

# Analyse und Verifikation

LVA 185.276, VU 2.0, ECTS 3.0

SS 2013

(Stand: 26.05.2013)

Jens Knoop



Technische Universität Wien  
Institut für Computersprachen



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Lit/863

# Inhaltsverzeichnis

## Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Inhaltsverzeichnis (1)

## Teil I: Motivation

- ▶ **Kap. 1: Grundlagen**
  - 1.1 Motivation
  - 1.2 Modellsprache *WHILE*
  - 1.3 Semantik von Ausdrücken
  - 1.4 Syntaktische und semantische Substitution
  - 1.5 Induktive Beweisprinzipien
  - 1.6 Semantikdefinitionsstile
- ▶ **Kap. 2: Operationelle Semantik von *WHILE***
  - 2.1 Strukturell operationelle Semantik
  - 2.2 Natürliche Semantik
  - 2.3 Strukturell operationelle und natürliche Semantik im Vergleich

# Inhaltsverzeichnis (2)

- ▶ **Kap. 3: Denotationelle Semantik von *WHILE***
  - 3.1 Denotationelle Semantik
  - 3.2 Fixpunktfunktional
  - 3.3 Ordnungen, Verbände, Fixpunkttheoreme
- ▶ **Kap. 4: Axiomatische Semantik von *WHILE***
  - 4.1 Partielle und totale Korrektheit
  - 4.2 Beweiskalkül für partielle Korrektheit
  - 4.3 Beweiskalkül für totale Korrektheit
  - 4.4 Korrektheit und Vollständigkeit
  - 4.5 Beispiele zum Beweis partieller Korrektheit
  - 4.6 Beispiele zum Beweis totaler Korrektheit
  - 4.7 Automatische Ansätze axiomatischer Programmverifikation
  - 4.8 Historische Meilensteine der Programmverifikation
- ▶ **Kap. 5: Worst-Case Execution Time Analyse**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Inhaltsverzeichnis (3)

## Teil II: Analyse

- ▶ Kap. 6: Programmanalyse
  - 6.1 Motivation
  - 6.2 Datenflussanalyse
  - 6.3 *MOP*-Ansatz
  - 6.4 *MaxFP*-Ansatz
  - 6.5 Koinzidenz- und Sicherheitstheorem
  - 6.6 Generischer Fixpunktalgorithmus und Terminierungstheorem
  - 6.7 Zusammenfassung und Überblick
  - 6.8 Beispiele: Verfügbare Ausdrücke, einfache Konstanten
    - 6.8.1 Verfügbare Ausdrücke
    - 6.8.2 Einfache Konstanten
- ▶ Kap. 7: Programmverifikation vs. Programmanalyse

# Inhaltsverzeichnis (4)

- ▶ **Kap. 8: Reverse Datenflussanalyse**
  - 8.1 Grundlagen
  - 8.2 *rJOP*-Ansatz
  - 8.3 *rMinFP*-Ansatz
  - 8.4 Reverses Koinzidenz- und Sicherheitstheorem
  - 8.5 Generischer Fixpunktalgorithmus und Reverses Terminierungstheorem
  - 8.6 Analyse und Verifikation: Analogien und Gemeinsamkeiten
  - 8.7 Anwendungen reverser Datenflussanalyse
  - 8.8 Zusammenfassung
- ▶ **Kap. 9: Chaotische Fixpunktiteration**
  - 9.1 Motivation
  - 9.2 Chaotisches Fixpunktiterationstheorem
  - 9.3 Anwendungen
    - 9.3.1 Vektor-Iteration
    - 9.3.2 Datenflussanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Inhaltsverzeichnis (5)

## ► Kap. 10: Basisblock- vs. Einzelanweisungsgraphen

### 10.1 Motivation

10.1.1 Kantenbenannter Einzelanweisungsansatz

10.1.2 Knotenbenannter Basisblockansatz

### 10.2 Verfügbarkeit von Ausdrücken

10.1.1 Knotenbenannter Basisblockansatz

10.2.2 Knotenbenannter Einzelanweisungsansatz

10.2.3 Kantenbenannter Einzelanweisungsansatz

### 10.3 Konstantenausbreitung und -faltung

10.3.1 Kantenbenannter Einzelanweisungsansatz

10.3.2 Knotenbenannter Basisblockansatz

### 10.4 Schattenhafte Variablen

### 10.5 Fazit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L7/863 r

# Inhaltsverzeichnis (6)

## Teil III: Transformation und Optimalität

- ▶ **Kap. 11: Elimination partiell toter Anweisungen**
  - 11.1 Motivation
  - 11.2 PDCE/PFCE: Transformation und Optimalität
  - 11.3 Implementierung von PDCE/PFCE
- ▶ **Kap. 12: Elimination partiell redundanter Anweisungen**
  - 12.1 Motivation
  - 12.2 EAM: Einheitliche PREE/PRAE-Behandlung
  - 12.3 EAM: Transformation und Optimalität
- ▶ **Kap. 13: Kombination von PRAE und PDCE**
- ▶ **Kap. 14: Konstantenfaltung auf dem Wertegraphen**
  - 14.1 Motivation
  - 14.2 Der VG-Konstantenfaltungsalgorithmus
  - 14.3 Der PVG-Konstantenfaltungsalgorithmus



## Teil IV: Abstrakte Interpretation und Modellprüfung

### ▶ Kap. 15: Abstrakte Interpretation und DFA

15.1 Theorie abstrakter Interpretation

15.2 Systematische Konstruktion abstrakter Interpretationen

15.3 Korrektheit abstrakter Interpretationen

15.4 Vollständigkeit und Optimalität

### ▶ Kap. 16: Modellprüfung und DFA

# Inhaltsverzeichnis (8)

## Teil V: Abschluss und Ausblick

- ▶ Kap. 17: Resümee und Perspektiven
- ▶ Literatur
- ▶ Anhang
  - ▶ A Mathematische Grundlagen
    - A.1 Mengen und Relationen
    - A.2 Ordnungen
    - A.3 Verbände
    - A.4 Vollständige partielle Ordnungen
    - A.5 Fixpunkte und Fixpunkttheoreme

# Teil I

## Motivation

### Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Kapitel 1

## Grundlagen

Inhalt

**Kap. 1**

1.1

1.2

1.3

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

12/863

## Syntax und Semantik von Programmiersprachen:

- ▶ **Syntax:** Regelwerk zur präzisen Beschreibung wohlgeformter Programme
- ▶ **Semantik:** Regelwerk zur präzisen Beschreibung der Bedeutung oder des Verhaltens wohlgeformter Programme oder Programmteile (aber auch von Hardware beispielsweise)

# Literaturhinweise (1)

## Als Textbücher:

- ▶ Hanne R. Nielson, Flemming Nielson. [Semantics with Applications: An Appetizer](#). Springer-V., 2007.
- ▶ Hanne R. Nielson, Flemming Nielson. [Semantics with Applications: A Formal Introduction](#). Wiley Professional Computing, Wiley, 1992.

Bem.: Eine (überarbeitete) Version ist frei erhältlich auf [www.daimi.au.dk/~bra8130/Wiley\\_book/wiley.html](http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html)

# Literaturhinweise (2)

Ergänzend, weiterführend und vertiefend:

- ▶ Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. [Verification of Sequential and Concurrent Programs](#). 3. Auflage, Springer-V., 2009.
- ▶ Ernst-Rüdiger Olderog, Bernhard Steffen. [Formale Semantik und Programmverifikation](#). In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.
- ▶ Krzysztof R. Apt, Ernst-Rüdiger Olderog. [Programmverifikation – Sequentielle, parallele und verteilte Programme](#). Springer-V., 1994.

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

15/863

# Literaturhinweise (3)

- ▶ Jacques Loeckx, Kurt Sieber. [The Foundations of Program Verification](#). Wiley, 1984.
- ▶ Krzysztof R. Apt. [Ten Years of Hoare's Logic: A Survey – Part 1](#). ACM Transactions on Programming Languages and Systems 3(4):431-483, 1981.
- ▶ Krzysztof R. Apt. [Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism](#). Theoretical Computer Science 28(1-2):83-109, 1984.



# Unser Beispielszenario

- ▶ Modell(programmier)sprache *WHILE*
  - ▶ Syntax und Semantik
- ▶ Semantikdefinitionsstile
  1. Operationelle Semantik
    - 1.1 Großschritt-Semantik: Natürliche Semantik
    - 1.2 Kleinschritt-Semantik: Strukturell operationelle Semantik
  2. Denotationelle Semantik
  3. Axiomatische Semantik

## ▶ Operationelle Semantik

Die Bedeutung eines (programmiersprachlichen) Konstrukts ist durch die Berechnung beschrieben, die es bei seiner Ausführung auf der Maschine induziert. Wichtig ist insbesondere, **wie** der Effekt der Berechnung erzeugt wird.

## ▶ Denotationelle Semantik

Die Bedeutung eines Konstrukts wird durch mathematische Objekte modelliert, die den Effekt der Ausführung der Konstrukte repräsentieren. Wichtig ist **einzig** der Effekt, nicht wie er bewirkt wird.

## ▶ Axiomatische Semantik

Bestimmte Eigenschaften des Effekts der Ausführung eines Konstrukts werden als **Zusicherungen** ausgedrückt.

Bestimmte andere Aspekte der Ausführung werden dabei i.a. ignoriert.

# Kapitel 1.1

## Motivation

Inhalt

Kap. 1

**1.1**

1.2

1.3

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

19/863

...formale Semantik von Programmiersprachen einzuführen:

Die (mathematische) Rigorosität formaler Semantik

- ▶ erlaubt Mehrdeutigkeiten, Über- und Unterspezifikationen natürlichsprachlicher Beschreibungen aufzudecken und aufzulösen
- ▶ bietet die Grundlage für vertrauenswürdige Implementierungen der Programmiersprache, für die Analyse, Verifikation und Transformation von Programmen

# Unsere Modellsprache

- ▶ Die (Programmier-) Sprache *WHILE*

- ▶ Syntax
- ▶ Semantik

- ▶ Semantikdefinitionsstile

...wofür sie besonders geeignet sind und wie sie in Beziehung zueinander stehen:

- ▶ Operationelle Semantik ( $\rightsquigarrow$  Sprachimplementierung)
  - ▶ Natürliche Semantik
  - ▶ Strukturell operationelle Semantik
- ▶ Denotationelle Semantik ( $\rightsquigarrow$  Sprachdesign)
- ▶ Axiomatische Semantik ( $\rightsquigarrow$  Anwend'gsprogrammierung)
  - ▶ Beweiskalküle für partielle und totale Korrektheit
  - ▶ Korrektheit, Vollständigkeit

# Kapitel 1.2

## Modellsprache *WHILE*

Inhalt

Kap. 1

1.1

**1.2**

1.3

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

122/863

# Modellsprache *WHILE*

*WHILE*, der sog. “while”-Kern imperativer Programmiersprachen, besitzt:

- ▶ Zuweisungen (einschließlich der leeren Anweisung)
- ▶ Fallunterscheidungen
- ▶ while-Schleifen
- ▶ Sequentielle Komposition

Bemerkung: *WHILE* ist “schlank”, doch Turing-mächtig!

# Syntax und Semantik von *WHILE* im Überblick

- **Syntax:** Programme der Form:

$$\begin{aligned} \pi \quad ::= & \quad x := a \mid \textit{skip} \mid \\ & \quad \textit{if } b \textit{ then } \pi_1 \textit{ else } \pi_2 \textit{ fi} \mid \\ & \quad \textit{while } b \textit{ do } \pi_1 \textit{ od} \mid \\ & \quad \pi_1; \pi_2 \end{aligned}$$

- **Semantik:** Zustandstransformationen der Form:

$$\llbracket \pi \rrbracket : \Sigma \hookrightarrow \Sigma$$

mit  $\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{Var} \rightarrow D \}$  Menge aller **Zustände** über der **Variablenmenge** **Var** und geeignetem **Datenbereich**  $D$ .

(In der Folge werden wir für  $D$  meist die Menge der ganzen Zahlen  $\mathbb{Z}$  betrachten. Notationelle Konventionen: Der Pfeil  $\rightarrow$  bezeichnet **totale** Funktionen, der Pfeil  $\hookrightarrow$  **partielle** Funktionen; das Zeichen  $=_{df}$  steht für “definitionsgemäß gleich”.)



# Syntax von Zahlwörtern und Ausdrücken

## Zahlwörter (Numerale)

$$\begin{aligned}z & ::= 0 \mid 1 \mid 2 \mid \dots \mid 9 \\n & ::= z \mid n z\end{aligned}$$

## Arithmetische Ausdrücke

$$a ::= n \mid x \mid a_1 + a_2 \mid a_1 * a_2 \mid a_1 - a_2 \mid a_1 / a_2 \mid \dots$$

## Wahrheitswertausdrücke (Boolesche Ausdrücke)

$$\begin{aligned}b & ::= \text{true} \mid \text{false} \mid \\ & a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 < a_2 \mid a_1 \leq a_2 \mid \dots \mid \\ & b_1 \wedge b_2 \mid b_1 \vee b_2 \mid \neg b_1\end{aligned}$$

# Vereinbarungen

In der Folge bezeichnen wir mit:

- ▶ **Num** die Menge der *Zahlwörter*,  $n \in \mathbf{Num}$
- ▶ **Var** die Menge der *Variablen*,  $x \in \mathbf{Var}$
- ▶ **Aexpr** die Menge *arithmetischer Ausdrücke*,  $a \in \mathbf{Aexpr}$
- ▶ **Bexpr** die Menge *Boolescher Ausdrücke*,  $b \in \mathbf{Bexpr}$
- ▶ **Prg** die Menge aller *WHILE-Programme*,  $\pi \in \mathbf{Prg}$

In der Folge betrachten wir im Detail:

- ▶ Operationelle Semantik

- ▶ Natürliche Semantik:  $\llbracket \cdot \rrbracket_{ns} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$

- ▶ Strukturell operationelle Semantik:

- $\llbracket \cdot \rrbracket_{sos} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$

- ▶ Denotationelle Semantik:  $\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$

- ▶ Axiomatische Semantik: ...hat abweichenden Fokus!

...und deren Beziehungen zueinander, d.h. die Beziehungen zwischen

$$\llbracket \cdot \rrbracket_{sos}, \llbracket \cdot \rrbracket_{ns} \text{ und } \llbracket \cdot \rrbracket_{ds}$$

# Kapitel 1.3

## Semantik von Ausdrücken

Inhalt

Kap. 1

1.1

1.2

**1.3**

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

28/863

# Motivation

Die Semantik von *WHILE* stützt sich ab auf die Semantiken von

- ▶ Zahlwörtern
- ▶ arithmetischen Ausdrücken
- ▶ Wahrheitswertausdrücken

und den Begriff der

- ▶ Speicherzustände

Diese sind daher zunächst festzulegen.

# Speicherzustände

Die Menge der (Speicher-) Zustände ist gegeben durch

$$\blacktriangleright \Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{Var} \rightarrow \mathbb{Z} \}$$

wobei

$$\blacktriangleright \mathbb{Z} =_{df} \{ \dots, \mathbf{0}, \mathbf{1}, \mathbf{2}, \dots \}$$

die Menge der ganzen Zahlen bezeichnet.

# Semantik von Zahlwörtern

$\llbracket \cdot \rrbracket_N : \mathbf{Num} \rightarrow \mathbb{Z}$  ist induktiv definiert durch

- ▶  $\llbracket 0 \rrbracket_N =_{df} \mathbf{0}, \dots, \llbracket 9 \rrbracket_N =_{df} \mathbf{9}$
- ▶  $\llbracket ni \rrbracket_N =_{df} plus(mal(\mathbf{10}, \llbracket n \rrbracket_A), \llbracket i \rrbracket_N), i \in \{0, \dots, 9\}$
- ▶  $\llbracket -n \rrbracket_N =_{df} minus(\llbracket n \rrbracket_N)$

## Bemerkung:

- ▶  $0, 1, 2, \dots$  bezeichnen **syntaktische** Entitäten, Darstellungen von Zahlen;  $\mathbf{0}, \mathbf{1}, \mathbf{2}, \dots$  bezeichnen **semantische** Entitäten, hier ganze Zahlen:  $\mathbb{Z} =_{df} \{\dots, \mathbf{0}, \mathbf{1}, \mathbf{2}, \dots\}$ .
- ▶  $plus, mal : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ ,  $minus : \mathbb{Z} \rightarrow \mathbb{Z}$  bezeichnen semantische Operationen, hier die übliche Addition, Multiplikation und Vorzeichenwechsel auf  $\mathbb{Z}$ .
- ▶ Die Semantik von Zahlwörtern ist **zustandsunabhängig**.

# Semantik arithmetischer und Wahrheitswertausdrücke

## Semantik

- ▶ arithmetischer Ausdrücke:  $\llbracket \cdot \rrbracket_A : \mathbf{Aexpr} \rightarrow (\Sigma \hookrightarrow \mathbb{Z})$
- ▶ Wahrheitswertausdrücke:  $\llbracket \cdot \rrbracket_B : \mathbf{Bexpr} \rightarrow (\Sigma \hookrightarrow \mathbb{B})$

wobei

- ▶  $\mathbb{Z} =_{df} \{\dots, \mathbf{0}, \mathbf{1}, \mathbf{2}, \dots\}$  die Menge **ganzer Zahlen**
- ▶  $\mathbb{B} =_{df} \{\mathbf{wahr}, \mathbf{falsch}\}$  die Menge der **Wahrheitswerte**

bezeichnen.



# Semantik arithmetischer Ausdrücke

$\llbracket \cdot \rrbracket_A : \mathbf{Aexpr} \rightarrow (\Sigma \hookrightarrow \mathbb{Z})$  ist induktiv definiert durch

- ▶  $\llbracket n \rrbracket_A(\sigma) =_{df} \llbracket n \rrbracket_N$
- ▶  $\llbracket x \rrbracket_A(\sigma) =_{df} \sigma(x)$
- ▶  $\llbracket a_1 + a_2 \rrbracket_A(\sigma) =_{df} plus(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- ▶  $\llbracket a_1 * a_2 \rrbracket_A(\sigma) =_{df} mal(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- ▶  $\llbracket a_1 - a_2 \rrbracket_A(\sigma) =_{df} minus(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma))$
- ▶ ... (andere Operatoren analog)

wobei

- ▶ *plus*, *mal*, *minus* :  $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$  die übliche Addition, Multiplikation und Subtraktion auf  $\mathbb{Z}$  bezeichnen.

**Bemerkung:**

- ▶  $+$ ,  $*$ ,  $-$  bezeichnen **syntaktische** Entitäten (kurz: **Operatoren**); *plus*, *mal*, *minus* bezeichnen **semantische** Entitäten (kurz: **Operationen**)

# Semantik Boolescher Ausdrücke (1)

$\llbracket \cdot \rrbracket_B : \mathbf{Bexpr} \rightarrow (\Sigma \hookrightarrow \mathbb{B})$  ist induktiv definiert durch

- ▶  $\llbracket \text{true} \rrbracket_B(\sigma) =_{df} \mathbf{wahr}$
- ▶  $\llbracket \text{false} \rrbracket_B(\sigma) =_{df} \mathbf{falsch}$
- ▶  $\llbracket a_1 = a_2 \rrbracket_B(\sigma) =_{df} \begin{cases} \mathbf{wahr} & \text{falls } \text{gleich}(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma)) \\ \mathbf{falsch} & \text{sonst} \end{cases}$
- ▶ ... (andere Relatoren (z.B.  $<$ ,  $\leq$ , ...) analog)
  
- ▶  $\llbracket \neg b \rrbracket_B(\sigma) =_{df} \text{neg}(\llbracket b \rrbracket_B(\sigma))$
- ▶  $\llbracket b_1 \wedge b_2 \rrbracket_B(\sigma) =_{df} \text{und}(\llbracket b_1 \rrbracket_B(\sigma), \llbracket b_2 \rrbracket_B(\sigma))$
- ▶  $\llbracket b_1 \vee b_2 \rrbracket_B(\sigma) =_{df} \text{oder}(\llbracket b_1 \rrbracket_B(\sigma), \llbracket b_2 \rrbracket_B(\sigma))$

# Semantik Boolescher Ausdrücke (2)

Dabei bezeichnen

- ▶ **wahr** und **falsch** die Wahrheitswertkonstanten “wahr” und “falsch”
- ▶ *und*, *oder* :  $\mathbb{B} \times \mathbb{B} \rightarrow \mathbb{B}$  und *neg* :  $\mathbb{B} \rightarrow \mathbb{B}$  die übliche zweistellige logische Konjunktion und Disjunktion und einstellige Negation auf  $\mathbb{B}$
- ▶ *gleich* :  $\mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{B}$  die übliche Gleichheitsrelation auf  $\mathbb{Z}$

Inhalt

Kap. 1

1.1

1.2

**1.3**

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

35/863

# Semantik Boolescher Ausdrücke (3)

## Bemerkung:

- ▶ Beachte auch hier den Unterschied zwischen den **syntaktischen** Entitäten *true* und *false* und ihren **semantischen** Gegenstücken **wahr** und **falsch**
- ▶ Beachte ebenso den Unterschied zwischen den
  - ▶ Relatoren  $=$ ,  $>$ ,  $<$  (**syntaktische** Entitäten) und den **Relationen** *gleich*, *größer*, *kleiner* (**semantische** Entitäten)
  - ▶ Operatoren  $\neg$ ,  $\wedge$ ,  $\vee$  (**syntaktische** Entitäten) und den **Operationen** *neg*, *und*, *oder* (**semantische** Entitäten)

# Kapitel 1.4

## Syntaktische und semantische Substitution

# Freie Variablen

## ► arithmetischer Ausdrücke:

$$FV(n) = \emptyset$$

$$FV(x) = \{x\}$$

$$FV(a_1 + a_2) = FV(a_1) \cup FV(a_2)$$

...

## ► Boolescher Ausdrücke:

$$FV(\text{true}) = \emptyset$$

$$FV(\text{false}) = \emptyset$$

$$FV(a_1 = a_2) = FV(a_1) \cup FV(a_2)$$

...

$$FV(b_1 \wedge b_2) = FV(b_1) \cup FV(b_2)$$

$$FV(b_1 \vee b_2) = FV(b_1) \cup FV(b_2)$$

$$FV(\neg b_1) = FV(b_1)$$

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

1.5

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

38/863

# Eigenschaften von $\llbracket \cdot \rrbracket_A$ und $\llbracket \cdot \rrbracket_B$

## Lemma (1.4.1)

Seien  $a \in \mathbf{AExpr}$  und  $\sigma, \sigma' \in \Sigma$  mit  $\sigma(x) = \sigma'(x)$  für alle  $x \in FV(a)$ . Dann gilt:

$$\llbracket a \rrbracket_A(\sigma) = \llbracket a \rrbracket_A(\sigma')$$

## Lemma (1.4.2)

Seien  $b \in \mathbf{BExpr}$  und  $\sigma, \sigma' \in \Sigma$  mit  $\sigma(x) = \sigma'(x)$  für alle  $x \in FV(b)$ . Dann gilt:

$$\llbracket b \rrbracket_B(\sigma) = \llbracket b \rrbracket_B(\sigma')$$

# Syntaktische und semantische Substitution

Von zentraler Bedeutung:

- ▶ Der **Substitutionsbegriff!**

**Substitution** tritt in zwei Spielarten in Erscheinung:

- ▶ **Syntaktische** Substitution
- ▶ **Semantische** Substitution

Der Zusammenhang von **syntaktischer** und **semantischer** **Substitution** wird beschrieben durch:

- ▶ Das **Substitutionslemma 1.4.5**



# Syntaktische Substitution

## Definition (1.4.3, Syntaktische Substitution)

Die **syntaktische Substitution** für arithmetische Terme ist eine dreistellige Abbildung

$$\cdot[\cdot/\cdot] : \mathbf{Aexpr} \times \mathbf{Aexpr} \times \mathbf{Var} \rightarrow \mathbf{Aexpr}$$

die induktiv definiert ist durch:

$$n[t/x] =_{df} n \quad \text{für } n \in \mathbf{Num}$$

$$y[t/x] =_{df} \begin{cases} t & \text{falls } y = x \\ y & \text{sonst} \end{cases}$$

$$(t_1 \text{ op } t_2)[t/x] =_{df} (t_1[t/x] \text{ op } t_2[t/x]) \quad \text{für } op \in \{+, *, -, \dots\}$$

# Semantische Substitution

## Definition (1.4.4, Semantische Substitution)

Die **semantische Substitution** ist eine dreistellige Abbildung

$$\cdot[\cdot/\cdot] : \Sigma \times \mathbb{Z} \times \mathbf{Var} \rightarrow \Sigma$$

die definiert ist durch:

$$\sigma[\mathbf{z}/x](y) =_{df} \begin{cases} \mathbf{z} & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

# Substitutionslemma

Zusammenhang von syntaktischer und semantischer Substitution:

Lemma (1.4.5, Substitutionslemma)

$$\llbracket e[t/x] \rrbracket_A(\sigma) = \llbracket e \rrbracket_A(\sigma[\llbracket t \rrbracket_A(\sigma)/x])$$

wobei

- ▶  $[t/x]$  die *syntaktische Substitution* und
- ▶  $\llbracket t \rrbracket_A(\sigma)/x$  die *semantische Substitution*

bezeichnen.

