\mathfrak{R}' :: t \rightarrow t'_1 \rightarrow \dots \rightarrow \bar{t}$ Normalform.

\mathfrak{R} enthält unendlich viele outermost Reduktionsschritte (sonst wäre sie nicht Π -fair). Dann ist aber auch $\mathfrak{R} / \mathfrak{R}'$ unendlich. ζ .

Beachte: Der Satz gilt nicht für LMOM-Strategie: Eigenschaft II ist nicht erfüllt. Betrachte hierfür $a \rightarrow b, c \rightarrow c, f(x, b) \rightarrow d$.

Folgerungen für orthogonale Systeme

Definition 11.14. Sei R orthogonal, $l \rightarrow r \in R$ 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-Orthogonal	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 $l \rightarrow r \in R, x \in Var(l)$.

- R heißt **variablenreduzierend**, falls für alle $l \rightarrow r \in R, |l|_x > |r|_x$
- R heißt **variablenerhaltend**, falls für alle $l \rightarrow r \in R, |l|_x = |r|_x$
- R heißt **variablenvermehrend**, falls für alle $l \rightarrow r \in R, |l|_x \leq |r|_x$
- 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:**
 $R :: f(x, y) \rightarrow g(h(x), y), g(x, y) \rightarrow l(x, y), a \rightarrow c, b \rightarrow d.$
 Betrachte den Term $f(a, b)$ und seine Ableitungen.
 Alle Ableitungsfolgen sind gleich lang.
- Variablenvermehrend (non erasing):**
 $R :: f(x, b) \rightarrow g(x, x), a \rightarrow b, c \rightarrow d.$
 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 $L(D) :: t \rightarrow t_1 \rightarrow \dots \rightarrow t_i \xrightarrow{S} \dots \rightarrow t_j \xrightarrow{*} t'$.
 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 $D' :: t \rightarrow t_1 \rightarrow \dots \rightarrow t_i \xrightarrow{S_i} t'_{i+1} \xrightarrow{S} \dots \xrightarrow{<} t'_j \xrightarrow{*} t'$
 $|D'| \leq |D|, L(D') < L(D) \rightsquigarrow$ es gibt Ableitung D' mit $L(D') = 0$.

Eine sequentielle Strategie für paror Systeme

Beispiel 11.28. Seien $f, g : \mathbb{N}^+ \rightarrow \mathbb{N}$ rekursive Funktionen. Definiere Reduktionssystem R auf $\mathbb{N} \times \mathbb{N}$ mit Regeln:

- ▶ $(x, y) \rightarrow (f(x), y)$ falls $x, y > 0$
- ▶ $(x, y) \rightarrow (x, g(y))$ falls $x, y > 0$
- ▶ $(x, 0) \rightarrow (0, 0)$ falls $x > 0$
- ▶ $(0, y) \rightarrow (0, 0)$ falls $y > 0$

Offenbar ist R konfluent. Einzige Normalform ist $(0, 0)$ und für $x, y > 0$, (x, y) hat Normalform gdw. $\exists n. f^n(x) = 0 \vee g^n(y) = 0$.

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 Einschrittstrategie 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 $f^i(x) = f^i(x) \neq 0$. 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 $g^i(y) = g^i(y) \neq 0$. 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 Einschrittstrategie 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 $\mathfrak{R} :: t_0 \rightarrow t_1 \rightarrow \dots$ eine Reduktionsfolge im TES R . Markiere in jedem Schritt in \mathfrak{R} alle Top-Symbole von Redex 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:: $P \times Q \rightarrow x x, R \rightarrow S, l x \rightarrow x$
 Betrachte $\mathfrak{R} :: PR(IQ) \rightarrow \underline{PR}Q \rightarrow \underline{R}R \rightarrow SR$
 Es gibt keine standard Reduktion von $PR(IQ)$ nach SR

Fakt: λ -Kalkül 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, $t \in \text{Term}(R)$ mit Normalform $t \downarrow$. Ein Redex $s \subseteq t$ ist ein **notwendiger** Redex, falls in jeder Reduktionsfolge $t \rightarrow \dots \rightarrow t \downarrow$ 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, $t \in \text{Term}(R)$, 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 $C[\dots, \dots, \dots]$ ein Kontext mit n -Plätze (Löcher), σ eine Substitution der Redexe s_1, \dots, s_n in Plätze $1, \dots, n$. Das Lemma impliziert folgende Eigenschaft:
 $\forall C[\dots, \dots, \dots]$ in Normalform, $\forall \sigma \exists i. s_i$ notwendig in $C[s_1, \dots, s_n]$.
 Welches s_i notwendig ist, hängt von σ ab.

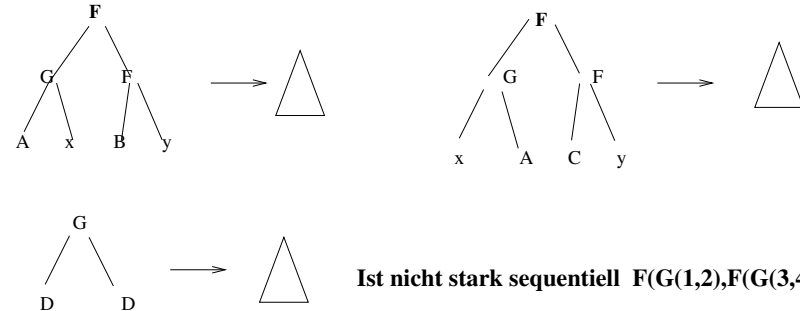
Sequentielle Orthogonale TES

Definition 11.37. Sei R orthogonal.

- ▶ R ist sequentiell* gdw. $\forall C[\dots, \dots, \dots]$ in normalform $\exists i \forall \sigma. s_i$ ist notwendig in $C[s_1, \dots, s_n]$
 Leider ist diese Eigenschaft unentscheidbar
- ▶ Sei $C[\dots]$ Kontext. Die Reduktionsrelation $\rightarrow_?$ (mögliche Reduktion) ist definiert durch

$$C[s] \rightarrow_? C[r]$$
 für jeden Redex s und beliebigen Term r
 $\rightarrow_?^*$ und Reste analog definiert.
- ▶ Ein Redex s in t heißt **stark notwendig** wenn in jeder Reduktionsfolge $t \rightarrow_? \dots \rightarrow_? t'$, wobei t' eine Normalform ist, ein Nachfolger von s reduziert wird.
- ▶ R ist **stark sequentiell** falls $\forall C[\dots, \dots, \dots]$ in Normalform $\exists i \forall \sigma. s_i$ stark notwendig.

Beispiel



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 Einschnittstrategie für solche Systeme