Analog gilt ein entsprechendes Substitutionslemma für  $\llbracket \cdot \rrbracket_B$ .

# Kapitel 1.5

## Induktive Beweisprinzipien

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

**1.5**

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

44/863

# Induktive Beweisprinzipien (1)

## Grundlegend:

- ▶ Vollständige Induktion
- ▶ Verallgemeinerte Induktion
- ▶ Strukturelle Induktion

...zum Beweis einer Aussage (insbesondere der drei Lemmata in Kapitel 1.4).

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

**1.5**

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

45/863

# Induktive Beweisprinzipien (2)

Sei  $A$  eine Aussage,  $S$  eine (induktiv definierte) Struktur und bezeichne  $\text{Komp}(s)$  die Menge der Komponenten von  $s \in S$ .

Die Prinzipien der

▶ vollständigen Induktion

$$(A(1) \wedge (\forall n \in \mathbb{N}. A(n) \Rightarrow A(n+1))) \Rightarrow \forall n \in \mathbb{N}. A(n)$$

▶ verallgemeinerten Induktion

$$(\forall n \in \mathbb{N}. (\forall m < n. A(m)) \Rightarrow A(n)) \Rightarrow \forall n \in \mathbb{N}. A(n)$$

▶ strukturellen Induktion

$$(\forall s \in S. \forall s' \in \text{Komp}(s). A(s')) \Rightarrow \forall s \in S. A(s)$$

# Induktive Beweisprinzipien (3)

## Bemerkung:

- ▶ Vollständige, verallgemeinerte und strukturelle Induktion sind gleich mächtig.
- ▶ Abhängig vom Anwendungsfall ist oft eines der Induktionsprinzipien zweckmäßiger, d.h. einfacher anzuwenden. Zum Beweis von Aussagen über induktiv definierte Datenstrukturen ist i.a. das Prinzip der strukturellen Induktion am zweckmäßigsten.

# Beispiel: Beweis von Lemma 1.4.1 (1)

...durch strukturelle Induktion (über den induktiven Aufbau arithmetischer Ausdrücke)

Seien  $a \in \mathbf{AExpr}$  und  $\sigma, \sigma' \in \Sigma$  mit  $\sigma(x) = \sigma'(x)$  für alle  $x \in FV(a)$ .

Induktionsanfang:

Fall 1: Sei  $a \equiv n$ ,  $n \in \mathbf{Num}$ .

Mit den Definitionen von  $\llbracket \cdot \rrbracket_A$  und  $\llbracket \cdot \rrbracket_N$  erhalten wir unmittelbar wie gewünscht:

$$\llbracket a \rrbracket_A(\sigma) = \llbracket n \rrbracket_A(\sigma) = \llbracket n \rrbracket_N = \llbracket n \rrbracket_A(\sigma') = \llbracket a \rrbracket_A(\sigma')$$



## Beispiel: Beweis von Lemma 1.4.1 (2)

Fall 2: Sei  $a \equiv x$ ,  $x \in \mathbf{Var}$ .

Mit der Definition von  $\llbracket \_ \rrbracket_A$  erhalten wir auch hier wie gewünscht:

$$\llbracket a \rrbracket_A(\sigma) = \llbracket x \rrbracket_A(\sigma) = \sigma(x) = \sigma'(x) = \llbracket x \rrbracket_A(\sigma') = \llbracket a \rrbracket_A(\sigma')$$

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

**1.5**

1.6

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

49/863

## Beispiel: Beweis von Lemma 1.4.1 (3)

Induktionsschluss:

Fall 3: Sei  $a \equiv a_1 + a_2$ ,  $a_1, a_2 \in \mathbf{Aexpr}$

Wir erhalten wie gewünscht:

$$\begin{aligned} & \llbracket a \rrbracket_A(\sigma) \\ \text{(Wahl von } a) &= \llbracket a_1 + a_2 \rrbracket_A(\sigma) \\ \text{(Def. von } \llbracket \cdot \rrbracket_A) &= \text{plus}(\llbracket a_1 \rrbracket_A(\sigma), \llbracket a_2 \rrbracket_A(\sigma)) \\ \text{(Induktionshypothese für } a_1, a_2) &= \text{plus}(\llbracket a_1 \rrbracket_A(\sigma'), \llbracket a_2 \rrbracket_A(\sigma')) \\ \text{(Def. von } \llbracket \cdot \rrbracket_A) &= \llbracket a_1 + a_2 \rrbracket_A(\sigma') \\ \text{(Wahl von } a) &= \llbracket a \rrbracket_A(\sigma') \end{aligned}$$

Übrige Fälle: Analog.

q.e.d.

# Kapitel 1.6

## Semantikdefinitionsstile

Inhalt

Kap. 1

1.1

1.2

1.3

1.4

1.5

**1.6**

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

51/863

# Motivation

Die **Semantik** einer Programmiersprache kann auf verschiedene Weise festgelegt werden. Wir sprechen hier von unterschiedlichen **Definitionsstilen**. Sie gewähren eine unterschiedliche Sicht auf die Bedeutung der Sprache und richten sich daher implizit an andere Adressaten.

Besonders grundlegend und wichtig sind der



- ▶ **operationelle**
- ▶ **denotationelle**
- ▶ **axiomatische**

Definitionstil.

# Intuition und Ausblick

- ▶ Sprach- und Anwendungsimplementierersicht
  - ▶ Operationelle Semantik
    - ▶ Strukturell operationelle Semantik (small-step semantics)
    - ▶ Natürliche Semantik (big-step semantics)
- ▶ Sprachentwicklersicht
  - ▶ Denotationelle Semantik
- ▶ (Anwendungs-) Programmierer- und Verifizierersicht
  - ▶ Axiomatische Semantik




# Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (1)

-  Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2. Auflage, Springer-V., 2001. (Chapter 9.2, Semantics of Programming Languages)
-  Julien Bertrane, Patrick Cousot, Radhia Cousot, J r me Feret, Laurent Mauborgne, Antoine Min , Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (2)




-  Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.
-  Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*. Communications of the ACM 53(2):66-75, 2010.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (3)

-  Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009. (Chapter 1, Imperative Core; Chapter 1.1, Five Constructs)
-  Gerhard Goos, Wolf Zimmermann. *Programmiersprachen*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 515-562, 2006. (Kapitel 2.2, Elemente von Programmiersprachen: Syntax, Semantik und Pragmatik, Syntaktische Eigenschaften, Semantische Eigenschaften)
-  Steve P. Miller, Michael W. Whalen, Darren D. Cofer. *Software Model Checking Takes Off*. Communications of the ACM 53(2):58-64, 2010.



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 1 (4)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 1, Introduction)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 1, Introduction)
-  Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

# Kapitel 2

## Operationelle Semantik von *WHILE*

Inhalt

Kap. 1

**Kap. 2**

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

58/863

# Operationelle Semantik von *WHILE*

*...die Bedeutung eines (programmiersprachlichen) Konstrukts ist durch die Berechnung beschrieben, die es bei seiner Ausführung auf der Maschine induziert. Wichtig ist, wie der Effekt der Berechnung erzeugt wird.*

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

59/863

# Kapitel 2.1

## Strukturell operationelle Semantik

Inhalt

Kap. 1

Kap. 2

**2.1**

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

60/863

# Strukturell operationelle Semantik (small-step semantics)

*...beschreibt den Ablauf der einzelnen Berechnungsschritte, die stattfinden; daher auch die Bezeichnung **Kleinschritt-Semantik**.*

Inhalt

Kap. 1

Kap. 2

**2.1**

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

61/863

- ▶ Gordon D. Plotkin. [A Structural Approach to Operational Semantics](#). Journal of Logic and Algebraic Programming 60-61:17-139, 2004.
- ▶ Gordon D. Plotkin. [An Operational Semantics for CSP](#). In Proceedings of the TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Hrsg.), North-Holland, Amsterdam, 1982.
- ▶ Gordon D. Plotkin. [A Structural Approach to Operational Semantics](#). Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981, reprinted 1991.

# Strukturell operationelle Semantik (1)

Die **strukturell operationelle Semantik (SO-Semantik)** von *WHILE* (im Sinne von Gordon D. Plotkin)

- ▶ ordnet jedem Programm  $\pi$  als Bedeutung eine partiell definierte **Zustandstransformation**

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- ▶ **Zustandstransformationsfunktional**  $\llbracket \cdot \rrbracket_{\text{SOS}}$  der **strukturell operationellen Semantik**

$$\llbracket \cdot \rrbracket_{\text{SOS}} : \mathbf{Prg} \rightarrow (\Sigma \hookrightarrow \Sigma)$$

definieren wir in der Folge im Detail.

# Strukturell operationelle Semantik (2)

## Intuitiv:

- ▶ Die **SO-Semantik** beschreibt den Berechnungsvorgang von Programmen  $\pi \in \mathbf{Prg}$  als Folge elementarer Speicherzustandsübergänge.

## Zentral dafür:

- ▶ Der Begriff der **Konfiguration!**



# Konfigurationen

Wir unterscheiden:

- ▶ **Nichtterminale** bzw. **(Zwischen-) Konfigurationen**  $\gamma$  der Form  $\langle \pi, \sigma \rangle$ :  
...das (Rest-) Programm  $\pi$  ist auf den (Zwischen-) Zustand  $\sigma$  anzuwenden.
- ▶ **Terminale** bzw. **finale Konfigurationen**  $\gamma$  der Form  $\sigma$ :  
...der Zustand  $\sigma$  ist der Resultatzustand nach Ende der (regulären) Berechnung.

Vereinbarung:

- ▶  $\Gamma \stackrel{df}{=} (\mathbf{Prg} \times \Sigma) \cup \Sigma$  bezeichne die Menge aller Konfigurationen,  $\gamma \in \Gamma$ .

# SOS-Regeln von *WHILE* – Axiome (1)

$$[\text{skip}_{\text{SOS}}] \frac{\overline{\quad}}{\langle \text{skip}, \sigma \rangle \Rightarrow \sigma}$$

$$[\text{ass}_{\text{SOS}}] \frac{\overline{\quad}}{\langle x := t, \sigma \rangle \Rightarrow \sigma[\llbracket t \rrbracket_A(\sigma) / x]}$$

$$[\text{if}^{\text{tt}}_{\text{SOS}}] \frac{\overline{\quad}}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \Rightarrow \langle \pi_1, \sigma \rangle} \llbracket b \rrbracket_B(\sigma) = \mathbf{wahr}$$

$$[\text{if}^{\text{ff}}_{\text{SOS}}] \frac{\overline{\quad}}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \Rightarrow \langle \pi_2, \sigma \rangle} \llbracket b \rrbracket_B(\sigma) = \mathbf{falsch}$$

$$[\text{while}_{\text{SOS}}] \frac{\overline{\quad}}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \Rightarrow \langle \text{if } b \text{ then } \pi; \text{ while } b \text{ do } \pi \text{ od else skip fi}, \sigma \rangle}$$

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

66/863

# SOS-Regeln von *WHILE* – Regeln (2)

$$[\text{comp}_{\text{SOS}}^1] \quad \frac{\langle \pi_1, \sigma \rangle \Rightarrow \langle \pi_1', \sigma' \rangle}{\langle \pi_1; \pi_2, \sigma \rangle \Rightarrow \langle \pi_1'; \pi_2, \sigma' \rangle}$$

$$[\text{comp}_{\text{SOS}}^2] \quad \frac{\langle \pi_1, \sigma \rangle \Rightarrow \sigma'}{\langle \pi_1; \pi_2, \sigma \rangle \Rightarrow \langle \pi_2, \sigma' \rangle}$$

Inhalt

Kap. 1

Kap. 2

**2.1**

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

67/863

# Sprechweisen

Wir unterscheiden:

- ▶ Prämissenlose Regeln, sog. **Axiome**, der Form

$$\frac{\text{---}}{\text{Konklusion}}$$

- ▶ Prämissenbehaftete Regeln, sog. (**echte**) **Regeln**, der Form

$$\frac{\text{Prämisse(n)}}{\text{Konklusion}} \quad [Randbedingung(en)]$$

mit optionalen **Randbedingungen** (**Seitenbedingungen**)

wie z.B. in der Regel  $[if_{SOS}^{ff}]$  in Form von

$\llbracket b \rrbracket_B(\sigma) = \mathbf{falsch}$ .

# Beobachtung

Im Fall der **SO-Semantik** von **WHILE** haben wir also:

- ▶ **5 Axiome**

- ...für die leere Anweisung, Zuweisung, Fallunterscheidung und while-Schleife.

- ▶ **2 Regeln**

- ...für die sequentielle Komposition.

# Berechnungsschritt, Berechnungsfolge

- ▶ Ein **Berechnungsschritt** ist von der Form

$$\langle \pi, \sigma \rangle \equiv \Rightarrow \gamma \quad \text{mit} \quad \gamma \in \Gamma =_{df} (\mathbf{Prg} \times \Sigma) \cup \Sigma$$

- ▶ Eine **Berechnungsfolge** eines Programms  $\pi$  angesetzt auf einen (Start-) Zustand  $\sigma \in \Sigma$  ist
  - ▶ eine **endliche Folge**  $\gamma_0, \dots, \gamma_k$  von Konfigurationen mit  $\gamma_0 = \langle \pi, \sigma \rangle$  und  $\gamma_i \equiv \Rightarrow \gamma_{i+1}$  für alle  $i \in \{0, \dots, k-1\}$ ,
  - ▶ eine **unendliche Folge** von Konfigurationen mit  $\gamma_0 = \langle \pi, \sigma \rangle$  und  $\gamma_i \equiv \Rightarrow \gamma_{i+1}$  für alle  $i \in \mathbb{N}$ .

## Definition (2.1.1, Terminierung und Divergenz)

Eine **maximale** (d.h. nicht mehr verlängerbare) **Berechnungsfolge** heißt

- ▶ **regulär terminierend**, wenn sie endlich ist und die letzte Konfiguration aus  $\Sigma$  ist
- ▶ **divergierend**, falls sie unendlich ist
- ▶ **irregulär terminierend** sonst (z.B. Division durch **0**)

# Beispiel 1(4)

Sei

- ▶  $\sigma \in \Sigma$  mit  $\sigma(x) = 3$
- ▶  $\pi \in \mathbf{Prg}$  mit  
 $\pi \equiv y := 1; \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}$   
(Bemerkung:  $\equiv$  steht für 'syntaktisch identisch')

Bestimme

- ▶ die von der Anfangskonfiguration ' $\pi$  angesetzt auf  $\sigma$ '

$\langle y := 1; \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma \rangle$

induzierte Berechnungsfolge.



## Beispiel 2(4)

- $\langle y := 1; \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma \rangle$
- $\Rightarrow \langle \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[\mathbf{1}/y] \rangle$
- $\Rightarrow \langle \text{ if } x \langle \rangle 1$
- then  $y := y * x; x := x - 1;$
- while  $x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}$
- else *skip* fi,  $\sigma[\mathbf{1}/y] \rangle$
- $\Rightarrow \langle y := y * x; x := x - 1;$
- while  $x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[\mathbf{1}/y] \rangle$
- $\Rightarrow \langle x := x - 1;$
- while  $x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, (\sigma[\mathbf{1}/y])[\mathbf{3}/y] \rangle$
- $( \hat{=} \langle x := x - 1;$
- while  $x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[\mathbf{3}/y] \rangle )$

## Beispiel 3(4)

- $\Rightarrow$   $\langle \text{if } x \langle \rangle 1$   
    then  $y := y * x; x := x - 1;$   
        while  $x \langle \rangle 1$  do  $y := y * x; x := x - 1$  od  
    else *skip* fi,  $(\sigma[3/y])[2/x] \rangle$
- $\Rightarrow$   $\langle y := y * x; x := x - 1;$   
    while  $x \langle \rangle 1$  do  $y := y * x; x := x - 1$  od,  $(\sigma[3/y])[2/x] \rangle$
- $\Rightarrow$   $\langle x := x - 1;$   
    while  $x \langle \rangle 1$  do  $y := y * x; x := x - 1$  od,  $(\sigma[6/y])[2/x] \rangle$
- $\Rightarrow$   $\langle \text{while } x \langle \rangle 1$  do  $y := y * x; x := x - 1$  od,  $(\sigma[6/y])[1/x] \rangle$

## Beispiel 4(4)

$\Rightarrow$   $\langle \text{if } x \langle \rangle 1$   
    then  $y := y * x; x := x - 1;$   
        while  $x \langle \rangle 1$  do  $y := y * x; x := x - 1$  od  
    else *skip* fi,  $(\sigma[\mathbf{6}/y])[\mathbf{1}/x]$   
 $\Rightarrow$   $\langle \text{skip}, (\sigma[\mathbf{6}/y])[\mathbf{1}/x] \rangle$   
 $\Rightarrow$   $(\sigma[\mathbf{6}/y])[\mathbf{1}/x]$

# Detailbetrachtung zum Beispiel 1(3)

$$([ass_{sos}], [comp_{sos}^2]) \Rightarrow \langle y := 1; \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma \rangle \equiv \langle \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle$$

steht vereinfachend für:

$$[comp_{sos}^2] \frac{[ass_{sos}] \frac{\langle y := 1, \sigma \rangle \equiv \sigma[1/y]}{\langle y := 1; \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma \rangle \equiv \langle \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle}}{\langle y := 1; \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma \rangle \equiv \langle \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle}}$$

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

76/863

# Detailbetrachtung zum Beispiel 2(3)

$$[\text{while}_{\text{SOS}}] \equiv \langle \text{while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle$$
$$\begin{aligned} & \langle \text{if } x \langle \rangle 1 \\ & \quad \text{then } y := y * x; x := x - 1; \\ & \quad \quad \text{while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od} \\ & \quad \text{else skip fi}, \sigma[1/y] \rangle \end{aligned}$$

steht vereinfachend für:

$$[\text{while}_{\text{SOS}}] \frac{\text{---}}{\langle \text{while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle \equiv \langle \text{if } x \langle \rangle 1 \text{ then } y := y * x; x := x - 1; \text{ while } x \langle \rangle 1 \text{ do } y := y * x; x := x - 1 \text{ od} \text{ else skip fi}, \sigma[1/y] \rangle}$$

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

77/863

# Detailbetrachtung zum Beispiel 3(3)

$\langle (y := y * x; x := x - 1);$   
 $\text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle$

$([ass_{sos}],$   
 $[comp_{sos}^2],$   
 $[comp_{sos}^1]) \Rightarrow \langle x := x - 1;$   
 $\text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od},$   
 $(\sigma[1/y])[3/y] \rangle$

steht vereinfachend für:

$$\begin{array}{c}
 [comp_{sos}^2] \frac{[ass_{sos}] \frac{\langle y := y * x, \sigma[1/y] \rangle \equiv \langle \sigma[1/y] \rangle [3/y] \rangle}{\langle y := y * x; x := x - 1, \sigma[1/y] \rangle \equiv \langle x := x - 1, (\sigma[1/y])[3/y] \rangle}}{\langle (y := y * x; x := x - 1); \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, \sigma[1/y] \rangle \equiv \langle x := x - 1; \text{while } x <> 1 \text{ do } y := y * x; x := x - 1 \text{ od}, (\sigma[1/y])[3/y] \rangle} \\
 [comp_{sos}^1]
 \end{array}$$

# Determinismus der SOS-Regeln

## Lemma (2.1.2)

$$\forall \pi \in \mathbf{Prg}, \sigma \in \Sigma, \gamma, \gamma' \in \Gamma. \langle \pi, \sigma \rangle \Rightarrow \gamma \wedge \langle \pi, \sigma \rangle \Rightarrow \gamma' \Rightarrow \gamma = \gamma'$$

## Korollar (2.1.3)

Die von den SOS-Regeln für eine Konfiguration induzierte Berechnungsfolge ist eindeutig bestimmt, d.h. *deterministisch*.

Salopper, wenn auch weniger präzise:

Die SO-Semantik von *WHILE* ist deterministisch!

# Das Semantikfunktional $\llbracket \cdot \rrbracket_{SOS}$

Dank [Korollar 2.1.3](#) ist es sinnvoll festzulegen:

## Definition (2.1.4, SO-Semantik)

Die [strukturell operationelle Semantik](#) von *WHILE* ist gegeben durch das Funktional

$$\llbracket \cdot \rrbracket_{SOS} : \mathbf{Prg} \rightarrow (\Sigma \hookrightarrow \Sigma)$$

das definiert ist durch:

$$\forall \pi \in \mathbf{Prg}, \sigma \in \Sigma. \llbracket \pi \rrbracket_{SOS}(\sigma) =_{df} \begin{cases} \sigma' & \text{falls } \langle \pi, \sigma \rangle \equiv^* \sigma' \\ undef & \text{sonst} \end{cases}$$

**Bemerkung:**  $\equiv^*$  bezeichnet die reflexiv-transitive Hülle von  $\equiv$ .

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

80/863



# Variante induktiver Beweisführung

## Induktion über die Länge von Berechnungsfolgen:

- ▶ **Induktionsanfang**
  - ▶ Beweise, dass  $A$  für Berechnungsfolgen der Länge 0 gilt.
- ▶ **Induktionsschritt**
  - ▶ Beweise unter der Annahme, dass  $A$  für Berechnungsfolgen der Länge kleiner oder gleich  $k$  gilt (**Induktionshypothese!**), dass  $A$  auch für Berechnungsfolgen der Länge  $k + 1$  gilt.

# Anwendung

Induktive Beweisführung über die Länge von Berechnungsfolgen ist typisch zum Nachweis von Aussagen über Eigenschaften strukturell operationeller Semantik.

Ein Beispiel hierfür ist der Beweis des folgenden Lemmas:

## Lemma (2.1.5)

$$\forall \pi, \pi' \in \mathbf{Prg}, \sigma, \sigma'' \in \Sigma, k \in \mathbb{N}. (\langle \pi_1; \pi_2, \sigma \rangle \Rightarrow^k \sigma'') \Rightarrow$$
$$\exists \sigma' \in \Sigma, k_1, k_2 \in \mathbb{N}. (k_1 + k_2 = k \wedge \langle \pi_1, \sigma \rangle \Rightarrow^{k_1} \sigma' \wedge \langle \pi_2, \sigma' \rangle \Rightarrow^{k_2} \sigma'')$$

# Kapitel 2.2

## Natürliche Semantik

Inhalt

Kap. 1

Kap. 2

2.1

**2.2**

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Natürliche Semantik (big-step semantics)

*...beschreibt wie sich das Gesamtergebnis der Programmausführung ergibt; daher auch die Bezeichnung **Großschritt-Semantik**.*

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

84/863

# Natürliche Semantik (1)

Die natürliche Semantik (N-Semantik) von *WHILE*

- ▶ ordnet jedem Programm  $\pi$  als Bedeutung eine partiell definierte **Zustandstransformation**

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- ▶ **Zustandstransformationsfunktional**  $\llbracket \cdot \rrbracket_{ns}$  der natürlichen Semantik

$$\llbracket \cdot \rrbracket_{ns} : \mathbf{Prg} \rightarrow (\Sigma \hookrightarrow \Sigma)$$

definieren wir in der Folge im Detail.

# Natürliche Semantik (2)

Intuitiv:

- ▶ Die **N-Semantik** ist am Zusammenhang zwischen **initialem** und **finalem** Speicherzustand einer Berechnung eines Programms  $\pi \in \mathbf{Prg}$  interessiert.

Zentral auch hier:

- ▶ Der von der SO-Semantik bekannte Begriff der **Konfiguration**.

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

86/863

# NS-Regeln von *WHILE* (1)

$$[\text{skip}_{ns}] \quad \frac{\text{---}}{\langle \text{skip}, \sigma \rangle \rightarrow \sigma}$$

$$[\text{ass}_{ns}] \quad \frac{\text{---}}{\langle x := t, \sigma \rangle \rightarrow \sigma[\llbracket t \rrbracket_A(\sigma) / x]}$$

$$[\text{comp}_{ns}] \quad \frac{\langle \pi_1, \sigma \rangle \rightarrow \sigma', \langle \pi_2, \sigma' \rangle \rightarrow \sigma''}{\langle \pi_1; \pi_2, \sigma \rangle \rightarrow \sigma''}$$

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

87/863

# NS-Regeln von WHILE (2)

Inhalt

Kap. 1

Kap. 2

2.1

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

88/863

$$[\text{if}_{ns}^{tt}] \quad \frac{\langle \pi_1, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow \sigma'}$$

$$\llbracket b \rrbracket_B(\sigma) = \text{wahr}$$

$$[\text{if}_{ns}^{ff}] \quad \frac{\langle \pi_2, \sigma \rangle \rightarrow \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow \sigma'}$$

$$\llbracket b \rrbracket_B(\sigma) = \text{falsch}$$

$$[\text{while}_{ns}^{tt}] \quad \frac{\langle \pi, \sigma \rangle \rightarrow \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \rightarrow \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow \sigma''}$$

$$\llbracket b \rrbracket_B(\sigma) = \text{wahr}$$

$$[\text{while}_{ns}^{ff}] \quad \frac{\text{—}}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow \sigma}$$

$$\llbracket b \rrbracket_B(\sigma) = \text{falsch}$$





# Beispiel 2(2)

Das gleiche Beispiel in "etwas" gefälligerer Darstellung:

$$\begin{array}{c}
 \frac{[\text{comp}_m] \frac{[\text{ass}_m] \frac{}{\langle y := 1, \sigma \rangle \longrightarrow \sigma [1/y]} \quad [\text{while}_m^{\text{ff}}] \frac{[\text{comp}_m] \frac{[\text{ass}_m] \frac{}{\langle y := y^*x, \sigma [1/y] \rangle \longrightarrow \sigma [3/y]} \quad [\text{ass}_m] \frac{}{\langle x := x-1, \sigma [3/y] \rangle \longrightarrow \sigma [3/y] [2/x]} \quad \langle y := y^*x; x := x-1, \sigma [1/y] \rangle \longrightarrow \sigma [3/y] [2/x]}{\langle \text{while } x < 1 \text{ do } y := y^*x; x := x-1 \text{ od}, \sigma [1/y] \rangle \longrightarrow \sigma [6/y] [1/x]} \quad \text{T}}{\langle y := 1; \text{while } x < 1 \text{ do } y := y^*x; x := x-1 \text{ od}, \sigma \rangle \longrightarrow \sigma [6/y] [1/x]}
 \end{array}$$

T ≡

$$\begin{array}{c}
 \frac{[\text{while}_m^{\text{ff}}] \frac{[\text{comp}_m] \frac{[\text{ass}_m] \frac{}{\langle y := y^*x, \sigma [3/y] [2/x] \rangle \longrightarrow \sigma [6/y] [2/x]} \quad [\text{ass}_m] \frac{}{\langle x := x-1, \sigma [6/y] [2/x] \rangle \longrightarrow \sigma [6/y] [1/x]} \quad \langle y := y^*x; x := x-1, \sigma [3/y] [2/x] \rangle \longrightarrow \sigma [6/y] [1/x]}{\langle \text{while } x < 1 \text{ do } y := y^*x; x := x-1 \text{ od}, \sigma [6/y] [1/x] \rangle \longrightarrow \sigma [6/y] [1/x]} \quad [\text{while}_m^{\text{ff}}] \frac{}{\langle \text{while } x < 1 \text{ do } y := y^*x; x := x-1 \text{ od}, \sigma [6/y] [1/x] \rangle \longrightarrow \sigma [6/y] [1/x]}}{\langle \text{while } x < 1 \text{ do } y := y^*x; x := x-1 \text{ od}, \sigma [3/y] [2/x] \rangle \longrightarrow \sigma [6/y] [1/x]}
 \end{array}$$

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

90/863

# Determinismus der NS-Regeln

## Lemma (2.2.1)

$\forall \pi \in \mathbf{Prg}, \sigma \in \Sigma, \gamma, \gamma' \in \Gamma. \langle \pi, \sigma \rangle \rightarrow \gamma \wedge \langle \pi, \sigma \rangle \rightarrow \gamma' \Rightarrow \gamma = \gamma'$

## Korollar (2.2.2)

*Die von den NS-Regeln für eine Konfiguration induzierte finale Konfiguration ist (sofern definiert) eindeutig bestimmt, d.h. deterministisch.*

Salopper, wenn auch weniger präzise:

Die N-Semantik von *WHILE* ist deterministisch!

Inhalt

Kap. 1

Kap. 2

2.1

2.2

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

91/863

# Das Semantikfunktional $\llbracket \cdot \rrbracket_{ns}$

Dank [Korollar 2.2.2](#) ist es sinnvoll festzulegen:

## Definition (2.2.3, N-Semantik)

Die [natürliche Semantik](#) von *WHILE* ist gegeben durch das Funktional

$$\llbracket \cdot \rrbracket_{ns} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$$

das definiert ist durch:

$$\forall \pi \in \mathbf{Prg}, \sigma \in \Sigma. \llbracket \pi \rrbracket_{ns}(\sigma) \stackrel{df}{=} \begin{cases} \sigma' & \text{falls } \langle \pi, \sigma \rangle \rightarrow \sigma' \\ \text{undef} & \text{sonst} \end{cases}$$

# Variante induktiver Beweisführung

## Induktion über die Form von Ableitungsbäumen:

- ▶ **Induktionsanfang**
  - ▶ Beweise, dass  $A$  für die Axiome des Transitionssystems gilt (und somit für alle nichtzusammengesetzten Ableitungsbäume).
- ▶ **Induktionsschritt**
  - ▶ Beweise für jede echte Regel des Transitionssystems unter der Annahme, dass  $A$  für jede Prämisse dieser Regel gilt (**Induktionshypothese!**), dass  $A$  auch für die Konklusion dieser Regel gilt, sofern die (optional vorhandenen) Randbedingungen der Regel erfüllt sind.

# Anwendung

Induktive Beweisführung über die Form von Ableitungsbäumen ist typisch zum Nachweis von Aussagen über Eigenschaften natürlicher Semantik.

Ein Beispiel hierfür ist der Beweis von Lemma 2.2.1!

Inhalt

Kap. 1

Kap. 2

2.1

**2.2**

2.3

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

94/863

# Kapitel 2.3

## Strukturell operationelle und natürliche Semantik im Vergleich

# Strukturell operationelle Semantik

Der Fokus liegt auf

- ▶ **individuellen Schritten** einer Berechnungsfolge, d.h. auf der Ausführung von Zuweisungen und Tests

Intuitive Bedeutung der **Transitionsrelation**  $\langle \pi, \sigma \rangle \Rightarrow \gamma$  mit  $\gamma$  von der Form  $\langle \pi', \sigma' \rangle$  oder  $\sigma'$ : Die **Transition** beschreibt den **ersten** Schritt d. Berechnungsf. v.  $\pi$  angesetzt auf  $\sigma$ .

Dabei sind folgende **Übergänge** möglich:

- ▶  $\gamma$  ist von der Form  $\langle \pi', \sigma' \rangle$ :  
Die Abarbeitung von  $\pi$  ist nicht vollständig; das Restprogramm  $\pi'$  ist auf  $\sigma'$  anzusetzen. Ist von  $\langle \pi', \sigma' \rangle$  kein Transitionsübergang möglich (z.B. Division durch **0**), so terminiert die Abarbeitung von  $\pi$  in  $\langle \pi', \sigma' \rangle$  **irregulär**.
- ▶  $\gamma$  ist von der Form  $\sigma'$ :  
Die Abarbeitung von  $\pi$  ist vollständig;  $\pi$  angesetzt auf  $\sigma$  terminiert in einem Schritt in  $\sigma'$  **regulär**.



Der Fokus liegt auf

- ▶ Zusammenhang von **initialem** und **finalelem** Zustand einer Berechnungsfolge

Intuitive Bedeutung von

$$\langle \pi, \sigma \rangle \rightarrow \gamma$$

mit  $\gamma$  von der Form  $\sigma'$  ist:  $\pi$  angesetzt auf initialen Zustand  $\sigma$  terminiert schließlich im finalen Zustand  $\sigma'$ . Existiert ein solches  $\sigma'$  nicht, so ist die N-Semantik undefiniert für den initialen Zustand  $\sigma$ .

# Zusammenhang von $\llbracket \cdot \rrbracket_{SOS}$ und $\llbracket \cdot \rrbracket_{NS}$

## Lemma (2.3.1)

$$\forall \pi \in \mathbf{Prg}, \forall \sigma, \sigma' \in \Sigma. \langle \pi, \sigma \rangle \rightarrow \sigma' \Rightarrow \langle \pi, \sigma \rangle \equiv^* \sigma'$$

**Beweis** durch Induktion über den Aufbau des Ableitungsbaums für  $\langle \pi, \sigma \rangle \rightarrow \sigma'$ .

## Lemma (2.3.2)

$$\forall \pi \in \mathbf{Prg}, \forall \sigma, \sigma' \in \Sigma, \forall k \in \mathbf{N}. \langle \pi, \sigma \rangle \equiv^k \sigma' \Rightarrow \langle \pi, \sigma \rangle \rightarrow \sigma'$$

**Beweis** durch Induktion über die Länge der Ableitungsfolge  $\langle \pi, \sigma \rangle \equiv^k \sigma'$ , d.h. durch Induktion über  $k$ .



# Äquivalenz von SO- und N-Semantik

Aus Lemma 2.3.1 und Lemma 2.3.2 folgt:




Theorem (2.3.3, Äquivalenz von  $\llbracket \cdot \rrbracket_{sos}$  und  $\llbracket \cdot \rrbracket_{ns}$ )

$$\forall \pi \in \mathbf{Prg}. \llbracket \pi \rrbracket_{sos} = \llbracket \pi \rrbracket_{ns}$$

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (1)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 2, Operational Semantics)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 2, Operational Semantics)

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 2 (2)

-  Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981, reprinted 1991.
-  Gordon D. Plotkin. *An Operational Semantics for CSP*. In Proceedings of TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Ed.), North-Holland, Amsterdam, 1982.
-  Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:17-139, 2004.

# Kapitel 3

## Denotationelle Semantik von *WHILE*

Inhalt

Kap. 1

Kap. 2

**Kap. 3**

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Denotationelle Semantik von *WHILE*

*...die Bedeutung eines Konstrukts wird durch mathematische Objekte modelliert, die den Effekt der Ausführung der Konstrukte repräsentieren. Wichtig ist **einzig** der Effekt, nicht wie er bewirkt wird.*

Inhalt

Kap. 1

Kap. 2

**Kap. 3**

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Denotationelle Semantik

Die **denotationelle Semantik (D-Semantik)** von **WHILE**

- ▶ ordnet jedem Programm  $\pi$  als Bedeutung eine partiell definierte **Zustandstransformation**

$$\Sigma \hookrightarrow \Sigma$$

zu.

Das zugehörige

- ▶ **Zustandstransformationsfunktional**  $\llbracket \cdot \rrbracket_{ds}$  der **denotationellen Semantik**

$$\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \hookrightarrow \Sigma)$$

definieren wir in der Folge im Detail.



# Vergleich operationeller und denotationeller Semantik

- ▶ **Operationelle** Semantik  
...der Fokus liegt darauf, **wie** ein Programm ausgeführt wird.
- ▶ **Denotationelle** Semantik  
...der Fokus liegt auf dem **Effekt**, den die Ausführung eines Programms hat: Für jedes **syntaktische** Konstrukt gibt es eine **semantische** Funktion, die ersterem ein **mathematisches Objekt** zuweist, d.h. eine Funktion, die den Effekt der Ausführung des Konstrukts beschreibt (jedoch nicht, wie dieser Effekt erreicht wird).

Inhalt

Kap. 1

Kap. 2

**Kap. 3**

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

105/863

# Kompositionalität

Zentral für denotationelle Semantiken: **Kompositionalität!**

**Intuitiv:**

- ▶ Für jedes Element der elementaren syntaktischen Konstrukte/Kategorien gibt es eine zugehörige semantische Funktion.
- ▶ Für jedes Element eines zusammengesetzten syntaktischen Konstrukts/Kategorie gibt es eine semantische Funktion, die über die semantischen Funktionen der Komponenten des zusammengesetzten Konstrukts definiert ist.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

106/863

# Kapitel 3.1

## Denotationelle Semantik

Inhalt

Kap. 1

Kap. 2

Kap. 3

**3.1**

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Def. Gleichungen der denotationellen Semantik

$$\llbracket \text{skip} \rrbracket_{ds} = Id$$

$$\llbracket x := t \rrbracket_{ds}(\sigma) = \sigma[\llbracket t \rrbracket_A(\sigma)/x]$$

$$\llbracket \pi_1; \pi_2 \rrbracket_{ds} = \llbracket \pi_2 \rrbracket_{ds} \circ \llbracket \pi_1 \rrbracket_{ds}$$

$$\llbracket \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi} \rrbracket_{ds} = \text{cond}(\llbracket b \rrbracket_B, \llbracket \pi_1 \rrbracket_{ds}, \llbracket \pi_2 \rrbracket_{ds})$$

$$\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} = \text{FIX } F$$

$$\text{mit } F g = \text{cond}(\llbracket b \rrbracket_B, g \circ \llbracket \pi \rrbracket_{ds}, Id)$$

Dabei bezeichnen *FIX* das sog. **Zustandstransformationsfunktional** und  $Id : \Sigma \rightarrow \Sigma$  die **identische Zustandstransformation** definiert durch:  $\forall \sigma \in \Sigma. Id(\sigma) =_{df} \sigma$

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

108/863

# Das Hilfsfunktional cond: Fallunterscheidung

Funktionalität:

$$\text{cond} : (\Sigma \hookrightarrow \text{IB}) \times (\Sigma \hookrightarrow \Sigma) \times (\Sigma \hookrightarrow \Sigma) \rightarrow (\Sigma \hookrightarrow \Sigma)$$

definiert durch

$$\text{cond}(p, g_1, g_2) \sigma =_{df} \begin{cases} g_1 \sigma & \text{falls } p \sigma = \mathbf{wahr} \\ g_2 \sigma & \text{falls } p \sigma = \mathbf{falsch} \\ \text{undef} & \text{sonst} \end{cases}$$

Bemerkung zu den Argumenten und zum Resultat von *cond*:

- ▶ 1. Argument: Prädikat (in unserem Kontext partiell definiert; siehe Kapitel 1.3)
- ▶ 2. und 3. Argument: Je eine partiell definierte Zustands-  
transformation
- ▶ Resultat: Ebenfalls eine partiell definierte Zustandstrans-  
formation

# Daraus ergibt sich

...für die Bedeutung der Fallunterscheidung:

$$\begin{aligned} & \llbracket \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi} \rrbracket_{ds} \sigma \\ &= \text{cond}(\llbracket b \rrbracket_B, \llbracket \pi_1 \rrbracket_{ds}, \llbracket \pi_2 \rrbracket_{ds}) \sigma \\ &= \begin{cases} \sigma' & \text{falls } (\llbracket b \rrbracket_B \sigma = \mathbf{wahr} \wedge \llbracket \pi_1 \rrbracket_{ds} \sigma = \sigma') \\ & \vee (\llbracket b \rrbracket_B \sigma = \mathbf{falsch} \wedge \llbracket \pi_2 \rrbracket_{ds} \sigma = \sigma') \\ \text{undef} & \text{falls } (\llbracket b \rrbracket_B \sigma = \text{undef}) \\ & \vee (\llbracket b \rrbracket_B \sigma = \mathbf{wahr} \wedge \llbracket \pi_1 \rrbracket_{ds} \sigma = \text{undef}) \\ & \vee (\llbracket b \rrbracket_B \sigma = \mathbf{falsch} \wedge \llbracket \pi_2 \rrbracket_{ds} \sigma = \text{undef}) \end{cases} \end{aligned}$$

## Erinnerung:

- ▶ In unserem Szenario sind  $\llbracket \cdot \rrbracket_A$  und  $\llbracket \cdot \rrbracket_B$  partiell definiert (z.B. Division durch  $\mathbf{0}$ ).

# Das Hilfsfunktional $FIX$

Funktionalität:

$$FIX : ((\Sigma \hookrightarrow \Sigma) \rightarrow (\Sigma \hookrightarrow \Sigma)) \rightarrow (\Sigma \hookrightarrow \Sigma)$$

Im Zshg. mit  $\llbracket \cdot \rrbracket_{ds}$  wird  $FIX$  angewendet auf das Funktional

$$F : (\Sigma \hookrightarrow \Sigma) \rightarrow (\Sigma \hookrightarrow \Sigma)$$

das definiert ist durch:

$$F g = \text{cond}(\llbracket b \rrbracket_B, g \circ \llbracket \pi \rrbracket_{ds}, Id)$$

mit  $b$  und  $\pi$  aus dem Kontext

$$\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$$

Beachte:

- ▶  $FIX$  ist ein Funktional, das  
sog. **Zustandstransformationsfunktional**.

# Zustandstransformationsfunktional FIX

1. Offenbar müssen  $\text{if } b \text{ then } (\pi; \text{while } b \text{ do } \pi \text{ od}) \text{ else skip fi}$  und  $\text{while } b \text{ do } \pi \text{ od}$  denselben Effekt haben.

2. Somit gilt:  $\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} =$

$$\text{cond}(\llbracket b \rrbracket_B, \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} \circ \llbracket \pi \rrbracket_{ds}, \text{Id})$$

3. Anwenden von  $F$  auf  $\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$  liefert gleichfalls:  $F \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} = df$

$$\text{cond}(\llbracket b \rrbracket_B, \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} \circ \llbracket \pi \rrbracket_{ds}, \text{Id})$$

4. Aus 2.) und 3.) folgt damit:

$$F \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} = \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$$

Das heißt: Die denotationelle Semantik der while-Schleife ist ein Fixpunkt des Funktionals  $F$ .

In der Folge werden wir sehen: Der **kleinste Fixpunkt!**

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

112/863



# 'Vernünftigkeit' der definierenden Gleichungen der denotationellen Semantik

## Lemma (3.1.1)

*Für alle  $\pi \in \mathbf{Prg}$  ist durch die "Definierenden Gleichungen der denotationellen Semantik" von der gleichnamigen Folie in eindeutiger Weise eine (partielle) Funktion definiert.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

113/863

# Das Semantikfunktional $\llbracket \cdot \rrbracket_{ds}$

Dank [Lemma 3.1.1](#) ist es sinnvoll festzulegen:

## Definition (3.1.2, D-Semantik)

Die [denotationelle Semantik](#) von *WHILE* ist gegeben durch das Funktional

$$\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \hookrightarrow \Sigma)$$

wobei für alle  $\pi \in \mathbf{Prg}$  die Funktion  $\llbracket \pi \rrbracket_{ds}$  diejenige gemäß [Lemma 3.1.1](#) gegebene (partielle) Funktion ist.

# Zusammenhang von $\llbracket \cdot \rrbracket_{ds}$ und $\llbracket \cdot \rrbracket_{sos}$

## Lemma (3.1.2)

$$\forall \pi \in \mathbf{Prg}. \llbracket \pi \rrbracket_{ds} \sqsubseteq \llbracket \pi \rrbracket_{sos}$$

**Beweis** durch strukturelle Induktion über den induktiven Aufbau von  $\pi$ .

## Lemma (3.1.3)

$$\forall \pi \in \mathbf{Prg}. \llbracket \pi \rrbracket_{sos} \sqsubseteq \llbracket \pi \rrbracket_{ds}$$

**Beweis** Zeige, dass für alle  $\sigma, \sigma' \in \Sigma$  gilt:

$$\langle \pi, \sigma \rangle \equiv^* \sigma' \Rightarrow \llbracket \pi \rrbracket_{ds}(\sigma) = \sigma'.$$

# Äquivalenz von $\llbracket \cdot \rrbracket_{ds}$ und $\llbracket \cdot \rrbracket_{sos}$

Korollar (3.1.4, Äquivalenz von  $\llbracket \cdot \rrbracket_{ds}$  und  $\llbracket \cdot \rrbracket_{sos}$ )

$$\forall \pi \in \mathbf{Prg}. \llbracket \pi \rrbracket_{ds} = \llbracket \pi \rrbracket_{sos}$$

# Äquivalenz von $\llbracket \pi \rrbracket_{ds}$ , $\llbracket \pi \rrbracket_{sos}$ und $\llbracket \pi \rrbracket_{ns}$

Aus Theorem 3.1.4 und Theorem 2.3.3 folgt:

Theorem (3.1.5)

$$\forall \pi \in \mathbf{Prg}. \llbracket \pi \rrbracket_{ds} = \llbracket \pi \rrbracket_{sos} = \llbracket \pi \rrbracket_{ns}$$

# Fazit

Die Äquivalenz der strukturell operationellen, natürlichen und denotationellen Semantik von *WHILE* legt es nahe, den semantikangebenden Index in der Folge fortzulassen und vereinfachend von  $\llbracket \cdot \rrbracket$  als **der** Semantik der Sprache *WHILE* zu sprechen:

$$\llbracket \cdot \rrbracket : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$$

definiert durch (z.B.)

$$\llbracket \cdot \rrbracket =_{df} \llbracket \cdot \rrbracket_{sos}$$

# Kapitel 3.2

## Fixpunktfunktional

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

**3.2**

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Zur Bedeutung von $FIX F$

In der Folge wollen wir die Bedeutung von

▶  $FIX F$

genauer untersuchen und aufklären.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

**3.2**

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16



# Zur Bedeutung von FIX F

## Erinnerung:

Hilfsfunktional  $FIX$  mit Funktionalität:

$$FIX : ((\Sigma \leftrightarrow \Sigma) \rightarrow (\Sigma \leftrightarrow \Sigma)) \rightarrow (\Sigma \leftrightarrow \Sigma)$$

Im Zshg. mit  $\llbracket \cdot \rrbracket_{ds}$  wird  $FIX$  angewendet auf das Funktional

$$F : (\Sigma \leftrightarrow \Sigma) \rightarrow (\Sigma \leftrightarrow \Sigma)$$

das definiert ist durch:

$$F g = \text{cond}(\llbracket b \rrbracket_B, g \circ \llbracket \pi \rrbracket_{ds}, Id)$$

mit  $b$  und  $\pi$  aus dem Kontext

$$\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$$

Wir wollen zeigen:

- ▶ Die denotationelle Semantik der while-Schleife ist der kleinste Fixpunkt des Funktionals  $F$ .

# Dazu folgende Erinnerung

Aus der Beobachtung, dass

- ▶  $\text{while } b \text{ do } \pi \text{ od}$  dieselbe Bedeutung haben muss wie  $\text{if } b \text{ then } (\pi; \text{while } b \text{ do } \pi \text{ od}) \text{ else skip fi}$

folgt die Gleichheit von:

- ▶  $\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} = \text{cond}(\llbracket b \rrbracket_B, \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds} \circ \llbracket \pi \rrbracket_{ds}, \text{Id})$

Zusammen mit der Definition von  $F$  folgt damit:

- ▶  $\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$  muss Fixpunkt des Funktionals  $F$  sein, das definiert ist durch

$$F g = \text{cond}(\llbracket b \rrbracket_B, g \circ \llbracket \pi \rrbracket_{ds}, \text{Id})$$

Das heißt, es muss gelten:

$$F(\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}) = \llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$$

Dies führt uns zu einer **kompositionellen** Definition von  $\llbracket \text{while } b \text{ do } \pi \text{ od} \rrbracket_{ds}$  und damit insgesamt von  $\llbracket \cdot \rrbracket_{ds}$ .

# Unser Arbeitsplan

## Wir benötigen:

- ▶ Einige Resultate aus der [Fixpunkttheorie](#).

## Diese erlauben uns:

- ▶ Nachzuweisen, dass diese Resultate auf unsere Situation anwendbar sind.

## Dabei wird vorausgesetzt:

- ▶ Mathematische Grundlagen über Ordnungen, CPOs, Stetigkeit von Funktionen und benötigte Resultate darüber, insbesondere das Fixpunkttheorem.

Diese Grundlagen sind in Kapitel 3.3 (und umfangreicher im Anhang) zusammengefasst.

# Folgende drei Eigenschaften

...werdend entscheidend sein:

1.  $[\Sigma \leftrightarrow \Sigma]$  kann vollständig partiell geordnet werden.
2.  $F$  ist im Anwendungskontext stetig.
3. Fixpunktbildung wird im Anwendungskontext ausschließlich auf stetige Funktionen angewendet.

Insgesamt folgt aus diesen Eigenschaften die **Wohldefiniertheit** von

$$\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$$

# Ordnung auf Zustandstransformationen

Bezeichne

- ▶  $[\Sigma \hookrightarrow \Sigma]$  die Menge der partiell definierten Zustands-  
transformationen.

Wir definieren folgende Ordnung(srelation) auf  $[\Sigma \hookrightarrow \Sigma]$ :

$$g_1 \sqsubseteq g_2 \iff \forall \sigma \in \Sigma. g_1 \sigma \text{ definiert} = \sigma' \Rightarrow g_2 \sigma \text{ definiert} = \sigma'$$

mit  $g_1, g_2 \in [\Sigma \hookrightarrow \Sigma]$

## Lemma (3.2.1)

1.  $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$  ist eine partielle Ordnung.
2. Die total undefinierte (d.h. nirgends definierte) Funktion  $\perp : \Sigma \hookrightarrow \Sigma$  mit  $\perp \sigma = \text{undef}$  für alle  $\sigma \in \Sigma$  ist kleinstes Element in  $([\Sigma \hookrightarrow \Sigma], \sqsubseteq)$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

125/863

# Ordnung auf Zustandstransformationen

Es gilt sogar:

## Lemma (3.2.2)

*Das Paar  $([\Sigma \leftrightarrow \Sigma], \sqsubseteq)$  ist eine vollständige partielle Ordnung (CPO) mit kleinstem Element  $\perp$ .*

Weiters gilt:

Die kleinste obere Schranke  $\bigsqcup Y$  einer Kette  $Y$  ist gegeben durch

$$\text{graph}(\bigsqcup Y) = \bigcup \{\text{graph}(g) \mid g \in Y\}$$

Das heißt:  $(\bigsqcup Y) \sigma = \sigma' \iff \exists g \in Y. g \sigma = \sigma'$

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

126/863

# Graph einer totalen Funktion

## Definition (3.2.3, Graph einer totalen Funktion)

Der **Graph** einer totalen Funktion  $f : M \rightarrow N$  ist definiert durch

$$\text{graph}(f) =_{df} \{ \langle m, n \rangle \in M \times N \mid f m = n \}$$

Für totale Funktionen gilt:

- ▶  $\langle m, n \rangle \in \text{graph}(f) \wedge \langle m, n' \rangle \in \text{graph}(f) \Rightarrow n = n'$   
(**rechtseindeutig**)
- ▶  $\forall m \in M. \exists n \in N. \langle m, n \rangle \in \text{graph}(f)$  (**linkstotal**)

# Graph einer partiellen Funktion

## Definition (3.2.4, Graph einer partiellen Funktion)

Der **Graph** einer partiellen Funktion  $f : M \hookrightarrow N$  mit Definitionsbereich  $M_f \subseteq M$  ist definiert durch

$$\mathit{graph}(f) =_{df} \{ \langle m, n \rangle \in M \times N \mid f\ m = n \wedge m \in M_f \}$$

**Vereinbarung:** Für  $f : M \hookrightarrow N$  partiell definierte Funktion auf  $M_f \subseteq M$  schreiben wir

- ▶  $f\ m = n$ , falls  $\langle m, n \rangle \in \mathit{graph}(f)$
- ▶  $f\ m = \mathit{undef}$ , falls  $m \notin M_f$



# Stetigkeitsresultate (1)

## Lemma (3.2.5)

Sei  $g_0 \in [\Sigma \hookrightarrow \Sigma]$ , sei  $p \in [\Sigma \hookrightarrow \mathbb{B}]$  und sei  $F$  definiert durch

$$F g = \text{cond}(p, g, g_0)$$

Dann gilt:  $F$  ist stetig.

## Vorgriff auf Kapitel 3.3:

Seien  $(C, \sqsubseteq_C)$  und  $(D, \sqsubseteq_D)$  zwei CPOs und sei  $f : C \rightarrow D$  eine Funktion von  $C$  nach  $D$ .

Dann heißt  $f$

- ▶ **monoton** gdw.  $\forall c, c' \in C. c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$   
(Erhalt der Ordnung der Elemente)
- ▶ **stetig** gdw.  $\forall C' \subseteq C. f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$   
(Erhalt der kleinsten oberen Schranken)

# Stetigkeitsresultate (2)

## Lemma (3.2.6)

Sei  $g_0 \in [\Sigma \hookrightarrow \Sigma]$  und sei  $F$  definiert durch

$$F g = g \circ g_0$$

Dann gilt:  $F$  ist stetig.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

**3.2**

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Insgesamt

Zusammen mit

Lemma (3.2.7)

Die Gleichungen zur Festlegung der denotationellen Semantik von *WHILE* definieren eine totale Funktion

$$\llbracket \cdot \rrbracket_{ds} \in [\mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)]$$

sind wir durch!

Wir können beweisen:

Theorem (3.2.8, Wohldefiniertheit von  $\llbracket \cdot \rrbracket_{ds}$ )

$$\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$$

ist *wohldefiniert!*

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Somit wie anfangs angedeutet

Aus

1. Die Menge  $[\Sigma \leftrightarrow \Sigma]$  der partiell definierten Zustands-  
transformationen bildet zusammen mit der Ordnung  $\sqsubseteq$   
eine CPO.
2. Funktional  $F$  mit " $F g = \text{cond}(p, g, g_0)$ " und " $g \circ g_0$ "  
sind stetig
3. In der Definition von  $\llbracket \cdot \rrbracket_{ds}$  wird die Fixpunktbildung  
ausschließlich auf stetige Funktionen angewendet.

...ergibt sich wie gewünscht die **Wohldefiniertheit** von:

$$\llbracket \cdot \rrbracket_{ds} : \mathbf{Prg} \rightarrow (\Sigma \leftrightarrow \Sigma)$$

# Kapitel 3.3

## Ordnungen, Verbände, Fixpunkttheoreme

# Mathematische Grundlagen

...im Zusammenhang mit der

- ▶ Definition der denotationellen Semantik von *WHILE* im Detail

sowie in späteren Kapiteln mit der

- ▶ Definition abstrakter Semantiken für Programmanalysen

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

**3.3**

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Von besonderer Wichtigkeit

...sind:

- ▶ Mengen, Relationen, Verbände
- ▶ Partielle und vollständige partielle Ordnungen
- ▶ Schranken, Fixpunkte, Fixpunkttheoreme

**Bemerkung:** Für eine umfangreichere Zusammenstellung siehe Anhang.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

**3.3**

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

135/863

# Mengen und Relationen

Sei  $M$  eine Menge,  $R$  eine Relation auf  $M$ , d.h.  $R \subseteq M \times M$ .

Dann heißt  $R$

- ▶ **reflexiv** gdw.  $\forall m \in M. m R m$
- ▶ **transitiv** gdw.  $\forall m, n, p \in M. m R n \wedge n R p \Rightarrow m R p$
- ▶ **antisymmetrisch**  
gdw.  $\forall m, n \in M. m R n \wedge n R m \Rightarrow m = n$



# Quasiordnungen und partielle Ordnungen

Eine Relation  $R$  auf  $M$  heißt

- ▶ **Quasiordnung** gdw.  $R$  ist reflexiv und transitiv
- ▶ **partielle Ordnung** gdw.  $R$  ist reflexiv, transitiv und antisymmetrisch

...eine **partielle Ordnung** ist also eine **antisymmetrische Quasiordnung**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

**3.3**

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

137/863

# Schranken und kleinste und größte Elemente

Sei  $(Q, \sqsubseteq)$  eine Quasiordnung, sei  $q \in Q$  und  $Q' \subseteq Q$ .

Dann heißt  $q$

- ▶ **obere (untere) Schranke** von  $Q'$ , in Zeichen:  
 $Q' \sqsubseteq q$  ( $q \sqsubseteq Q'$ ), wenn für alle  $q' \in Q'$  gilt:  
 $q' \sqsubseteq q$  ( $q \sqsubseteq q'$ )
- ▶ **kleinste obere (größte untere) Schranke** von  $Q'$ , wenn  $q$   
obere (untere) Schranke von  $Q'$  ist und für jede andere  
obere (untere) Schranke  $\hat{q}$  von  $Q'$  gilt:  $q \sqsubseteq \hat{q}$  ( $\hat{q} \sqsubseteq q$ )
- ▶ **größtes (kleinstes) Element** von  $Q$ , wenn gilt:  
 $Q \sqsubseteq q$  ( $q \sqsubseteq Q$ )

# Eindeutigkeit von Schranken und von kleinsten und größten Elementen

- ▶ In partiellen Ordnungen sind kleinste obere und größte untere Schranken eindeutig bestimmt, wenn sie existieren.
- ▶ Existenz (und damit Eindeutigkeit) vorausgesetzt, wird die kleinste obere (größte untere) Schranke einer Menge  $P' \subseteq P$  der Grundmenge einer partiellen Ordnung  $(P, \sqsubseteq)$  mit  $\sqcup P'$  ( $\sqcap P'$ ) bezeichnet. Man spricht dann auch vom **Supremum** und **Infimum** von  $P'$ .
- ▶ Analog für kleinste und größte Elemente. Existenz vorausgesetzt, werden sie üblicherweise mit  $\perp$  und  $\top$  bezeichnet.
- ▶ Beachte: Für Quasiordnungen gelten die vorstehenden Aussagen nicht.

# Verbände und vollständige Verbände

Sei  $(P, \sqsubseteq)$  eine partielle Ordnung.

Dann heißt  $(P, \sqsubseteq)$

- ▶ **Verband**, wenn jede **endliche** Teilmenge  $P'$  von  $P$  eine kleinste obere und eine größte untere Schranke in  $P$  besitzt
- ▶ **vollständiger Verband**, wenn **jede** Teilmenge  $P'$  von  $P$  eine kleinste obere und eine größte untere Schranke in  $P$  besitzt

...(vollständige) Verbände sind also spezielle partielle Ordnungen.

# Vollständige partielle Ordnungen

...ein etwas schwächerer, aber in der Informatik oft ausreichender und daher angemessenerer Begriff.

Sei  $(P, \sqsubseteq)$  eine partielle Ordnung.

Dann heißt  $(P, \sqsubseteq)$

- ▶ **vollständig**, kurz **CPO** (von engl. **complete partial order**), wenn jede aufsteigende Kette  $K \subseteq P$  eine kleinste obere Schranke in  $P$  besitzt.

Es gilt:

- ▶ Eine CPO  $(C, \sqsubseteq)$  (genauer wäre: “**kettenvollständige partielle Ordnung** (engl. **chain complete partial order (CCPO)**”) besitzt stets ein kleinstes Element, eindeutig bestimmt als Supremum der leeren Kette und üblicherweise mit  $\perp$  bezeichnet:  $\perp =_{df} \bigsqcup \emptyset$ .

# Ketten

Sei  $(P, \sqsubseteq)$  eine partielle Ordnung.

Eine Teilmenge  $K \subseteq P$  heißt

- ▶ **Kette** in  $P$ , wenn die Elemente in  $K$  total geordnet sind.

Für  $K = \{k_0 \sqsubseteq k_1 \sqsubseteq k_2 \sqsubseteq \dots\}$  ( $\{k_0 \supseteq k_1 \supseteq k_2 \supseteq \dots\}$ )

spricht man auch genauer von einer **aufsteigenden**  
(**absteigenden**) Kette in  $P$ .

Eine Kette  $K$  heißt

- ▶ **endlich**, wenn  $K$  endlich ist, sonst **unendlich**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

142/863

# Kettenendlichkeit, endliche Elemente

Eine **partielle Ordnung**  $(P, \sqsubseteq)$  heißt

- ▶ **kettenendlich** gdw.  $P$  enthält keine unendlichen Ketten.

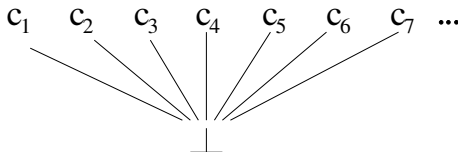
Ein **Element**  $p \in P$  heißt

- ▶ **endlich** gdw. die Menge  $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$  keine unendliche Kette enthält.
- ▶ **endlich relativ zu**  $r \in P$  gdw. die Menge  $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$  keine unendliche Kette enthält.

# Flache CPOs

Sei  $(C, \sqsubseteq)$  eine CPO. Dann heißt  $(C, \sqsubseteq)$

- ▶ **flach**, wenn für alle  $c, d \in C$  gilt:  
 $c \sqsubseteq d \Leftrightarrow c = \perp \vee c = d$





# Funktionsraum-CPO

Seien  $(C, \sqsubseteq_C)$  und  $(D, \sqsubseteq_D)$  zwei CPOs und  $[C \rightarrow D] =_{df} \{f : C \rightarrow D \mid f \text{ stetig}\}$  die Menge der stetigen Funktionen von  $C$  nach  $D$ .

Dann ist auch

- ▶ der **stetige Funktionenraum**  $([C \rightarrow D], \sqsubseteq)$  eine CPO mit
  - ▶  $\forall f, g \in [C \rightarrow D]. f \sqsubseteq g \iff \forall c \in C. f(c) \sqsubseteq_D g(c)$

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

145/863

# Monotone und stetige Funktionen auf CPOs

Seien  $(C, \sqsubseteq_C)$  und  $(D, \sqsubseteq_D)$  zwei CPOs und sei  $f : C \rightarrow D$  eine Funktion von  $C$  nach  $D$ .

Dann heißt  $f$

- ▶ **monoton** gdw.  $\forall c, c' \in C. c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$   
(Erhalt der Ordnung der Elemente)
- ▶ **stetig** gdw.  $\forall C' \subseteq C. f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$   
(Erhalt der kleinsten oberen Schranken)

# Eigenschaften von Funktionen auf CPOs

Mit den vorigen Bezeichnungen gilt:

## Lemma (3.3.1)

*f ist monoton gdw.  $\forall C' \subseteq C. f(\bigsqcup_C C') \sqsupseteq_D \bigsqcup_D f(C')$*

## Korollar (3.3.2)

*Eine stetige Funktion ist stets monoton,  
d.h.  $f$  stetig  $\Rightarrow f$  monoton.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

**3.3**

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

147/863

# Fixpunkte

Sei  $(C, \sqsubseteq)$  eine CPO,  $f : C \rightarrow C$  eine Funktion auf  $C$  und sei  $c$  ein Element von  $C$ , also  $c \in C$ .

Dann heißt  $c$

- ▶ **Fixpunkt** von  $f$  gdw.  $f(c) = c$

Ein **Fixpunkt**  $c$  von  $f$  heißt

- ▶ **kleinster Fixpunkt** von  $f$   
gdw.  $\forall d \in C. f(d) = d \Rightarrow c \sqsubseteq d$
- ▶ **größter Fixpunkt** von  $f$  gdw.  $\forall d \in C. f(d) = d \Rightarrow d \sqsubseteq c$

**Bezeichnungen:**

Der **kleinste** bzw. **größte Fixpunkt** einer Funktion  $f$  wird oft mit  $\mu f$  bzw.  $\nu f$  bezeichnet.

# Bedingte Fixpunkte, inflationäre Funktionen

Seien  $d, c_d \in C$ . Dann heißt  $c_d$

- ▶ **bedingter kleinster Fixpunkt** von  $f$  bezüglich  $d$  gdw.  $c_d$  ist der kleinste Fixpunkt von  $C$  mit  $d \sqsubseteq c_d$ , d.h. für alle anderen Fixpunkte  $x$  von  $f$  mit  $d \sqsubseteq x$  gilt:  $c_d \sqsubseteq x$ .

Sei  $(C, \sqsubseteq)$  eine CPO und sei  $f : C \rightarrow C$  eine Funktion auf  $C$ . Dann heißt  $f$

- ▶ **inflationär (vergrößernd)** gdw.  $\forall c \in C. c \sqsubseteq f(c)$

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

149/863

## Theorem (3.3.3, Knaster/Tarski, Kleene)

Sei  $(C, \sqsubseteq)$  eine CPO und sei  $f : C \rightarrow C$  eine stetige Funktion auf  $C$ .

Dann hat  $f$  einen kleinsten Fixpunkt  $\mu f$  und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette (sog. *Kleene-Kette*)  $\{\perp, f(\perp), f^2(\perp), \dots\}$ , d.h.

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) = \bigsqcup \{\perp, f(\perp), f^2(\perp), \dots\}$$

# Beweis des Fixpunkttheorems 3.3.3

Zu zeigen:  $\mu f$ :

1. existiert
2. ist Fixpunkt
3. ist kleinster Fixpunkt

Zum Beweis von 1., 2. und 3. siehe Anhang A5.

# Bedingte Fixpunkte

## Theorem (3.3.4, Bedingte Fixpunkte)

*Sei  $(C, \sqsubseteq)$  eine CPO, sei  $f : C \rightarrow C$  eine stetige, inflationäre Funktion auf  $C$  und sei  $d \in C$ .*

*Dann hat  $f$  einen kleinsten bedingten Fixpunkt  $\mu f_d$  bezüglich  $d$  und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette  $\{d, f(d), f^2(d), \dots\}$ , d.h.*

$$\mu f_d = \bigsqcup_{i \in \mathbb{N}_0} f^i(d) = \bigsqcup \{d, f(d), f^2(d), \dots\}$$



# Endliche Fixpunkte

## Theorem (3.3.5, Endliche Fixpunkte)

Sei  $(C, \sqsubseteq)$  eine CPO und sei  $f : C \rightarrow C$  eine stetige Funktion auf  $C$ .

Dann gilt: Sind in der Kleene-Kette von  $f$  zwei aufeinanderfolgende Glieder gleich, etwa  $f^i(\perp) = f^{i+1}(\perp)$ , so gilt  $\mu f = f^i(\perp)$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

**3.3**

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

153/863

# Existenz endlicher Fixpunkte

Hinreichende Bedingungen für die Existenz endlicher Fixpunkte sind:

- ▶ Endlichkeit von Definitions- und Wertebereich von  $f$
- ▶  $f$  ist von der Form  $f(c) = c \sqcup g(c)$  für monotonen  $g$  über kettenendlichem Wertebereich

Inhalt

Kap. 1

Kap. 2

Kap. 3

3.1

3.2

3.3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13





Kap. 14

Kap. 15

Kap. 16

154/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 3

-  Michael J.C. Gordon. *The Denotational Description of Programming Languages*. Springer-V., 1979.
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 4, Denotational Semantics)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 5, Denotational Semantics; Chapter 6, More on Denotational Semantics)
-  Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1981.

# Kapitel 4

## Axiomatische Semantik von *WHILE*

Inhalt

Kap. 1

Kap. 2

Kap. 3

**Kap. 4**

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Axiomatische Semantik von *WHILE*

*...bestimmte Eigenschaften des Effekts der Ausführung eines Konstrukts werden als **Zusicherungen** ausgedrückt. Bestimmte andere Aspekte der Ausführung werden dabei i.a. ignoriert.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

**Kap. 4**

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

157/863

# Axiomatische Semantik

Wir betrachten insbesondere: Korrektheit und Vollständigkeit der axiomatischen Semantik

Erinnerung:

- ▶ Hoare-Tripel (syntaktische Sicht) bzw. Korrektheitsformeln (semantische Sicht) der Form

$$\{p\} \pi \{q\} \quad \text{bzw.} \quad [p] \pi [q]$$

- ▶ Gültigkeit einer Korrektheitsformel im Sinne
  - ▶ partieller Korrektheit
  - ▶ totaler Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

158/863

# Zwei wegbereitende klassische Arbeiten

- ▶ Robert W. Floyd. [Assigning Meaning to Programs](#). Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, Vol. 19, 19-32, 1967.
- ▶ Charles A.R. Hoare. [An Axiomatic Basis for Computer Programming](#). Communications of the ACM 12:576-580, 583, 1969.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Kapitel 4.1

## Partielle und totale Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**4.1**

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

160/863



# Partielle Korrektheit

Sei  $\pi \in \mathbf{Prg}$  ein *WHILE*-Programm und seien  $p$  und  $q$  zwei logische Formeln:

## Definition (4.1.1, Partielle Korrektheit)

Eine Hoaresche Zusicherung  $\{p\} \pi \{q\}$  heißt

- ▶ gültig (im Sinne der partiellen Korrektheit) oder kurz (partiell) korrekt (in Zeichen:  $\models_{pk} \{p\} \pi \{q\}$ ) gdw für jeden Anfangszustand  $\sigma$  gilt:

Ist die Vorbedingung  $p$  in  $\sigma$  erfüllt und terminiert die zugehörige Berechnung von  $\pi$  angesetzt auf  $\sigma$  regulär in einem Endzustand  $\sigma'$ , dann ist auch die Nachbedingung  $q$  in  $\sigma'$  erfüllt.

# Totale Korrektheit

Sei  $\pi \in \mathbf{Prg}$  ein *WHILE*-Programm und seien  $p$  und  $q$  zwei logische Formeln:

## Definition (4.1.2, Totale Korrektheit)

Eine Hoaresche Zusicherung  $[p] \pi [q]$  heißt

- ▶ gültig (im Sinne der totalen Korrektheit) oder kurz (total) korrekt (in Zeichen:  $\models_{tk} [p] \pi [q]$ ) gdw für jeden Anfangszustand  $\sigma$  gilt:

Ist die Vorbedingung  $p$  in  $\sigma$  erfüllt, dann terminiert die zugehörige Berechnung von  $\pi$  angesetzt auf  $\sigma$  regulär mit einem Endzustand  $\sigma'$  und die Nachbedingung  $q$  ist in  $\sigma'$  erfüllt.

“Totale Korrektheit = Partielle Korrektheit + Terminierung”

# Eigenschaften von Korrektheitsformeln (1)

## Definition (4.1.3, Charakterisierung)

Die Zustandsmenge

$$Ch(p) =_{df} \{ \sigma \in \Sigma \mid \llbracket p \rrbracket_B(\sigma) = \mathbf{wahr} \}$$

heißt **Charakterisierung** von  $p \in \mathbf{Bexp}$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**4.1**

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

164/863

# Eigenschaften von Korrektheitsformeln (2)

## Lemma (4.1.4, Eigenschaften von Korrektheitsformeln)

Eine Korrektheitsformel  $\{p\} \pi \{q\}$  ist

- ▶ *partiell korrekt* (in Zeichen:  $\models_{pk} \{p\} \pi \{q\}$ ), falls  $\llbracket \pi \rrbracket(Ch(p)) \subseteq Ch(q)$
- ▶ *total korrekt* (in Zeichen:  $\models_{tk} [p] \pi [q]$ ), falls  $\{p\} \pi \{q\}$  partiell korrekt ist und  $Def(\llbracket \pi \rrbracket) \supseteq Ch(p)$  gilt.

Dabei bezeichnet  $Def(\llbracket \pi \rrbracket)$  die Menge aller Zustände, für die  $\pi$  regulär terminiert.

**Vereinbarung:**  $\llbracket \pi \rrbracket(Ch(p)) =_{df} \{\llbracket \pi \rrbracket(\sigma) \mid \sigma \in Ch(p)\}$

# Erinnerung

...an einige Sprechweisen:

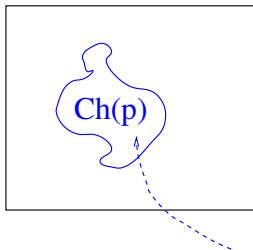
## Ein (deterministisches) Programm $\pi$

- ▶ angesetzt auf einen Anfangszustand  $\sigma$  **terminiert regulär** gdw.  $\pi$  nach endlich vielen Schritten in einem Zustand  $\sigma' \in \Sigma$  endet.
- ▶ angesetzt auf einen Anfangszustand  $\sigma$  **terminiert irregulär** gdw.  $\pi$  nach endlich vielen Schritten zu einer Konfiguration  $\langle \pi', \sigma' \rangle$  führt, für die es keine Folgekonfiguration gibt (z.B. wg. Division durch **0**).
- ▶ Ein Programm  $\pi$  heißt **divergent** gdw.  $\pi$  terminiert für keinen Anfangszustand regulär.

# Veranschaulichung (1)

...der Charakterisierung  $Ch(p)$  einer logischen Formel  $p$ :

Menge aller Zustände  $\Sigma$

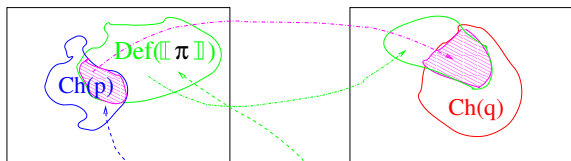


Charakterisierung von  $p$ :  $Ch(p) \subseteq \Sigma$

# Veranschaulichung (2)

...der Gültigkeit eine Hoareschen Zusicherung  $\{p\} \pi \{q\}$  im Sinne partieller Korrektheit:

Menge aller Zustände  $\Sigma$



Charakterisierung von  $p$ :  $Ch(p) \subseteq \Sigma$

Definitionsbereich von  $\pi$ :  $Def(\llbracket \pi \rrbracket) \subseteq \Sigma$



Bild von  $\llbracket \pi \rrbracket$  für  $Def(\llbracket \pi \rrbracket)$



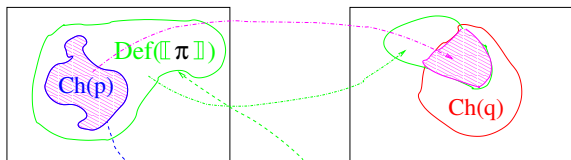
Bild von  $\llbracket \pi \rrbracket$  für  $Def(\llbracket \pi \rrbracket) \cap Ch(p)$



# Veranschaulichung (3)

...der Gültigkeit eine Hoareschen Zusicherung  $[p] \pi [q]$  im Sinne totaler Korrektheit:

Menge aller Zustände  $\Sigma$



Charakterisierung von  $p$ :  $Ch(p) \leq \Sigma$

Definitionsbereich von  $\pi$ :  $Def(\llbracket \pi \rrbracket) \leq \Sigma$



Bild von  $\llbracket \pi \rrbracket$  für  $Def(\llbracket \pi \rrbracket)$



Bild von  $\llbracket \pi \rrbracket$  für  $Def(\llbracket \pi \rrbracket) \wedge Ch(p)$

# Stärkste Nachbedingungen, schwächste Vorbedingungen

In der Folge:

## Präzisierung der Begriffe

- ▶ Stärkste Nachbedingung
- ▶ Schwächste (liberale) Vorbedingung

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**4.1**

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

170/863

# Stärkste Nach- und schwächste Vorbedingungen (1)

In der Situation der vorigen Abbildungen gilt:

- ▶  $\llbracket \pi \rrbracket(Ch(p))$  heißt **stärkste Nachbedingung** von  $\pi$  bezüglich  $p$ .
- ▶  $\llbracket \pi \rrbracket^{-1}(Ch(q))$  heißt **schwächste Vorbedingung** von  $\pi$  bezüglich  $q$ , wobei  $\llbracket \pi \rrbracket^{-1}(\Sigma') =_{df} \{\sigma \in \Sigma \mid \llbracket \pi \rrbracket(\sigma) \in \Sigma'\}$
- ▶  $\llbracket \pi \rrbracket^{-1}(Ch(q)) \cup C(Def(\llbracket \pi \rrbracket))$  heißt **schwächste liberale Vorbedingung** von  $\pi$  bezüglich  $q$ , wobei  $C$  den Mengenkomplementoperator (bzgl. der Grundmenge  $\Sigma$ ) bezeichnet.

# Stärkste Nach- und schwächste Vorbedingungen (2)

## Lemma (4.1.5)

Ist  $\llbracket \pi \rrbracket$  total definiert, d.h. gilt  $\text{Def}(\llbracket \pi \rrbracket) = \Sigma$ , dann gilt für alle Formeln  $p$  und  $q$ :

$$\llbracket \pi \rrbracket(\text{Ch}(p)) \subseteq \text{Ch}(q) \iff \llbracket \pi \rrbracket^{-1}(\text{Ch}(q)) \supseteq \text{Ch}(p)$$

**Beweis:** Übungsaufgabe

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

172/863

# Zusammenhang von partieller und totaler Korrektheit

## Lemma (4.1.6)

*Für deterministische Programme  $\pi$  gilt:*

$$\models_{tk} [p] \pi [q] \Rightarrow \models_{pk} \{p\} \pi \{q\}$$

*d.h. für deterministische Programme impliziert totale Korrektheit bzgl. eines Paares aus Vor- und Nachbedingung auch partielle Korrektheit bzgl. dieses Paares aus Vor- und Nachbedingung.*

# Schwächste Vor- & stärkste Nachbedingungen

...aus logischer Perspektive:

## Definition (4.1.7, schwächer als)

Seien  $A, B, A_1, A_2, \dots$  (logische) Formeln

- ▶  $A$  heißt **schwächer** als  $B$ , wenn gilt:  $B \Rightarrow A$
- ▶  $A_i$  heißt **schwächste** Formel in  $\{A_1, A_2, \dots\}$ , wenn gilt:  
 $A_j \Rightarrow A_i$  für alle  $j$ .

## Definition (4.1.8, stärker als)

Seien  $A, B, A_1, A_2, \dots$  (logische) Formeln

- ▶  $A$  heißt **stärker** als  $B$ , wenn gilt:  $B$  ist schwächer als  $A$ ,  
d.h. wenn gilt:  $A \Rightarrow B$
- ▶  $A_i$  heißt **stärkste** Formel in  $\{A_1, A_2, \dots\}$ , wenn gilt:  
 $A_i \Rightarrow A_j$  für alle  $j$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

174/863

# Schwächste Vorbedingungen

## Definition (4.1.9, Schwächste Vorbedingung)

Sei  $\pi$  ein Programm und  $q$  eine Formel. Dann heißt

- ▶  $wp(\pi, q)$  **schwächste Vorbedingung** für totale Korrektheit von  $\pi$  bezüglich (der Nachbedingung)  $q$ , wenn

$$[wp(\pi, q)] \pi [q]$$

total korrekt ist und  $wp(\pi, q)$  die schwächste Formel mit dieser Eigenschaft ist.

# Schwächste liberale Vorbedingungen

## Definition (4.1.10, Schwächste liberale Vorbeding'g)

Sei  $\pi$  ein Programm und  $q$  eine Formel. Dann heißt

- ▶  $wlp(\pi, q)$  **schwächste liberale Vorbedingung** für partielle Korrektheit von  $\pi$  bezüglich (der Nachbedingung)  $q$ , wenn

$$\{wlp(\pi, q)\} \pi \{q\}$$

partiell korrekt ist und  $wlp(\pi, q)$  die schwächste Formel mit dieser Eigenschaft ist.



# Zusammenfassung von Sprechweisen (1)

Hoaresche Zusicherungen sind von einer der zwei Formen

- ▶  $\{p\} \pi \{q\}$  und
- ▶  $[p] \pi [q]$

wobei

- ▶  $p, q$  logische Formeln sind (meist prädikatenlogische Formeln 1. Stufe)
- ▶  $\pi$  ein Programm ist
- ▶  $p$  und  $q$  Vor- bzw. Nachbedingung heißen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

177/863

# Zusammenfassung von Sprechweisen (2)

In einer **Hoareschen Zusicherung** werden üblicherweise

- ▶ **geschweifte Klammern** wie in

$$\{p\} \pi \{q\}$$

für Tripel im Sinne **partieller Korrektheit** und

- ▶ **eckige Klammern** wie in

$$[p] \pi [q]$$

für Tripel im Sinne **totaler Korrektheit**

benutzt.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

178/863

# Zwei Beispiele Hoarescher Zusicherungen

$$\{a > 0\}$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$\{y = a!\}$$

...zum Ausdruck partieller Korrektheit von  $\pi$  bzgl. der Vorbedingung  $a > 0$  und der Nachbedingung  $y = a!$

$$[a > 0]$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$[y = a!]$$

...zum Ausdruck totaler Korrektheit von  $\pi$  bzgl. der Vorbedingung  $a > 0$  und der Nachbedingung  $y = a!$

# Kapitel 4.2

## Beweiskalkül für partielle Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

**4.2**

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

180/863

# Hoare-Kalkül $HK_{PK}$ für partielle Korrektheit

$$[\text{skip}] \frac{\text{—}}{\{p\} \text{ skip } \{p\}}$$

$$[\text{ass}] \frac{\text{—}}{\{p[t \setminus x]\} x := t \{p\}}$$

$$[\text{comp}] \frac{\{p\} \pi_1 \{r\}, \{r\} \pi_2 \{q\}}{\{p\} \pi_1; \pi_2 \{q\}}$$

$$[\text{ite}] \frac{\{p \wedge b\} \pi_1 \{q\}, \{p \wedge \neg b\} \pi_2 \{q\}}{\{p\} \text{ if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } \{q\}}$$

$$[\text{while}] \frac{\{I \wedge b\} \pi \{I\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

$$[\text{cons}] \frac{p \Rightarrow p_1, \{p_1\} \pi \{q_1\}, q_1 \Rightarrow q}{\{p\} \pi \{q\}}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

181/863

# Diskussion von Vorwärtszuweisungsregel(n)

- ▶ Eine **Vorwärtsregel** für die Zuweisung wie

$$[\text{ass}_{fwd}] \quad \frac{\overline{\quad}}{\{p\} \ x:=t \ \{\exists z. \ p[z/x] \wedge \ x=t[z/x]\}}$$

mag natürlich erscheinen, ist aber beweistechnisch unangenehm durch das Mitschleppen quantifizierter Formeln.

- ▶ **Beachte:** Folgende scheinbar naheliegende quantorfreie Realisierung der Vorwärtszuweisungsregel ist nicht korrekt:

$$[\text{ass}_{naive}] \quad \frac{\overline{\quad}}{\{p\} \ x:=t \ \{p[t/x]\}}$$

**Beweis:** Übungsaufgabe

# Kapitel 4.3

## Beweiskalkül für totale Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Hoare-Kalküle: Partielle/totale Korrektheit (1)

Für die Sprache *WHILE* sind

- ▶ die Hoare-Kalküle für partielle und totale Korrektheit i.w. *identisch*.

Sie unterscheiden sich

- ▶ *einzig* in der Regel zur Behandlung der *while-Schleife*.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

184/863



# Hoare-Kalküle: Partielle/totale Korrektheit (2)

Im Kern muss die Regel [while] von  $HK_{PK}$  ersetzt werden durch eine

- ▶ terminationsordnungssensitive Regel.

Hierfür gibt es unterschiedliche Möglichkeiten, wobei eine Abwägung zu treffen ist zwischen

- ▶ Einfachheit der Regel
- ▶ Einfachheit der Regelanwendung.

In der Folge werden wir

- ▶ zwei Varianten mit unterschiedlicher Abwägung vorstellen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

185/863

# Hoare-Kalkül $HK_{TK}$ für totale Korrektheit

## Variante 1: Regeleinfachheit vor Regelanwendungseinfachheit

...ersetze die Regel [while] aus  $HK_{PK}$  durch:

$$[\text{while}'_{TK}] \quad \frac{I \Rightarrow t \geq 0, \{I \wedge b \wedge t = w\} \pi \{I \wedge t < w\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

wobei

- ▶  $t$  arithmetischer Term über ganzen Zahlen,
- ▶  $w$  ganzzahlige Variable, die in  $I$ ,  $b$ ,  $\pi$  und  $t$  nicht frei vorkommt.

$\rightsquigarrow (\mathbb{N}, <)$  als **Terminationsordnung!**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

186/863

# Hoare-Kalkül $HK_{TK}$ für totale Korrektheit

## Variante 2: Regelanwendungseinfachheit vor Regeleinfachheit

$$[\text{while}_{TK}] \quad \frac{I \wedge b \Rightarrow u[t/v], \{I \wedge b \wedge t=w\} \pi \{I \wedge t < w\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

wobei

- ▶  $u$  Boolescher Ausdruck über der Variablen  $v$ ,
- ▶  $t$  Term (sog. Terminierungsterm),
- ▶  $w$  Variable, die in  $I$ ,  $b$ ,  $\pi$  und  $t$  nicht frei vorkommt,
- ▶  $M =_{df} \{\sigma(v) \mid \sigma \in Ch(u)\}$  bzgl.  $\sqsubset$  Noethersch geordnete Menge (sog. noethersche Halbordnung).

$\rightsquigarrow (M, \sqsubset)$  als **Terminationsordnung!**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

187/863

# Vergleich von $[\text{while}_{TK}]$ und $[\text{while}'_{TK}]$

## Beachte:

- ▶  $[\text{while}_{TK}]$ : beliebige Noethersche Ordnung als Terminationsordnung zulässig:  $(M, \sqsubset)$
- ▶  $[\text{while}'_{TK}]$ : statt beliebiger Noetherscher Terminationsordnung Festlegung auf eine spezielle Noethersche Ordnung als Terminationsordnung, nämlich:  $(\mathbb{N}, <)$ .

Oft erfordert die Rückspiegelung einer sich 'natürlich' anbietenden Noetherschen Terminationsordnung auf die spezielle Noethersche Ordnung  $(\mathbb{N}, <)$  zusätzlichen Modellierungsaufwand.

In diesen Fällen hat  $[\text{while}_{TK}]$  pragmatische Vorteile gegenüber Regel  $[\text{while}'_{TK}]$ .

# Zur Vollständigkeit

...seien die übrigen Regeln des Hoare-Kalkül  $HK_{TK}$  für totale Korrektheit hier ebenfalls angegeben:

$$[\text{skip}] \frac{\text{---}}{\{p\} \text{ skip } \{p\}}$$

$$[\text{ass}] \frac{\text{---}}{\{p[t \setminus x]\} x := t \{p\}}$$

$$[\text{comp}] \frac{\{p\} \pi_1 \{r\}, \{r\} \pi_2 \{q\}}{\{p\} \pi_1; \pi_2 \{q\}}$$

$$[\text{ite}] \frac{\{p \wedge b\} \pi_1 \{q\}, \{p \wedge \neg b\} \pi_2 \{q\}}{\{p\} \text{ if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } \{q\}}$$

$$[\text{cons}] \frac{p \Rightarrow p_1, \{p_1\} \pi \{q_1\}, q_1 \Rightarrow q}{\{p\} \pi \{q\}}$$

# Bemerkung

In den vorigen Regeln verwenden wir (auch) für totale Korrektheit geschweifte statt eckiger Klammern für zugesicherte Eigenschaften, um einen Bezeichnungskonflikt mit der ebenfalls durch eckige Klammern bezeichneten **syntaktischen Substitution** zu vermeiden.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Wohlfundierte Ordnungen (1)

## Definition (4.3.1, Irreflexive partielle Ordnung)

Sei  $P$  eine Menge und sei  $<$  eine irreflexive und transitive Relation auf  $P$ .

Dann ist das Paar  $(P, <)$  eine **irreflexive partielle Ordnung**.

**Beispiele:**  $(\mathbb{Z}, <)$ ,  $(\mathbb{Z}, >)$ ,  $(\mathbb{N}, <)$ ,  $(\mathbb{N}, >)$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

191/863

# Wohlfundierte Ordnungen (2)

## Definition (4.3.2, Wohlfundierte Ordnung)

Sei  $(P, <)$  eine irreflexive partielle Ordnung und sei  $W$  eine Teilmenge von  $P$ .

Dann heißt die Relation  $<$  auf  $W$  **wohlfundiert**, wenn es keine unendlich absteigende Kette

$$\dots < w_2 < w_1 < w_0$$

von Elementen  $w_i \in W$  gibt. Das Paar  $(W, <)$  heißt dann eine **wohlfundierte Struktur** oder auch eine **wohlfundierte** oder **Noethersche Ordnung**.

**Sprechweise:** Gilt  $w < w'$  für  $w, w' \in W$ , sagen wir,  $w$  ist **kleiner als**  $w'$  oder  $w'$  ist **größer als**  $w$ .

**Beispiele:**  $(\mathbb{N}, <)$ , aber nicht  $(\mathbb{Z}, <)$ ,  $(\mathbb{Z}, >)$  oder  $(\mathbb{N}, >)$



# Prinzipien zur Konstruktion wohlfundierter Ordnungen

...aus gegebenen wohlfundierten Ordnungen:

## Lemma (4.3.3)

Seien  $(W_1, <_1)$  und  $(W_2, <_2)$  zwei wohlfundierte Ordnungen.  
Dann sind auch

- ▶  $(W_1 \times W_2, <_{com})$  mit *komponentenweiser* Ordnung definiert durch

$$(m_1, m_2) <_{com} (n_1, n_2) \text{ gdw. } m_1 <_1 n_1 \wedge m_2 <_2 n_2$$

- ▶  $(W_1 \times W_2, <_{lex})$  mit *lexikographischer* Ordnung def. durch

$$(m_1, m_2) <_{lex} (n_1, n_2) \text{ gdw.}$$

$$(m_1 <_1 n_1) \vee (m_1 = n_1 \wedge m_2 <_2 n_2)$$

wohlfundierte Ordnungen.

# Anmerkungen zu

...den der

- ▶ Konsequenzregel [cons] und der
- ▶ Schleifenregeln [while<sub>PK</sub>] und [while<sub>TK</sub>]

von  $HK_{PK}$  bzw.  $HK_{TK}$  zugrundeliegenden Intuitionen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Zur Konsequenzregel (1)

$$[\text{cons}] \quad \frac{p \Rightarrow p_1, \{p_1\} \pi \{q_1\}, q_1 \Rightarrow q}{\{p\} \pi \{q\}}$$

Intuitiv:

Die **Konsequenzregel**

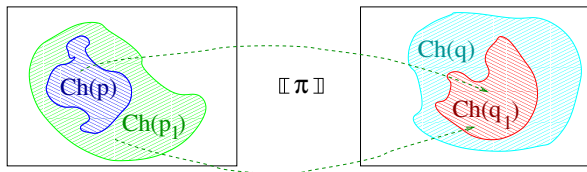
- ▶ stellt die Schnittstelle zwischen Programmverifikation und den logischen Formeln der Zusageungssprache dar
- ▶ erlaubt es,
  - ▶ Vorbedingungen zu **verstärken**  
(Übergang von  $p_1$  zu  $p$  möglich, falls  $p \Rightarrow p_1$  ( $\Leftrightarrow Ch(p) \subseteq Ch(p_1)$ ))
  - ▶ Nachbedingungen **abzuschwächen**  
(Übergang von  $q_1$  zu  $q$  möglich, falls  $q_1 \Rightarrow q$  ( $\Leftrightarrow Ch(q_1) \subseteq Ch(q)$ ))

um so die Anwendung anderer Beweisregeln zu ermöglichen.

# Zur Konsequenzregel (2)

Veranschaulichung von Verstärkung und Abschwächung:

Menge aller Zustände  $\Sigma$



$$p \implies p_1 \quad \{p_1\} \pi \{q_1\} \quad q_1 \implies q$$

$$\text{z.B.: } x > 5 \implies x > 0 \quad \{x > 0\} \pi \{y > 5\} \quad y > 5 \implies y > 0$$

# Zur while-Regel in $HK_{PK}$

$$[\text{while}] \quad \frac{\{I \wedge b\} \pi \{I\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

## Intuitiv:

- ▶ Das durch  $I$  beschriebene Prädikat gilt
  - ▶ vor und nach jeder Ausführung des Rumpfes der while-Schleife
  - ▶ und wird deswegen als **Invariante** der while-Schleife bezeichnet.
- ▶ Die while-Regel besagt weiter, dass
  - ▶ wenn zusätzlich (zur Invarianten) auch  $b$  vor jeder Ausführung des Schleifenrumpfs gilt, dass nach Beendigung der while-Schleife  $\neg b$  wahr ist.

# Zur while-Regel in $HK_{TK}$ (1)

Erinnerung:

$$[\text{while}_{TK}] \frac{I \wedge b \Rightarrow u[t/v], \{I \wedge b \wedge t=w\} \pi \{I \wedge t < w\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

wobei

- ▶  $u$  Boolescher Ausdruck über der Variablen  $v$ ,
- ▶  $t$  arithmetischer Term,
- ▶  $w$  Variable, die in  $I$ ,  $b$ ,  $\pi$  und  $t$  nicht frei vorkommt,
- ▶  $M =_{df} \{\sigma(v) \mid Ch(u)\}$  bzgl.  $\sqsubset$  Noethersch geordnete Menge (sog. Noethersche Halbordnung).

$\rightsquigarrow (M, \sqsubset)$  Terminationsordnung!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

198/863

## Zur while-Regel in $HK_{TK}$ (2)

- ▶ **Prämisse 1:**  $I \wedge b \Rightarrow u[t/v]$

Wann immer der Schleifenrumpf noch einmal ausgeführt wird (d.h.  $I \wedge b$  ist wahr), gilt, dass  $u[t/v]$  wahr ist, woraus aufgrund der Definition von  $M$  folgt, dass der Wert von  $t$  Element einer Noethersch geordneten Menge ist.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

199/863

# Zur while-Regel in $HK_{TK}$ (3)

- ▶ **Prämisse 2:**  $\{I \wedge b \wedge t = w\} \pi \{I \wedge t < w\}$ 
  - ▶  $w$  speichert den initialen Wert von  $t$  ( $w$  ist sog. *logische Variable*), d.h. den Wert, den  $t$  vor Eintritt in die Schleife hat (gilt, da  $w$  als logische Variable insbesondere nicht in  $\pi$  vorkommt)
  - ▶ Zusammen damit, dass der Wert von  $w$  (als logische Variable) invariant unter der Ausführung des Schleifenrumpfs ist, garantiert  $t < w$  in der Nachbedingung von Prämisse 2, dass der Wert von  $t$  nach jeder Ausführung des Schleifenrumpfs bzgl. der noetherschen Ordnung abgenommen hat.



## Zur while-Regel in $HK_{TK}$ (4)

- ▶ Zusammen implizieren die obigen beiden Punkte die Terminierung der while-Schleife, da es in einer Noetherschen geordneten Menge keine unendlich absteigenden Ketten gibt. Folglich kann die Bedingung  $l \wedge b$  in Prämisse 1 nicht unendlich oft wahr sein, da dies zusammen mit Prämisse 2 ein unendliches Absteigen erforderte.)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

**4.3**

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

201/863

# Programm- vs. logische Variablen (1)

Wir unterscheiden in Zusicherungen  $\{p\} \pi \{q\}$  zwischen:

- ▶ **Programmvariablen**

...Variablen, die in  $\pi$  vorkommen

- ▶ **logischen Variablen**

...Variablen, die in  $\pi$  nicht vorkommen

Logische Variablen erlauben

- ▶ sich **initiale** Werte von Programmvariablen zu “merken”, um in Nachbedingungen geeignet darauf Bezug zu nehmen.

# Programm- vs. logische Variablen (2)

## Beispiel:

- ▶  $\{x = n\} y := 1; \text{ while } x \neq 1 \text{ do } y := y * x; x := x - 1 \text{ od } \{y = n! \wedge n > 0\}$   
...die Nachbedingung macht eine Aussage über den Zusammenhang des Anfangswertes von  $x$  (gespeichert in  $n$ ) und des schließlichen Wertes von  $y$ .
- ▶  $\{x = n\} y := 1; \text{ while } x \neq 1 \text{ do } y := y * x; x := x - 1 \text{ od } \{y = x! \wedge x > 0\}$   
...die Nachbedingung macht eine Aussage über den Zusammenhang der schließlichen Werte von  $x$  und  $y$ .  
(**Beachte:** nur mit Programmvariablen keine Aussage über die Fakultätsberechnung in diesem Bsp.!) )

# $HK_{TK}$ versus $HK_{PK}$

Beachte:

$HK_{TK}$  und  $HK_{PK}$  sind bis auf die (Prämisse der) Schleifenregel identisch:

- ▶ **Totale Korrektheit:**  $[\text{while}_{TK}]$

$$[\text{while}_{TK}] \frac{I \wedge b \Rightarrow u[t/v], \{I \wedge b \wedge t=w\} \pi \{I \wedge t < w\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

- ▶ **Partielle Korrektheit:**  $[\text{while}_{PK}]$

$$[\text{while}_{PK}] \frac{\{I \wedge b\} \pi \{I\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

# Abschließende Beobachtung

## Beweistechnische Anmerkung:

“Zerlegt” man  $[\text{while}'_{TK}]$  wie folgt:

$$[\text{while}''_{TK}] \quad \frac{I \Rightarrow t \geq 0, \{I \wedge b\} \pi \{I\}, \{I \wedge b \wedge t = w\} \pi \{t < w\}}{\{I\} \text{ while } b \text{ do } \pi \text{ od } \{I \wedge \neg b\}}$$

wird deutlich, dass der Nachweis totaler Korrektheit einer Hoareschen Zusicherung besteht aus

- ▶ dem Nachweis ihrer partiellen Korrektheit
- ▶ dem Nachweis der Termination

Diese Trennung kann im Beweis explizit vollzogen werden (“**Totale Korrektheit = Partielle Korrektheit + Terminierung**”). Der Gesamtbeweis wird dadurch modular. Der Terminationsnachweis selbst ist oft einfach.

**Bemerkung:** Die obige Trennung kann für  $[\text{while}_{TK}]$  analog vorgenommen werden.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

205/863

# Kapitel 4.4

## Korrektheit und Vollständigkeit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

**4.4**

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Korrektheits- und Vollständigkeitsfrage

Sei  $K$  ein Kalkül für partielle bzw. totale Korrektheit

Zentral sind dann:

- ▶ **Korrektheitsfrage:** Ist jede mithilfe von  $K$  ableitbare (in Zeichen:  $\vdash$ ) Korrektheitsformel partiell bzw. total korrekt?
- ▶ **Vollständigkeitsfrage:** Ist jede partiell bzw. total korrekte Korrektheitsformel (in Zeichen:  $\models$ ) mithilfe von  $K$  ableitbar?

Speziell:

- ▶ Sind  $HK_{PK}$  und  $HK_{TK}$  **korrekt** und **vollständig**?

# Korrektheits- u. Vollständigkeitsbedeutung

## Definition (4.4.1, Korrektheit und Vollständigkeit)

Sei  $K$  ein Hoarescher Beweiskalkül (z.B.  $HK_{PK}$  und  $HK_{TK}$ ).

Dann heißt  $K$

- ▶ **korrekt** (engl. **sound**), falls gilt: Ist eine Korrektheitsformel mit  $K$  herleitbar/beweisbar (in Zeichen:  $\vdash \{p\} \pi \{q\}$ ), dann ist sie auch **semantisch gültig**:

$$\vdash \{p\} \pi \{q\} \Rightarrow \models \{p\} \pi \{q\}$$

- ▶ **vollständig** (engl. **complete**), falls gilt: Ist eine Korrektheitsformel semantisch gültig, dann ist sie auch mit  $K$  herleitbar/beweisbar:

$$\models \{p\} \pi \{q\} \Rightarrow \vdash \{p\} \pi \{q\}$$



# Bemerkung

Die Sprechweise

- ▶ Hoaresches Tripel oder kurz Hoare-Tripel bzw.
- ▶ Hoaresche Zusicherung oder kurz Korrektheitsformel

betont jeweils die

- ▶ syntaktische Sicht ( $\vdash_{pk} \{p\} \pi \{q\}$ ) bzw.
- ▶ semantische Sicht ( $\models_{pk} \{p\} \pi \{q\}$ )

auf

- ▶  $\{p\} \pi \{q\}$  (analog für  $[p] \pi [q]$ )

# Korrektheit von $HK_{PK}$ und $HK_{TK}$

## Theorem (4.4.2, Korrektheit von $HK_{PK}$ , $HK_{TK}$ )

1.  $HK_{PK}$  ist *korrekt*, d.h. jede mit  $HK_{PK}$  ableitbare Korrektheitsformel (in Zeichen:  $\vdash_{pk} \{p\} \pi \{q\}$ ) ist gültig im Sinne partieller Korrektheit:

$$\vdash_{pk} \{p\} \pi \{q\} \Rightarrow \models_{pk} \{p\} \pi \{q\}$$

2.  $HK_{TK}$  ist *korrekt*, d.h. jede mit  $HK_{TK}$  ableitbare Korrektheitsformel (in Zeichen:  $\vdash_{tk} [p] \pi [q]$ ) ist gültig im Sinne totaler Korrektheit:

$$\vdash_{tk} [p] \pi [q] \Rightarrow \models_{tk} [p] \pi [q]$$

**Beweis** durch Induktion über die Anzahl der Regelanwendungen im Beweisbaum zur Ableitung der Korrektheitsformel.

# Zur Vollständigkeit Hoarescher Beweiskalküle

Generell müssen wir unterscheiden zwischen Vollständigkeit

- ▶ **extensionaler** und
- ▶ **intensionaler**

Ansätze.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

**4.4**

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Extensionale vs. intensionale Ansätze

- ▶ **Extensional**

↔ Vor- und Nachbedingungen sind durch **Prädikate** beschrieben.

- ▶ **Intensional**

↔ Vor- und Nachbedingungen sind durch **Formeln einer Zusicherungssprache** beschrieben.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

**4.4**

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

212/863

# Vollständigkeit von $HK_{PK}$ und $HK_{TK}$

Für den extensionalen Ansatz gilt:

## Theorem (4.4.3, Vollständigkeit von $HK_{PK}$ & $HK_{TK}$ )

1.  $HK_{PK}$  ist *vollständig*, d.h. jede im Sinne partieller Korrektheit gültige Korrektheitsformel ist mit  $HK_{PK}$  ableitbar:

$$\models_{pk} \{p\} \pi \{q\} \Rightarrow \vdash_{pk} \{p\} \pi \{q\}$$

2.  $HK_{TK}$  ist *vollständig*, d.h. jede im Sinne totaler Korrektheit gültige Korrektheitsformel ist mit  $HK_{TK}$  ableitbar:

$$\models_{tk} [p] \pi [q] \Rightarrow \vdash_{tk} [p] \pi [q]$$

**Beweis** durch strukturelle Induktion über den Aufbau von  $\pi$ .

# Zur Vollständigkeit von $HK_{PK}$ und $HK_{TK}$

Für intensionale Ansätze (durch unterschiedliche Wahlen der Zusicherungssprache) gilt Vollständigkeit i.a. nur relativ zur **Entscheidbarkeit** und **Ausdruckskraft** der Zusicherungssprache.

## Informell

### ► **Entscheidbarkeit**

...ist die Gültigkeit von Formeln der Zusicherungssprache algorithmisch verifizierbar bzw. falsifizierbar?

### ► **Ausdruckskraft**

...lassen sich alle Prädikate, insbesondere schwächste und schwächste liberale Vorbedingungen und Terminationsfunktionen, durch Formeln der Zusicherungssprache beschreiben?

↪ *tieferliegende Frage*: ...lassen sich schwächste Vorbedingungen, etc. syntaktisch ausdrücken?

**Stichwort:** Relative Vollständigkeit im Sinne von Cook.

# Kapitel 4.5

## Beispiele zum Beweis partieller Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Die beiden Beispiele im Überblick (1)

...Beweis der partiellen Korrektheit Hoarescher Zusicherungen anhand zweier Programme zur Berechnung

- ▶ der Fakultät und
- ▶ der ganzzahligen Division mit Rest

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

216/863



# Die beiden Beispiele im Überblick (2)

Im Detail:

Beweise, dass die beiden Hoareschen Tripel

$$\{a > 0\}$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$\{y = a!\}$$

und

$$\{x \geq 0 \wedge y > 0\}$$

$q := 0; r := x; \text{ while } r \geq y \text{ do } q := q + 1; r := r - y \text{ od}$

$$\{x = q * y + r \wedge 0 \leq r < y\}$$

gültig sind im Sinne partieller Korrektheit.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

217/863

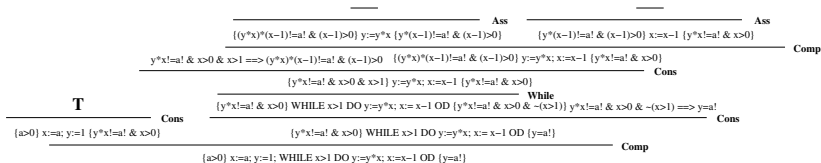
# Die beiden Beispiele im Überblick (3)

In der Folge geben wir die Beweise dafür in zwei Notationsvarianten an:

- ▶ **baumartig** (kanonische Variante)
- ▶ **lineare Beweisskizze** (pragmatische Variante)

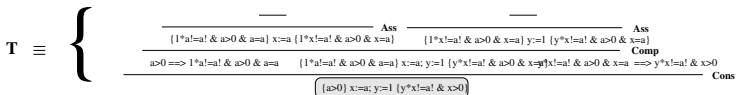
# Fakultätsbeispiel: Baumartiger Beweis (1)

## Erster Beweis



- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- 4.1
- 4.2
- 4.3
- 4.4
- 4.5
- 4.6
- 4.7
- 4.8
- Kap. 5
- Kap. 6
- Kap. 7
- Kap. 8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13

wobei



& : Logisches und  
- : Logisches nicht

# Fakultätsbeispiel: Baumartiger Beweis (2)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

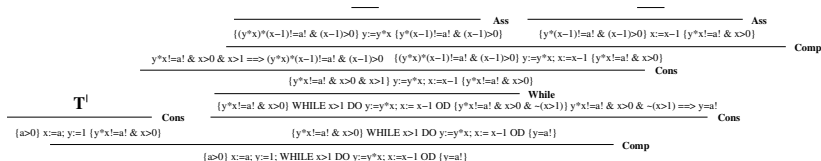
Kap. 10

Kap. 11

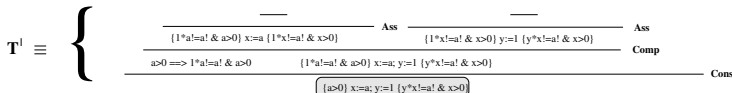
Kap. 12

Kap. 13

## Zweiter Beweis



wobei



& : Logisches und

- : Logisches nicht

# Divisionsbeispiel: Baumartiger Beweis

	$\frac{\text{Ass} \quad \text{Ass}}{\text{Comp} \quad \frac{\{x=(q+1)^*y+r-y \ \& \ r-y>=0\} \ q:=q+1 \ \{x=q^*y+r-y \ \& \ r-y>=0\} \quad \{x=q^*y+r-y \ \& \ r-y>=0\} \ r:=r-y \ \{x=q^*y+r \ \& \ r>=0\}}{\text{Cons} \quad \frac{(x=q^*y+r \ \& \ r>=0 \ \& \ r>=y) \Rightarrow (x=(q+1)^*y+r-y \ \& \ r-y>=0) \quad \{x=(q+1)^*y+r-y \ \& \ r-y>=0\} \ q:=q+1; \ r:=r-y \ \{x=q^*y+r \ \& \ r>=0\}}{\text{While} \quad \frac{\{x=q^*y+r \ \& \ r>=0 \ \& \ r>=y\} \ q:=q+1; \ r:=r-y \ \{x=q^*y+r \ \& \ r>=0\}}{\text{Cons} \quad \frac{\{x=q^*y+r \ \& \ r>=0\} \ \text{WHILE } r>=y \ \text{DO } q:=q+1; \ r:=r-y \ \text{OD } \{x=q^*y+r \ \& \ r>=0 \ \& \ \sim(r>=y)\} \quad (x=q^*y+r \ \& \ r>=0 \ \& \ \sim(r>=y)) \Rightarrow (x=q^*y+r \ \& \ r>=0 \ \& \ r<y)}$
--	--

	$\frac{\text{Ass} \quad \text{Ass}}{\text{Comp} \quad \frac{\{x=0^*y+x \ \& \ x>=0\} \ q:=0 \ \{x=q^*y+x \ \& \ x>=0\} \quad \{x=q^*y+x \ \& \ x>=0\} \ r:=x \ \{x=q^*y+r \ \& \ r>=0\}}{\text{Cons} \quad \frac{(x>=0 \ \& \ y>0) \Rightarrow (x=0^*y+x \ \& \ x>=0) \ \{x=0^*y+x \ \& \ x>=0\} \ q:=0; \ r:=x \ \{x=q^*y+r \ \& \ r>=0\}}{\text{Comp} \quad \frac{\{x>=0 \ \& \ y>0\} \ q:=0; \ r:=x \ \{x=q^*y+r \ \& \ r>=0\} \quad \{x=q^*y+r \ \& \ r>=0\} \ \text{WHILE } r>=y \ \text{DO } q:=q+1; \ r:=r-y \ \text{OD } \{x=q^*y+r \ \& \ r>=0 \ \& \ r<y\}}{\text{Comp} \quad \frac{\{x>=0 \ \& \ y>0\} \ q:=0; \ r:=x; \ \text{WHILE } r>=y \ \text{DO } q:=q+1; \ r:=r-y \ \text{OD } \{x=q^*y+r \ \& \ r>=0 \ \& \ r<y\}}$
--	---



& : Logisches und  
 ~ : Logisches nicht

- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- 4.1
- 4.2
- 4.3
- 4.4
- 4.5**
- 4.6
- 4.7
- 4.8
- Kap. 5
- Kap. 6
- Kap. 7
- Kap. 8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13

# Lineare Beweisskizzen

- ▶ Die unmittelbare baumartige Notation von Hoareschen Korrektheitsbeweisen ist i.a. unhandlich.
- ▶ Alternativ hat sich deshalb eine Notationsvariante eingebürgert, bei der in den Programmtext Zusicherungen als Annotationen eingestreut werden.
- ▶ In der Folge demonstrieren wir diesen Notationsstil am Beispiel des Nachweises der partiellen Korrektheit unseres Fakultätsprogramms bezüglich der angegebenen Vor- und Nachbedingung. Man spricht auch von einem sog. **linearen Beweis** bzw. einer **linearen Beweisskizze**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

| 222 / 863

# Fakultätsbeispiel: Lineare Beweisskizze (1)

Beweise, dass das Hoaresche Tripel

$$\{a > 0\}$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$\{y = a!\}$$

gültig ist im Sinne partieller Korrektheit.

Wir entwickeln den Beweis in der Folge Schritt für Schritt!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

223/863

# Fakultätsbeispiel: Lineare Beweisskizze (2)

## Schritt 1

“*Träumen*” der Invariante:

$$\blacktriangleright \{y * x! = a! \wedge x > 0\}$$

...um die [while]-Regel anwenden zu können.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13



# Fakultätsbeispiel: Lineare Beweisskizze (3)

## Schritt 2 Behandlung des Rumpfs der while-Schleife

Der Nachweis der Gültigkeit von

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

erlaube mithilfe der [while]-Regel den Übergang zu:

$$\{y * x! = a! \wedge x > 0\}$$

while  $x > 1$  do

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

od [while]

$$\{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

225/863

# Fakultätsbeispiel: Lineare Beweisskizze (4)

Behandlung des Rumpfs der while-Schleife im Detail:

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$
$$x := x - 1;$$

$$\{y * x! = a! \wedge x > 0\}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (5)

Wegen Rückwärtszuweisungsregel wird der Rumpf der while-Schleife von hinten nach vorne bearbeitet:

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$y := y * x;$$

$$\{y * (x - 1)! = a! \wedge x - 1 > 0\}$$

$$x := x - 1; \text{ [ass]}$$

$$\{y * x! = a! \wedge x > 0\}$$

# Fakultätsbeispiel: Lineare Beweisskizze (6)

Nach abermaliger Anwendung der [ass]-Regel erhalten wir

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$$\{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\}$$

$$y := y * x; \text{ [ass]}$$

$$\{y * (x - 1)! = a! \wedge x - 1 > 0\}$$

$$x := x - 1; \text{ [ass]}$$

$$\{y * x! = a! \wedge x > 0\}$$

...wobei noch eine "Beweislücke" verbleibt!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

228/863

# Fakultätsbeispiel: Lineare Beweisskizze (7)

Schluss der “Beweislücke” in der zugrundeliegenden Theorie:

$$\{y * x! = a! \wedge x > 0 \wedge x > 1\}$$

$\Downarrow$  [cons]

$$\{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\}$$

$y := y * x; [ass]$

$$\{y * (x - 1)! = a! \wedge x - 1 > 0\}$$

$x := x - 1; [ass]$

$$\{y * x! = a! \wedge x > 0\}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (8)

Anwendung der [while]-Regel liefert nun wie gewünscht:

$$\begin{aligned} & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (9)

Schritt 3 Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

$$\begin{aligned} & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \\ & \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

231/863

# Fakultätsbeispiel: Lineare Beweisskizze (10)

Schluss der Beweislücke in der zugrundeliegenden Theorie:

$$\begin{aligned} & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y * x! = a! \wedge x > 0 \wedge x \leq 1\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y * x! = a! \wedge x = 1\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

232/863



# Fakultätsbeispiel: Lineare Beweisskizze (11)

Aus Platzgründen etwas verkürzt dargestellt:

$$\begin{aligned} & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \Downarrow [\text{cons}] \\ & \quad \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (12)

**Schritt 4** Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$\begin{aligned} & \{a > 0\} \\ & \quad x := a; \\ & \quad y := 1; \\ & \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \text{od } [\text{while}] \\ & \quad \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \Downarrow [\text{cons}] \\ & \quad \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (13)

Einmalige Anwendung der [ass]-Regel liefert:

$$\begin{aligned} & \{a > 0\} \\ & \quad x := a; \\ & \quad \{1 * x! = a! \wedge x > 0\} \\ & \quad \quad y := 1; \text{ [ass]} \\ & \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \text{while } x > 1 \text{ do} \\ & \quad \quad \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \quad \Downarrow \text{ [cons]} \\ & \quad \quad \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad \quad \quad y := y * x; \text{ [ass]} \\ & \quad \quad \quad \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad \quad \quad \quad x := x - 1; \text{ [ass]} \\ & \quad \quad \quad \quad \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \quad \text{od [while]} \\ & \quad \quad \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \quad \Downarrow \text{ [cons]} \\ & \quad \quad \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (14)

Abermalige Anwendung der [ass]-Regel liefert:

$$\begin{aligned} & \{a > 0\} \\ & \{1 * a! = a! \wedge a > 0\} \\ & \quad x := a; [\text{ass}] \\ & \{1 * x! = a! \wedge x > 0\} \\ & \quad y := 1; [\text{ass}] \\ & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \{y * x! = a! \wedge x > 0\} \\ & \quad \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (15)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$\begin{aligned} & \{a > 0\} \\ & \Downarrow [\text{cons}] \\ & \{1 * a! = a! \wedge a > 0\} \\ & \quad x := a; [\text{ass}] \\ & \{1 * x! = a! \wedge x > 0\} \\ & \quad y := 1; [\text{ass}] \\ & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \Downarrow [\text{cons}] \\ & \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad y := y * x; [\text{ass}] \\ & \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad x := x - 1; [\text{ass}] \\ & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Insgesamt (16)

$$\begin{aligned} & \{a > 0\} \\ & \Downarrow [\text{cons}] \\ & \{1 * a! = a! \wedge a > 0\} \\ & \quad x := a; [\text{ass}] \\ & \{1 * x! = a! \wedge x > 0\} \\ & \quad y := 1; [\text{ass}] \\ & \{y * x! = a! \wedge x > 0\} \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \{y * x! = a! \wedge x > 0 \wedge x > 1\} \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \{(y * x) * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \{y * (x - 1)! = a! \wedge x - 1 > 0\} \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \{y * x! = a! \wedge x > 0\} \\ & \quad \text{od } [\text{while}] \\ & \{y * x! = a! \wedge x > 0 \wedge \neg(x > 1)\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y * x! = a! \wedge x > 0 \wedge x \leq 1\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y * x! = a! \wedge x = 1\} \\ & \quad \Downarrow [\text{cons}] \\ & \{y = a!\} \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (17)

Damit haben wir insgesamt wie gewünscht gezeigt:

Das Hoaresche Tripel

$$\{a > 0\}$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$\{y = a!\}$$

ist gültig im Sinne partieller Korrektheit.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Vergleich linearen und baumartigen Beweisstils

Vorteil **linearen** gegenüber **baumartigen Beweisnotationstils**:

- ▶ wenig Redundanz
- ▶ deshalb insgesamt knappere Beweise

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

**4.5**

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13



# Kapitel 4.6

## Beispiele zum Beweis totaler Korrektheit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Das Beispiel im Überblick

Beweise, dass das Hoaresche Tripel

$$[a > 0]$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$[y = a!]$$

gültig ist im Sinne totaler Korrektheit.

Wir entwickeln den Beweis in der Folge Schritt für Schritt!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

242/863

# Wahl von Invariante und Terminierungsterm

## Schritt 1

### “Träumen”

- ▶ der Invariante:  $y * x! = a! \wedge x > 0$
- ▶ des Terminierungsterms:  $t \equiv x$
- ▶ von  $u$ :  $u \equiv v \geq 0$

...um die [while]-Regel anwenden zu können.

### Beachte:

- ▶ Aus der Wahl von  $u \equiv v \geq 0$  und von  $b \equiv x > 1$  folgt:
  - ▶  $M = \{0, 1, 2, 3, 4, \dots\}$
  - ▶  $(v \geq 0)[x/v] \equiv x \geq 0$

...und somit insgesamt:  $I \wedge b \succ \sigma(x) \in M$  mit  $(M, <)$   
Noethersch geordnet.

**Hinweis zur Notation:**  $\equiv$  steht für *syntaktisch gleich*

# Wahl von Invariante und Terminierungsterm

Mit der vorherigen Wahl von  $l$ ,  $t$  und  $u$  gilt:

$$\begin{aligned}M &=_{df} \{\sigma(v) \mid \sigma \in Ch(u)\} \\ &= \{\sigma(v) \mid \sigma \in Ch(v \geq 0)\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \wedge \text{groessergleich}(\llbracket v \rrbracket_A(\sigma), \llbracket 0 \rrbracket_A(\sigma))\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \wedge \text{groessergleich}(\sigma(v), \mathbf{0}) = \mathbf{wahr}\} \\ &= \{\sigma(v) \mid \sigma \in \Sigma \wedge \sigma(v) \geq \mathbf{0}\} \\ &= \mathbf{N} \cup \{\mathbf{0}\}\end{aligned}$$

Damit haben wir insbesondere:

- ▶  $(M, <) = (\mathbf{N} \cup \{\mathbf{0}\}, <)$  ist Noethersch geordnet.
- ▶  $u[t/x] = (v \geq 0)[x/v] = x \geq 0$

# Bemerkung

Der Beweis wird schrittweise in Form einer **linearen Beweisskizze** präsentiert.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (1)

## Schritt 2 Behandlung des Rumpfs der while-Schleife...

Der Nachweis der Gültigkeit von

$$\begin{aligned} & y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \end{aligned}$$

$y := y * x;$

$x := x - 1;$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

erlaubte mithilfe der [while]-Regel den Übergang zu:

$$[y * x! = a! \wedge x > 0]$$

while  $x > 1$  do

$$\begin{aligned} & y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \end{aligned}$$

$y := y * x;$

$x := x - 1;$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

od [while]

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

246/863

# Fakultätsbeispiel: Lineare Beweisskizze (2)

Behandlung des Rumpfs der while-Schleife im Detail:

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$y := y * x;$

$x := x - 1;$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (3)

Wegen Rückwärtszuweisungsregel wird der Rumpf der while-Schleife von hinten nach vorne bearbeitet:

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$y := y * x;$$

$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$$x := x - 1; [\text{ass}]$$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13



# Fakultätsbeispiel: Lineare Beweisskizze (4)

Nach abermaliger Anwendung der [ass]-Regel erhalten wir...

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$$[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$$y := y * x; [\text{ass}]$$

$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$$x := x - 1; [\text{ass}]$$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

...wobei noch eine "Beweislücke" verbleibt!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

249/863

# Fakultätsbeispiel: Lineare Beweisskizze (5)

Schluss der “Beweislücke” in der zugrundeliegenden Theorie:

$$\begin{aligned} & y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \end{aligned}$$

$\Downarrow$  [cons]

$$[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$$y := y * x; \text{ [ass]}$$

$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$$x := x - 1; \text{ [ass]}$$

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

# Fakultätsbeispiel: Lineare Beweisskizze (6)

Anwendung der [while]-Regel liefert nun wie gewünscht:

$$[y * x! = a! \wedge x > 0]$$

while  $x > 1$  do

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$

$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$\Downarrow$  [cons]

$$[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$y := y * x$ ; [ass]

$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$x := x - 1$ ; [ass]

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

od [while]

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

251/863

# Fakultätsbeispiel: Lineare Beweisskizze (7)

Schritt 3: Zur gewünschten Nachbedingung verbleibt offenbar ebenfalls eine Beweislücke:

$$[y * x! = a! \wedge x > 0]$$

while  $x > 1$  do

$$y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0$$
$$[y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w]$$

$\Downarrow$  [cons]

$$[(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$y := y * x$ ; [ass]

$$[y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w]$$

$x := x - 1$ ; [ass]

$$[y * x! = a! \wedge x > 0 \wedge x < w]$$

od [while]

$$[y * x! = a! \wedge x > 0 \wedge \neg(x > 1)]$$

$$[y = a!]$$

# Fakultätsbeispiel: Lineare Beweisskizze (8)

Schluss der Beweislücke in der zugrundeliegenden Theorie:

$$\begin{aligned} & [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \quad \quad \Downarrow [\text{cons}] \\ & [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \quad \quad \text{od } [\text{while}] \\ & [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y * x! = a! \wedge x > 0 \wedge x \leq 1] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y * x! = a! \wedge x = 1] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (9)

Aus Platzgründen etwas verkürzt dargestellt:

$$\begin{aligned} & [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \quad \quad \Downarrow [\text{cons}] \\ & \quad \quad [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \quad \text{od } [\text{while}] \\ & [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad \quad [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

254/863

# Fakultätsbeispiel: Lineare Beweisskizze (10)

Schritt 4 Es verbleibt, die Beweislücke zur gewünschten Vorbedingung zu schließen:

$$\begin{aligned} & [a > 0] \\ & \quad x := a; \\ & \quad y := 1; \\ & \quad [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \quad \Downarrow [\text{cons}] \\ & \quad [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad y := y * x; [\text{ass}] \\ & \quad \quad [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad x := x - 1; [\text{ass}] \\ & \quad \quad [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \quad \text{od } [\text{while}] \\ & \quad [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \quad \Downarrow [\text{cons}] \\ & \quad [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (11)

Einmalige Anwendung der [ass]-Regel liefert:

$$\begin{array}{l} [a > 0] \\ \\ x := a; \\ [1 * x! = a! \wedge x > 0] \\ y := 1; [ass] \\ [y * x! = a! \wedge x > 0] \\ \text{while } x > 1 \text{ do} \\ \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ \quad \Downarrow [cons] \\ [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ \quad y := y * x; [ass] \\ [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ \quad x := x - 1; [ass] \\ [y * x! = a! \wedge x > 0 \wedge x < w] \\ \quad \text{od [while]} \\ [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ \quad \Downarrow [cons] \\ [y = a!] \end{array}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13



# Fakultätsbeispiel: Lineare Beweisskizze (12)

Abermalige Anwendung der [ass]-Regel liefert:

$$\begin{aligned} & [a > 0] \\ & [1 * a! = a! \wedge a > 0] \\ & \quad x := a; [\text{ass}] \\ & [1 * x! = a! \wedge x > 0] \\ & \quad y := 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad y := y * x; [\text{ass}] \\ & [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad x := x - 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \text{od } [\text{while}] \\ & [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \quad \Downarrow [\text{cons}] \\ & [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Fakultätsbeispiel: Lineare Beweisskizze (13)

Schluss der letzten Beweislücke in der zugrundeliegenden Theorie:

$$\begin{aligned} & [a > 0] \\ & \Downarrow [\text{cons}] \\ & [1 * a! = a! \wedge a > 0] \\ & \quad x := a; [\text{ass}] \\ & [1 * x! = a! \wedge x > 0] \\ & \quad y := 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \Downarrow [\text{cons}] \\ & [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad y := y * x; [\text{ass}] \\ & [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad x := x - 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \text{od } [\text{while}] \\ & [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \Downarrow [\text{cons}] \\ & [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Insgesamt (14)

$$\begin{aligned} & [a > 0] \\ & \Downarrow [\text{cons}] \\ & [1 * a! = a! \wedge a > 0] \\ & \quad x := a; [\text{ass}] \\ & [1 * x! = a! \wedge x > 0] \\ & \quad y := 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0] \\ & \quad \text{while } x > 1 \text{ do} \\ & \quad \quad y * x! = a! \wedge x > 0 \wedge x > 1 \Rightarrow x \geq 0 \\ & \quad [y * x! = a! \wedge x > 0 \wedge x > 1 \wedge x = w] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [(y * x) * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad y := y * x; [\text{ass}] \\ & [y * (x - 1)! = a! \wedge x - 1 > 0 \wedge x - 1 < w] \\ & \quad \quad x := x - 1; [\text{ass}] \\ & [y * x! = a! \wedge x > 0 \wedge x < w] \\ & \quad \quad \text{od } [\text{while}] \\ & [y * x! = a! \wedge x > 0 \wedge \neg(x > 1)] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y * x! = a! \wedge x > 0 \wedge x \leq 1] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y * x! = a! \wedge x = 1] \\ & \quad \quad \Downarrow [\text{cons}] \\ & [y = a!] \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

259/863

# Fakultätsbeispiel: Lineare Beweisskizze (15)

Damit haben wir wie gewünscht insgesamt gezeigt:

Das Hoaresche Tripel

$$[a > 0]$$

$x := a; y := 1; \text{ while } x > 1 \text{ do } y := y * x; x := x - 1 \text{ od}$

$$[y = a!]$$

ist gültig im Sinne totaler Korrektheit.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

**4.6**

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Kapitel 4.7

## Automatische Ansätze axiomatischer Programmverifikation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

**4.7**

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Ansätze zu (semi-) automatischer axiomatischer Programmverifikation

Unter anderem:

- ▶ [Theorema](#), RISC, JKU Linz
- ▶ [KeY-Hoare](#), KIT Karlsruhe, Chalmers University of Technology, TU Darmstadt
- ▶ ...

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

262/863

# Theorema-Projekt, [www.theorema.org](http://www.theorema.org) (1)

“The Theorema project aims at extending current computer algebra systems by facilities for supporting mathematical proving. The present early-prototype version of the Theorema software system is implemented in Mathematica. The system consists of a general higher-order predicate logic prover and a collection of special provers that call each other depending on the particular proof situations. The individual provers imitate the proof style of human mathematicians and produce human-readable proofs in natural language presented in nested cells. The special provers are intimately connected with the functors that build up the various mathematical domains.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

| 263 / 863

# Theorema-Projekt, [www.theorema.org](http://www.theorema.org) (2)

The long-term goal of the project is to produce a complete system which supports the mathematician in creating interactive text-books, i.e. books containing, besides the ordinary passive text, active text representing algorithms in executable format, as well as proofs which can be studied at various levels of detail, and whose routine parts can be automatically generated. This system will provide a uniform (logic and software) framework in which a working mathematician, without leaving the system, can get computer-support while looping through all phases of the mathematical problem solving cycle. [...]"

(Exzerpt von <http://www.theorema.org>)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

264/863



# KeY-Projekt, [www.key-project.org](http://www.key-project.org) (1)

## Integrated Deductive Software Design

“The KeY System is a formal software development tool that aims to integrate design, implementation, formal specification, and formal verification of object-oriented software as seamlessly as possible. At the core of the system is a novel theorem prover for the first-order Dynamic Logic for Java with a user-friendly graphical interface.

The project was started in November 1998 at the University of Karlsruhe. It is now a joint project of Karlsruhe Institute of Technology and Chalmers University of Technology, Gothenburg, and TU Darmstadt.

The KeY tool is available for down-load. [...]”

(Exzerpt von <http://www.key-project.org>)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

|265/863

# KeY-Projekt, [www.key-project.org](http://www.key-project.org) (2)

KeY-Hoare, [www.key-project.org/download/hoare](http://www.key-project.org/download/hoare),  
unterstützt

- ▶ partielle Korrektheitsbeweise
- ▶ totale Korrektheitsbeweise und  
Ausführungszeitkorrektheitsbeweise (Versionen ab 0.1.6)
- ▶ ganzzahlige und Boolesche Felder (Versionen ab 0.1.7)

## Nützliche Handreichung:

- ▶ Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in a course on Program Verification at the Department of Computer Science at the Chalmers University of Technology on the Hoare Calculus and the usage of the tool KeY-Hoare, 19 pages.

<http://i12www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf>

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

|266/863

# Kapitel 4.8

## Historische Meilensteine der Programmverifikation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

**4.8**

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Meilensteine der Programmverifikation (1)

## Erste Anfänge

- 1949 Turings Vision: Korrekte Programme  
Beispiel Fakultätsfunktion: Zusicherungen und Terminierungsfunktion

## Axiomatische Methode

- 1967 Floyd: Flussdiagramme  
Hoare: while-Programme

## Erweiterung dieser Methode

- 1971 Hoare: Rekursive Prozeduren
- 1976/77 Owicki & Gries, Lamport: parallele Programme
- 1980/81 Apt, Francez & de Roever, Levin & Gries: verteilte Programme
- 1991 de Boer: parallele, objekt-orientierte Programme
- 1977 Pnueli: Temporale Logik für Programme
- 1979 Clarke: Grenzen der axiomatischen Methode

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

|268/863

# Meilensteine der Programmverifikation (2)

## Automatisierung der Verifikation

- 1981/82 Emerson & Clarke, Quielle & Sifakis: Model Checking
- 1977 Cousot & Cousot: Abstrakte Interpretation
- 1979 Deduktion: interaktive Theorembeweiser
- 1967 Automatische Terminierungsbeweise

## Entwicklung korrekter Programme

- 1976 Dijkstra: Kalkül der schwächsten Vorbedingung
- 1997 Meyer: Design-by-Contract
- 1969 Büchi & Landweber: Automatenbasierte Systeme

Quelle: Ernst-Rüdiger Olderog, Reinhard Wilhelm. [Turing und die Verifikation](#). Informatik Spektrum 35(4):271-279, 2012.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10




Kap. 11

Kap. 12

Kap. 13

269/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (1)

-  Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part 1*. ACM Transactions on Programming Languages and Systems 3:431-483, 1981.
-  Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism*. Theoretical Computer Science 28(1-2):83-109, 1984.
-  Krzysztof R. Apt, Ernst-Rüdiger Olderog. *Programmverifikation – Sequentielle, parallele und verteilte Programme*. Springer-V., 1994.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10




Kap. 11

Kap. 12





Kap. 13

270/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (2)


-  Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. Springer-V., 3rd edition, 2009. (Chapter 3, While Programs; Chap. 3.3, Verification; Chapter 3.4, Proof Outlines – Partial Correctness, Total Correctness; Chapter 3.5, Completeness)
-  Bernhard Beckert, Reiner Hähnle, Peter H. Schmitt (Hrsg.). *Verification of Object-Oriented Software: The KeYApproach*. LNCS 4334, Springer-V., 2007.
-  Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2nd edition, Springer-V., 2001. (Chapter 9, Programs: Semantics and Verification)

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (3)

-  Stephen A. Cook. *Soundness and Completeness of an Axiom System for Program Verification*. SIAM Journal on Computing 7(1):70-90, 1978.
-  Jaco W. De Backer. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.
-  Edmund M. Clarke. *Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*. Journal of the ACM 26(1):129-147, 1979.
-  Edmund M. Clarke, Stephen M. German, Joseph Y. Halpern. *Effective Axiomatizations of Hoare Logics*. Journal of the ACM 30(1):612-636, 1983.



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (4)

 Robert W. Floyd. *Assigning Meaning to Programs*. In Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, 19:19-32, 1967.

 Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in the course Program Verification at the Department of Computer Science at the Chalmers University of Technology, 19 pages. [www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf](http://www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf)

 Charles A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12(19):576-580, 583, 1969.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12



Kap. 13

273/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (5)

-  Tudor Jebelean, Laura Kovács, Nikolaj Popov. *Experimental Program Verification in the Theorema System*. In Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 92-99, 2004. [www.risc.jku.at/publications/download/risc\\_2243/KoPoJeb.pdf](http://www.risc.jku.at/publications/download/risc_2243/KoPoJeb.pdf)
-  Laura Kovács, Tudor Jebelean. *Practical Aspects of Imperative Program Verification using Theorema*. In Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003), 317-320, 2003. [www.risc.jku.at/publications/download/risc\\_464/synasc03.pdf](http://www.risc.jku.at/publications/download/risc_464/synasc03.pdf)

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (6)

-  Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. [www.risc.jku.at/publications/download/risc\\_2053/2003-11-06-A.pdf](http://www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf)
-  Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 1, Introduction: What do we want to know about the Program?, Chapter 2, How to prove a Program Correct: Programs without Loops; Chapter 3, How to prove a Program Correct: Iterative Programs)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10




Kap. 11

Kap. 12

Kap. 13

| 275 / 863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (7)

-  Jacques Loeckx, Kurt Sieber. *The Foundations of Program Verification*. Wiley, 1984.
-  Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008. (Chapter 19, Program Correctness Proofs; Chapter 19.3, Proofs using Floyd's Method of Invariant Assertions; Chapter 20.2.1, Floyd-Hoare Logic)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6, Axiomatic Program Verification)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9




Kap. 10

Kap. 11


Kap. 12

Kap. 13

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (8)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 9, Axiomatic Program Verification; Chapter 10, More on Axiomatic Program Verification)
-  David von Oheimb. *Hoare Logic for Java in Isabelle/HOL*. *Concurrency and Computation: Practice and Experience* 13(13):1173-1214, 2001.
-  Ernst-Rüdiger Olderog. *Correctness of Programs with Pascal-like Procedures without Global Variables*. *Theoretical Computer Science* 30(1):49-90, 1984.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 4 (9)

-  Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.
-  Ernst-Rüdiger Olderog, Reinhard Wilhelm. *Turing und die Verifikation*. Informatik Spektrum 35(4):271-279, 2012.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

4.1

4.2

4.3

4.4

4.5

4.6

4.7

4.8

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

278/863

# Kapitel 5

## Worst-Case Execution Time Analyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Motivation

Von Verifikation zu Analyse:

- ▶ [Worst-Case Execution Time](#)-Analyse als erstes Beispiel

...nach

- ▶ Hanne Riis Nielson, Flemming Nielson. [Semantics with Applications – A Formal Introduction](#). Wiley, 1992.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

280/863



# Worst-Case Execution Time (WCET)-Analyse

## Motivation:

- ▶ In vielen Anwendungsbereichen sind Aussagen über die Ausführungszeit erforderlich.
- ▶ Der Nachweis totaler Korrektheit garantiert zwar Terminierung, sagt aber nichts über den Ressourcen-, speziell den Zeitbedarf aus.

## In der Folge:

- ▶ Erweiterung und Adaptierung des Beweissystems für totale Korrektheit, um solche Aussagen zu ermöglichen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

281/863

# Die grundlegende Idee (1)

...zur Zuordnung von Ausführungszeiten:

- ▶ **Leere Anweisung**

...Ausführungszeit in  $\mathcal{O}(1)$ , d.h. Ausführungszeit ist beschränkt durch eine Konstante.

- ▶ **Zuweisung**

...Ausführungszeit in  $\mathcal{O}(1)$ .

- ▶ **(Sequentielle) Komposition**

...Ausführungszeit entspricht, bis auf einen konstanten Faktor, der Summe der Ausführungszeiten der Komponenten.

# Die grundlegende Idee (2)

- ▶ **Fallunterscheidung**

...Ausführungszeit entspricht, bis auf einen konstanten Faktor, der größeren der Ausführungszeiten der beiden Zweige.

- ▶ **(while)-Schleife**

...Ausführungszeit der Schleife entspricht, bis auf einen konstanten Faktor, der Summe der wiederholten Ausführungszeiten des Rumpfes der Schleife.

**Bemerkung:** Verfeinerungen der Granularität sind offenbar möglich.

...dieser grundlegenden Idee in 3 Schritten:

1. Angabe einer Semantik, die die Auswertungszeit arithmetischer und Boolescher Ausdrücke beschreibt.
2. Erweiterung und Adaption der natürlichen Semantik von *WHILE* zur Bestimmung der Ausführungszeit eines Programms.
3. Erweiterung und Adaption des Beweissystems für totale Korrektheit zum Nachweis über die Größenordnung der Ausführungszeit von Programmen.

# Erster Schritt

Festlegung von (abstrakten) Semantikfunktionen

- ▶  $\llbracket \cdot \rrbracket_{TA} : \mathbf{Aexpr} \rightarrow \mathbb{Z}$  und
- ▶  $\llbracket \cdot \rrbracket_{TB} : \mathbf{Bexpr} \rightarrow \mathbb{Z}$

zur Beschreibung der Auswertungszeit arithmetischer und Boolescher Ausdrücke (in Zeiteinheiten einer abstrakten Maschine).

# Semantik zur Ausführungszeit der Auswertung arithmetischer Ausdrücke

$\llbracket \cdot \rrbracket_{TA} : \mathbf{Aexpr} \rightarrow \mathbb{Z}$  induktiv definiert durch

- ▶  $\llbracket n \rrbracket_{TA} =_{df} \mathbf{1}$
- ▶  $\llbracket x \rrbracket_{TA} =_{df} \mathbf{1}$
- ▶  $\llbracket a_1 + a_2 \rrbracket_{TA} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶  $\llbracket a_1 * a_2 \rrbracket_{TA} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶  $\llbracket a_1 - a_2 \rrbracket_{TA} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶  $\llbracket a_1 / a_2 \rrbracket_{TA} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶ ... (andere Operatoren analog, ggf. auch mit operations-spezifischen Kosten)

# Semantik zur Ausführungszeit der Auswertung Boolescher Ausdrücke

$\llbracket \cdot \rrbracket_{TB} : \mathbf{Bexpr} \rightarrow \mathbb{Z}$  induktiv definiert durch

- ▶  $\llbracket true \rrbracket_{TB} =_{df} \mathbf{1}$
- ▶  $\llbracket false \rrbracket_{TB} =_{df} \mathbf{1}$
- ▶  $\llbracket a_1 = a_2 \rrbracket_{TB} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶  $\llbracket a_1 < a_2 \rrbracket_{TB} =_{df} \llbracket a_1 \rrbracket_{TA} + \llbracket a_2 \rrbracket_{TA} + \mathbf{1}$
- ▶ ... (andere Relatoren (z.B.  $\leq$ , ...) analog)
  
- ▶  $\llbracket \neg b \rrbracket_{TB} =_{df} \llbracket b \rrbracket_{TB} + \mathbf{1}$
- ▶  $\llbracket b_1 \wedge b_2 \rrbracket_{TB} =_{df} \llbracket b_1 \rrbracket_{TB} + \llbracket b_2 \rrbracket_{TB} + \mathbf{1}$
- ▶  $\llbracket b_1 \vee b_2 \rrbracket_{TB} =_{df} \llbracket b_1 \rrbracket_{TB} + \llbracket b_2 \rrbracket_{TB} + \mathbf{1}$

# Anmerkungen zu $\llbracket \cdot \rrbracket_{TA}$ und $\llbracket \cdot \rrbracket_{TB}$

## Die Semantikfunktionen

- ▶  $\llbracket \cdot \rrbracket_{TA}$
- ▶  $\llbracket \cdot \rrbracket_{TB}$

beschreiben intuitiv die Anzahl der Zeiteinheiten, die eine (hier nicht spezifizierte) abstrakte Maschine zur Auswertung arithmetischer und Boolescher Ausdrücke benötigt.



# Zweiter Schritt

Erweiterung und Anpassung der

- ▶ natürlichen Semantik von *WHILE*

zur Bestimmung der Ausführungszeit von Programmen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Übergang von Transitionen der Form

$$\langle \pi, \sigma \rangle \rightarrow \sigma'$$

zu Transitionen der Form

$$\langle \pi, \sigma \rangle \xrightarrow{t} \sigma'$$

mit der Bedeutung, dass  $\pi$  angesetzt auf  $\sigma$  nach  $t$  **Zeiteinheiten** in  $\sigma'$  terminiert.

# N-Semantik erweitert um den Zeitaspekt (1)

...für das Beispiel von *WHILE*:

$$[\text{skip}_{tns}] \quad \frac{\overline{\quad}}{\langle \text{skip}, \sigma \rangle \rightarrow^1 \sigma}$$

$$[\text{ass}_{tns}] \quad \frac{\overline{\quad}}{\langle x := t, \sigma \rangle \rightarrow^{[t]_{TA}+1} \sigma[[t]_A(\sigma)/x]}$$

$$[\text{comp}_{tns}] \quad \frac{\langle \pi_1, \sigma \rangle \rightarrow^{t_1} \sigma', \langle \pi_2, \sigma' \rangle \rightarrow^{t_2} \sigma''}{\langle \pi_1; \pi_2, \sigma \rangle \rightarrow^{t_1+t_2} \sigma''}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

291/863

# N-Semantik erweitert um den Zeitaspekt (2)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

292/863

$$[\text{if}_{tns}^{tt}] \frac{\langle \pi_1, \sigma \rangle \rightarrow^t \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow^{[b]_{TB+t+1}} \sigma'} \quad \llbracket b \rrbracket_B(\sigma) = \mathbf{wahr}$$

$$[\text{if}_{tns}^{ff}] \frac{\langle \pi_2, \sigma \rangle \rightarrow^t \sigma'}{\langle \text{if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi}, \sigma \rangle \rightarrow^{[b]_{TB+t+1}} \sigma'} \quad \llbracket b \rrbracket_B(\sigma) = \mathbf{falsch}$$

$$[\text{while}_{tns}^{tt}] \frac{\langle \pi, \sigma \rangle \rightarrow^t \sigma', \langle \text{while } b \text{ do } \pi \text{ od}, \sigma' \rangle \rightarrow^{t'} \sigma''}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[b]_{TB+t+t'+2}} \sigma''} \quad \llbracket b \rrbracket_B(\sigma) = \mathbf{wahr}$$

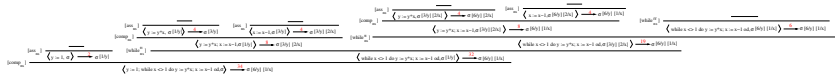
$$[\text{while}_{tns}^{ff}] \frac{\text{—}}{\langle \text{while } b \text{ do } \pi \text{ od}, \sigma \rangle \rightarrow^{[b]_{TB+3}} \sigma} \quad \llbracket b \rrbracket_B(\sigma) = \mathbf{falsch}$$

# Beispiel zur nat. "Zeit"-Semantik (1)

Sei  $\sigma \in \Sigma$  mit  $\sigma(x) = 3$ .

Dann gilt:

$$\langle y := 1; \text{ while } x \neq 1 \text{ do } y := y * x; x := x - 1 \text{ od, } \sigma \rangle \longrightarrow \sigma[\mathbf{6}/y][\mathbf{1}/x]$$



- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- Kap. 5**
- Kap. 6
- Kap. 7
- Kap. 8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13
- Kap. 14
- Kap. 15
- Kap. 16
- Kap. 17

# Beispiel zur nat. “Zeit”-Semantik (2)

Das gleiche Beispiel in “etwas” gefälligerer Darstellung:

$$\begin{array}{c}
 \frac{[ass_m] \frac{}{\langle y := 1, \sigma \rangle \xrightarrow{2} \sigma [1/y]} \quad [comp_m] \frac{[ass_m] \frac{}{\langle y := y^*x, \sigma [1/y] \rangle \xrightarrow{4} \sigma [3/y]} \quad [ass_m] \frac{}{\langle x := x-1, \sigma [3/y] \rangle \xrightarrow{4} \sigma [3/y] [2/x]} \quad [while_m^H] \frac{}{\langle y := y^*x; x := x-1, \sigma [1/y] \rangle \xrightarrow{8} \sigma [3/y] [2/x]} \quad [while_m^H] \frac{}{\langle while\ x < 1\ do\ y := y^*x; x := x-1\ od, \sigma [1/y] \rangle \xrightarrow{32} \sigma [6/y] [1/x]} }{[comp_m] \frac{}{\langle y := 1; while\ x < 1\ do\ y := y^*x; x := x-1\ od, \sigma \rangle \xrightarrow{34} \sigma [6/y] [1/x]} } T_{19}
 \end{array}$$

$$\begin{array}{c}
 \frac{[while_m^H] \frac{}{\langle while\ x < 1\ do\ y := y^*x; x := x-1\ od, \sigma [6/y] [1/x] \rangle \xrightarrow{6} \sigma [6/y] [1/x]} \quad [comp_m] \frac{[ass_m] \frac{}{\langle y := y^*x, \sigma [3/y] [2/x] \rangle \xrightarrow{4} \sigma [6/y] [2/x]} \quad [ass_m] \frac{}{\langle x := x-1, \sigma [6/y] [2/x] \rangle \xrightarrow{4} \sigma [6/y] [1/x]} \quad [while_m^H] \frac{}{\langle while\ x < 1\ do\ y := y^*x; x := x-1\ od, \sigma [3/y] [2/x] \rangle \xrightarrow{8} \sigma [6/y] [1/x]} }{[while_m^H] \frac{}{\langle while\ x < 1\ do\ y := y^*x; x := x-1\ od, \sigma [3/y] [2/x] \rangle \xrightarrow{19} \sigma [6/y] [1/x]} } T_{19}
 \end{array}$$

- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- Kap. 5**
- Kap. 6
- Kap. 7
- Kap. 8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13
- Kap. 14
- Kap. 15
- Kap. 16
- Kap. 17

# Dritter Schritt

Erweiterung und Anpassung der

- ▶ des Beweiskalküls für totale Korrektheit

um den Ausführungszeitaspekt von Programmen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Idee (1)

Übergang zu **Korrektheitsformeln** der Form

$$\{p\} \pi \{e \Downarrow q\}$$

wobei

- ▶  $p$  und  $q$  logische Formeln (wie bisher!) und
- ▶  $e \in \mathbf{Aexp}$  ein arithmetischer Ausdruck ist.



## Idee (2)

### Definition (5.1, Gültigkeit von Korrektheitsformeln)

Die **Korrektheitsformel**

$$\{p\} \pi \{e \Downarrow q\}$$

ist **gültig** gdw. für jeden Anfangszustand  $\sigma$  gilt: Ist die Vorbedingung  $p$  in  $\sigma$  erfüllt, **dann** terminiert die zugehörige Berechnung von  $\pi$  angesetzt auf  $\sigma$  regulär mit einem Endzustand  $\sigma'$  **und** die Nachbedingung  $q$  ist in  $\sigma'$  erfüllt, und die benötigte Ausführungszeit ist in  $\mathcal{O}(e)$ .

## Idee (3)

Anders ausgedrückt:

Die **Korrektheitsformel**

$$\{p\} \pi \{e \Downarrow q\}$$

ist **gültig** (in Zeichen:  $\models \{p\} \pi \{e \Downarrow q\}$ ) gdw.

es existiert eine natürliche Zahl **k**, so dass für alle Zustände  $\sigma$  gilt:

Ist die Vorbedingung  $p$  in  $\sigma$  erfüllt, dann gibt es einen Zustand  $\sigma'$  und eine natürliche Zahl **t**, so dass die Nachbedingung  $q$  in  $\sigma'$  erfüllt ist und weiters gilt:

$$t \leq \mathbf{k} * \llbracket e \rrbracket_A(\sigma)$$

# Idee (4)

## Beachte:

- ▶ Im Ausdruck

$$t \leq k * \llbracket e \rrbracket_A(\sigma)$$

wird der Ausdruck  $e$  im Anfangszustand  $\sigma$  ausgewertet, nicht im terminalen Zustand  $\sigma'$ .

- ▶ Diesem Umstand ist geschuldet, dass die (jetzt folgende) Festlegung der Regeln  $[comp_e]$  und  $[while_e]$  komplizierter ausfällt als möglicherweise zunächst vermutet.

# Intuition zu den Kalkülregeln (1)

...für  $[comp_e]$ :

- ▶ Die  $[comp_e]$ -Regel setzt voraus, dass es Beweise gibt, die zeigen, dass  $e_1$  und  $e_2$  die Größenordnung der Zahl der Schritte angeben zur Ausführung von  $\pi_1$  und  $\pi_2$ .
- ▶  $e_1$  drückt dies für  $\pi_1$  relativ zum Anfangszustand von  $\pi_1$  aus;  $e_2$  drückt dies für  $\pi_2$  relativ zum Anfangszustand von  $\pi_2$  aus.
- ▶ Aus diesem Grund drückt nicht einfach die Summe  $e_1 + e_2$  das Zeitverhalten der sequentiellen Komposition  $\pi_1; \pi_2$  aus.
- ▶ Vielmehr muss  $e_2$  durch einen Ausdruck  $e_2'$  ersetzt werden, so dass  $e_2'$  ausgewertet im Anfangszustand von  $e_1$  den Wert von  $e_2$  im Anfangszustand von  $\pi_2$  beschränkt.
- ▶ Dies wird durch die erweiterte Vor- und Nachbedingung von  $\pi_1$  unter Verwendung der frischen logischen Variable  $u$  erreicht.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

300/863

## Intuition zu den Kalkülregeln (2)

...für  $[while_e]$ :

- ▶ Die  $[while_e]$ -Regel geht davon aus, dass die Ausführungszeit des Schleifenrumpfs durch  $e_1$ , die der gesamten Schleife durch  $e$  beschränkt ist. Wie für die  $[comp_e]$ -Regel drückt die Summe  $e_1 + e$  nicht das Zeitverhalten der gesamten Schleife aus, da  $e_1$  sich auf den Zustand vor Ausführung des Rumpfs der while-Schleife Bezug nimmt und  $e$  auf den Zustand nach einmaliger Ausführung des Schleifenrumpfs.
- ▶ Ähnlich wie für die  $[comp_e]$ -Regel wird ein Ausdruck  $e'$  benötigt, der vor Ausführung des Schleifenrumpfs ausgewertet Ausdruck  $e$  nach dessen Ausführung beschränkt.
- ▶ Dann muss gelten, dass  $e$  die Ungleichung  $e \geq e_1 + e'$  erfüllt, da  $e$  die Ausführung der while-Schranke unabhängig von der Anzahl ihrer Wiederholungen beschränken muss.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

301/863

# Axiomatische Semantik mit Zeitaspekt (1)

$$[\text{skip}_e] \frac{\overline{\quad}}{\{p\} \text{ skip } \{1 \Downarrow p\}}$$

$$[\text{ass}_e] \frac{\overline{\quad}}{\{p[t \setminus x]\} x := t \{1 \Downarrow p\}}$$

$$[\text{comp}_e] \frac{\{p \wedge e'_2 = u\} \pi_1 \{e_1 \Downarrow r \wedge e_2 \leq u\}, \{r\} \pi_2 \{e_2 \Downarrow q\}}{\{p\} \pi_1; \pi_2 \{e_1 + e'_2 \Downarrow q\}}$$

wobei  $u$  frische logische Variable ist

$$[\text{ite}_e] \frac{\{p \wedge b\} \pi_1 \{e \Downarrow q\}, \{p \wedge \neg b\} \pi_2 \{e \Downarrow q\}}{\{p\} \text{ if } b \text{ then } \pi_1 \text{ else } \pi_2 \text{ fi } \{e \Downarrow q\}}$$

$$[\text{cons}_e] \frac{p \Rightarrow p' \quad \{p'\} \pi \{e' \Downarrow q'\} \quad q' \Rightarrow q}{\{p\} \pi \{e \Downarrow q\}}$$

wobei (für eine natürliche Zahl  $\mathbf{k}$ ) gilt:  $e' \leq \mathbf{k} * e$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

302/863

# Axiomatische Semantik mit Zeitaspekt (2)

$$[\text{while}_e] \quad \frac{\{p(z+1) \wedge e' = u\} \pi \{e_1 \Downarrow p(z) \wedge e \leq u\}}{\{\exists z. p(z)\} \text{ while } b \text{ do } \pi \text{ od } \{e \Downarrow p(0)\}}$$

wobei gilt:

$$p(z+1) \Rightarrow (b \wedge e \geq e_1 + e'),$$

$$p(0) \Rightarrow (\neg b \wedge 1 \leq e),$$

$u$  ist eine frische logische Variable,

$z$  nimmt Werte aus den natürlichen Zahlen an,

d.h.  $z \geq 0$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

303/863

# Beispiele (1)

## Die Korrektheitsformel

$$\{x=3\}$$

```
y:=1; while x/=1 do y:=y*x; x:=x-1 od
```

$$\{1 \Downarrow \text{True}\}$$

beschreibt, dass die Ausführungszeit des Fakultätsprogramms angesetzt auf einen Zustand, in dem  $x$  den Wert **3** hat, von der Größenordnung von **1** ist, also durch eine Konstante beschränkt ist.



# Beispiele (2)

## Die Korrektheitsformel



$$\{x > 0\}$$

```
y:=1; while x/=1 do y:=y*x; x:=x-1 od
```

$$\{x \Downarrow \text{True}\}$$

beschreibt, dass die Ausführungszeit des Fakultätsprogramms angesetzt auf einen Zustand, in dem  $x$  einen Wert größer als **0** hat, von der Größenordnung von  $x$  ist, also linear beschränkt ist.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (1)

-  Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, Albrecht Kadlec. *Beyond Loop Bounds: Comparing Annotation Languages for Worst-Case Execution Time Analysis*. Journal of Software and Systems Modeling 10(3):411-437, Springer-V., 2011.
-  Björn Lisper, Andreas Ermedahl, Dietmar Schreiner, Jens Knoop, Peter Gliwa. *Practical Experiences of Applying Source-level WCET Flow Analysis to Industrial Code*. Journal of Software Tools for Technology Transfer (STTT) 15(1):53-63, Springer-V., 2013.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

L306/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (2)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 6.5, Assertions for Execution Time)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 10.2, Assertions for Execution Time)
-  Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, Gernot Gebhard. *The T-CREST Approach of Compiler and WCET-Analysis Integration*. In Proceedings of the 9th International Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2013), June 2013, to appear.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 5 (3)



Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-case Execution Time Problem – Overview of Methods and Survey of Tools*. ACM Transactions on Embedded Computing Systems 7(3):36.1-36.53, 2008.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Teil II

## Analyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

**Kap. 5**

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Kapitel 6

## Programmanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

**Kap. 6**

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Programmanalyse

...speziell Datenflussanalyse.

Typische Fragen sind:

- ▶ Welchen Wert hat eine Variable an einer Programmstelle?  
~> Konstantenausbreitung und Faltung
- ▶ Steht der Wert eines Ausdrucks an einer Programmstelle verfügbar?  
~> Elimination (partiell) redundanten Codes, Code Motion
- ▶ Ist eine Variable tot an einer Programmstelle?  
~> Elimination (partiell) toten Codes

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

311/863

# Kapitel 6.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

**6.1**

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

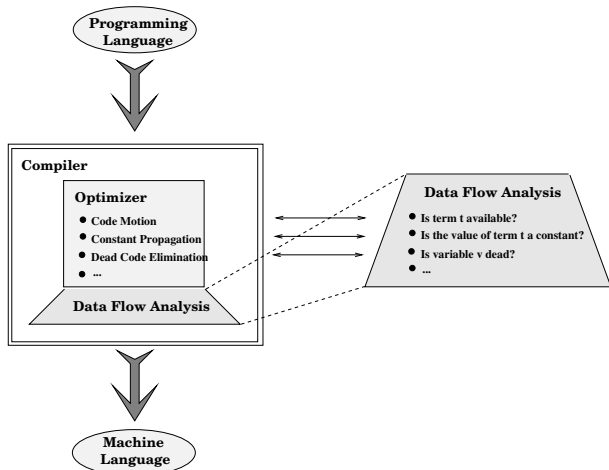
Kap. 11

Kap. 12



# Motivation

...(Programm-) Analyse zur (Programm-) Optimierung



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Zentrale Fragen

Grundlegendes:

- ▶ Was bedeutet **Optimalität**  
...in **Analyse** und in **Optimierung**?

Und auch (scheinbar) Nebensächliches:

- ▶ Was ist eine **angemessene** Programmrepräsentation?

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

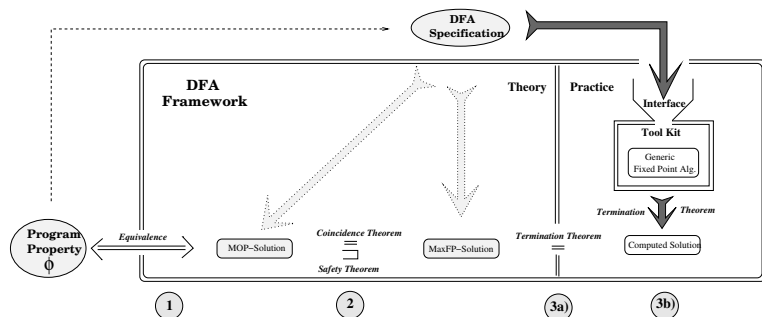
Kap. 10

Kap. 11

Kap. 12

# Überblick – DFA in Theorie und Praxis (1)

## DFA-Rahmen / DFA-Werkzeugkistensicht:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

315/863

# Überblick – DFA in Theorie und Praxis (2)

Komponenten einer (intraprozeduralen) DFA-Spezifikation:

▶ (Lokale) abstrakte Semantik

1. Ein Datenflussanalyseverband  $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top)$
2. Ein Datenflussanalysefunktional  $\llbracket \ ] : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$
3. Anfangsinformation/-zusicherung  $c_s \in \mathcal{C}$

Darauf aufbauend:

▶ Globalisierungsstrategien

1. “Meet over all Paths”-Ansatz (*MOP*)
2. Maximaler Fixpunktansatz (*MaxFP*)

▶ Generischer Fixpunktalgorithmus

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

316/863

# Überblick – DFA in Theorie und Praxis (3)

Hauptresultate:

- ▶ **Korrektheit:** Sicherheits-Theorem
- ▶ **Vollständigkeit:** Koinzidenz-Theorem

Sowie:

- ▶ **Effektivität:** Terminierungs-Theorem

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

317/863

# Überblick – DFA in Theorie und Praxis (4)

Genauer lassen sich unterscheiden:

- ▶ intraprozedurale,
- ▶ interprozedurale,
- ▶ objektorientierte,
- ▶ parallele,
- ▶ konditionale,
- ▶ ...

Datenflussanalyse (DFA).

Das grundlegende Prinzip...

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

**6.1**

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

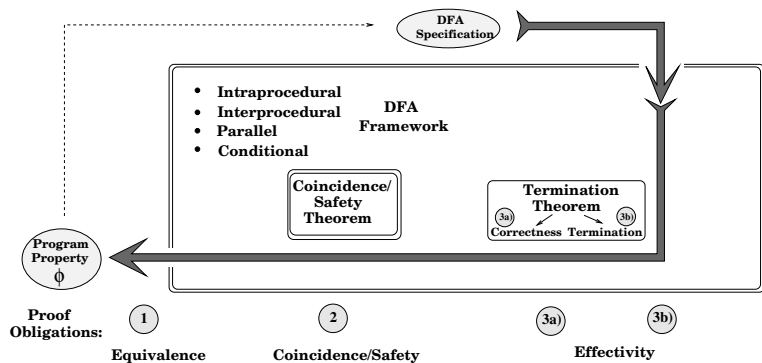
Kap. 11

Kap. 12

# Überblick – DFA in Theorie und Praxis (5)

...bleibt stets gleich!

Die allgemeine DFA-Rahmen / DFA-Werkzeugkistensicht:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

319/863

# Ziel

Optimale Programoptimierung!

...ein weißer Schimmel in der Informatik?

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

**6.1**

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12



# Ohne Fleiß kein Preis!

In der Sprechweise **optimierender Übersetzung**:

...ohne **Analyse** keine **Optimierung!**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Kapitel 6.2

## Datenflussanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Programmrepräsentation

Im Bereich der Programmanalyse, speziell **Datenflussanalyse**, ist üblich:

- ▶ die Repräsentation von Programmen in Form (nichtdeterministischer) **Flussgraphen**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

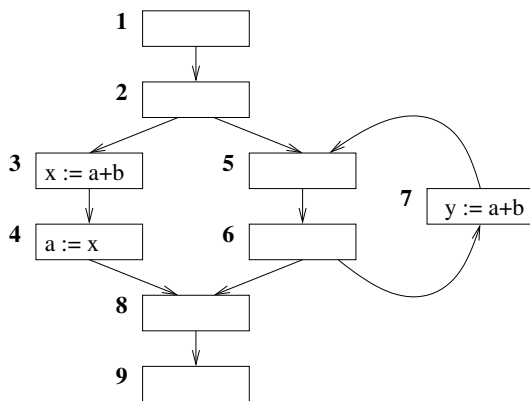
Kap. 10

Kap. 11

Kap. 12

323/863

# Beispiel



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

324/863

# Flussgraphen (1)

## Definition (6.2.1, Flussgraph)

Ein (nichtdeterministischer) **Flussgraph** ist ein Quadrupel  $G = (N, E, s, e)$  mit

- ▶ Knotenmenge (engl. **Nodes**)  $N$
- ▶ Kantenmenge (engl. **Edges**)  $E \subseteq N \times N$
- ▶ ausgezeichnetem **Startknoten**  $s$
- ▶ ausgezeichnetem **Endknoten**  $e$

Ohne Beschränkung der Allgemeinheit nehmen wir an, dass

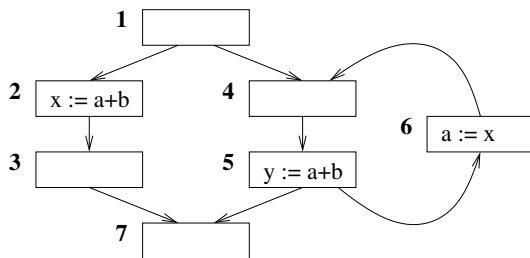
- ▶  $s$  keine **Vorgänger** (d.h. keine eingehenden Kanten) besitzt
- ▶  $e$  keine **Nachfolger** (d.h. keine ausgehenden Kanten) besitzt
- ▶ alle Knoten aus  $N$  auf einem **Pfad** von  $s$  nach  $e$  liegen

# Flussgraphen (2)

## Bemerkung:

- ▶ **Knoten** repräsentieren **Programmpunkte**
- ▶ **Kanten** die **Verzweigungsstruktur**.
- ▶ Elementare Programmanweisungen (Zuweisungen, Tests) können wahlweise durch
  - ▶ Knoten ( $\rightsquigarrow$  **knotenbenannter Flussgraph**)
  - ▶ Kanten ( $\rightsquigarrow$  **kantenbenannter Flussgraph**)repräsentiert werden.

# Beispiel: Knotenbenannter Flussgraph



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

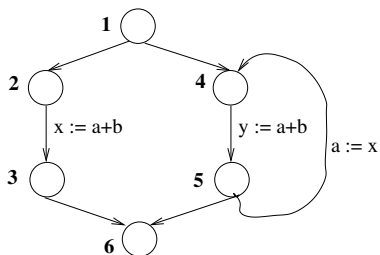
Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Beispiel: Kantenbenannter Flussgraph



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

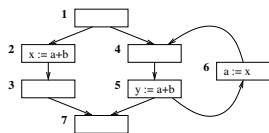
Kap. 12



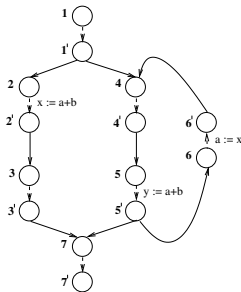
# Flussgraphendarstellungsvarianten (1)

Knoten- vs. kantenbenannte Flussgraphen  
(hier mit **Einzelanweisungsbenennung**)

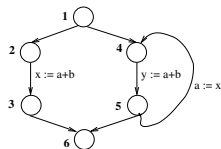
a)



b)



i) Schematisch



ii) "Optimiert"

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

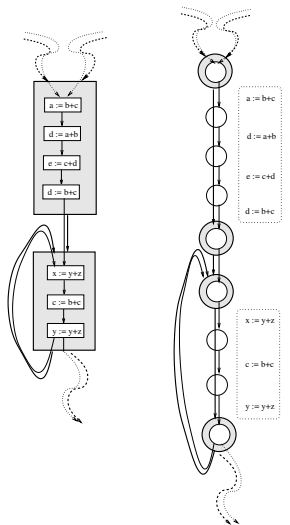
Kap. 11

Kap. 12

329/863

# Flussgraphdarstellungsvarianten (2)

Knoten- vs. kantenbenannte Flussgraphen  
(hier mit **Basisblockbenennung**)



Node-labeled (BB-) Graph

Edge-labeled (BB-) Graph

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Flussgraphdarstellungsvarianten (3)

Wir unterscheiden:

- ▶ **Knotenbenannte Graphen**
  - ▶ Einzelanweisungsgraphen (SI-Graphen)
  - ▶ Basisblockgraphen (BB-Graphen)
- ▶ **Kantenbenannte Graphen**
  - ▶ Einzelanweisungsgraphen (SI-Graphen)
  - ▶ Basisblockgraphen (BB-Graphen)

In der Folge betrachten wir bevorzugt **kantenbenannte SI-Graphen**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Bezeichnungen für Pfadmengen

Sei  $G = (N, E, s, e)$  ein Flussgraph, seien  $m, n$  zwei Knoten aus  $N$ . Dann bezeichne:

- ▶  $\mathbf{P}_G[m, n]$ : Die Menge aller Pfade von  $m$  nach  $n$ .
- ▶  $\mathbf{P}_G[m, n[$ : Die Menge aller Pfade von  $m$  zu einem Vorgänger von  $n$ .
- ▶  $\mathbf{P}_G]m, n]$ : Die Menge aller Pfade von einem Nachfolger von  $m$  nach  $n$ .
- ▶  $\mathbf{P}_G]m, n[$ : Die Menge aller Pfade von einem Nachfolger von  $m$  zu einem Vorgänger von  $n$ .

**Bemerkung:** Wenn  $G$  aus dem Kontext eindeutig hervorgeht, schreiben wir einfacher auch  $\mathbf{P}$  statt  $\mathbf{P}_G$ .

# DFA-Spezifikation, DFA-Problem

## Definition (6.2.2, DFA-Spezifikation)

Eine DFA-Spezifikation ist festgelegt durch

- ▶ eine (lokale) abstrakte Semantik bestehend aus
  1. einem DFA-Verband  $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top)$
  2. einem DFA-Funktional  $\llbracket \cdot \rrbracket : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$
- ▶ eine Anfangsinformation/-zusicherung:  $c_s \in \mathcal{C}$

## Definition (6.2.3, DFA-Problem)

Eine DFA-Spezifikation legt ein DFA-Problem fest.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

333/863

# Praktisch relevant

...sind sog.

- ▶ monotone
- ▶ distributive
- ▶ additive

DFA-Probleme über DFA-Verbänden, die die

- ▶ absteigende
- ▶ aufsteigende Kettenbedingung

erfüllen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

**6.2**

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Monotonie, Distributivität, Additivität

...von DFA-Funktionalen, DFA-Problemen:

## Definition (6.2.4)

Ein DFA-Funktional  $\llbracket \ ] : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$  heißt **monoton/distributiv/additiv** gdw für alle  $e \in E$  gilt, dass  $\llbracket e \rrbracket$  **monoton/distributiv/additiv** ist.

## Definition (6.2.5)

Ein DFA-Problem heißt **monoton/distributiv/additiv** gdw das DFA-Funktional  $\llbracket \ ]$  der zugrundeliegenden DFA-Spezifikation  $(\hat{\mathcal{C}}, \llbracket \ ], c_s)$  ist **monoton/distributiv/additiv**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

335/863

# Ausdehnung von Monotonie, Distributivität, Additivität (1)

...von Funktionen auf CPOs auf Funktionen auf (DFA-) Verbänden (vgl. Kapitel 3.3).

## Definition (6.2.6, Monotonie, Distributivität, Additivität)

Sei  $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top)$  ein vollständiger (DFA-) Verband und  $f : \mathcal{C} \rightarrow \mathcal{C}$  eine Funktion auf  $\mathcal{C}$ . Dann heißt  $f$

1. **monoton** gdw  $\forall c, c' \in \mathcal{C}. c \sqsubseteq c' \Rightarrow f(c) \sqsubseteq f(c')$   
(Erhalt der Ordnung der Elemente)
2. **distributiv** gdw  $\forall C' \subseteq \mathcal{C}. f(\sqcap C') = \sqcap \{f(c) \mid c \in C'\}$   
(Erhalt der größten unteren Schranken)
3. **additiv** gdw  $\forall C' \subseteq \mathcal{C}. f(\sqcup C') = \sqcup \{f(c) \mid c \in C'\}$   
(Erhalt der kleinsten oberen Schranken)



# Ausdehnung von Monotonie, Distributivität, Additivität (2)

Analog zum entsprechenden Lemma auf CPOs gilt auch auf (DFA-) Verbänden folgende oft nützliche äquivalente Charakterisierung der Monotonie (vgl. Kapitel 3.3):

## Lemma (6.2.7)

Sei  $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top)$  ein vollständiger (DFA-) Verband und  $f : \mathcal{C} \rightarrow \mathcal{C}$  eine Funktion auf  $\mathcal{C}$ . Dann gilt:

$$f \text{ ist monoton} \iff \forall C' \subseteq \mathcal{C}. f(\sqcap C') \sqsubseteq \sqcap \{f(c) \mid c \in C'\}$$

# Ausdehnung von absteigender/aufsteigender Kettenbedingung

...von CPOs auf (DFA-) Verbände (vgl. Kapitel 3.3):

## Definition (6.2.8, Ab-/aufsteigende Kettenbeding'g)

Ein (DFA-) Verband  $\hat{\mathcal{C}} = (\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top)$  erfüllt

1. die **absteigende Kettenbedingung**, falls jede absteigende Kette stationär wird, d.h. für jede Kette  $c_1 \sqsupseteq c_2 \sqsupseteq \dots \sqsupseteq c_n \sqsupseteq \dots$  gibt es einen Index  $m \geq 1$  so dass  $c_m = c_{m+j}$  für alle  $j \in \mathbb{N}$  gilt
2. die **aufsteigende Kettenbedingung**, falls jede aufsteigende Kette stationär wird, d.h. für jede Kette  $c_1 \sqsubseteq c_2 \sqsubseteq \dots \sqsubseteq c_n \sqsubseteq \dots$  gibt es einen Index  $m \geq 1$  so dass  $c_m = c_{m+j}$  für alle  $j \in \mathbb{N}$  gilt

# Nächstes Ziel

...die Globalisierung lokaler abstrakter Semantiken von Anweisungen auf Flussgraphen.

Dafür zwei (Globalisierungs-) Strategien:

- ▶ “Meet over all Paths”-Ansatz (*MOP*)  
~> führt auf spezifizierende Lösung eines DFA-Problems.
- ▶ Maximaler Fixpunktansatz (*MaxFP*)  
~> führt auf berechenbare Lösung eines DFA-Problems.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

339/863

# Kapitel 6.3

## *MOP*-Ansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

**6.3**

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Der MOP-Ansatz

Grundlegend für die MOP-Strategie:

Definition (6.3.1, Pfadausdehnung von  $\llbracket \cdot \rrbracket$ )

Die Ausdehnung einer lokalen abstrakten Semantik  $\llbracket \cdot \rrbracket$  auf Pfade  $p = \langle e_1, e_2, \dots, e_q \rangle$  ist definiert durch

$$\llbracket p \rrbracket =_{df} \begin{cases} Id_{\mathcal{C}} & \text{falls } q < 1 \\ \llbracket \langle e_2, \dots, e_q \rangle \rrbracket \circ \llbracket e_1 \rrbracket & \text{sonst} \end{cases}$$

wobei  $Id_{\mathcal{C}}$  die Identität auf  $\mathcal{C}$  bezeichnet.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

341/863

# Die MOP-Lösung

## Definition (6.3.2, MOP-Lösung)

Sei  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  die Spezifikation eines DFA-Problems. Dann ist für alle Knoten  $n \in N$  die **MOP-Lösung** definiert durch:

$$MOP_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}(n) = \bigsqcap \{ \llbracket p \rrbracket(c_s) \mid p \in \mathbf{P}[s, n] \}$$

Zentral:

- ▶ Die **MOP-Lösung** ist das Maß aller **DFA-Dinge**. Sie ist die **spezifizierende** Lösung eines durch  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  gegebenen intraprozeduralen DFA-Problems.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

**6.3**

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

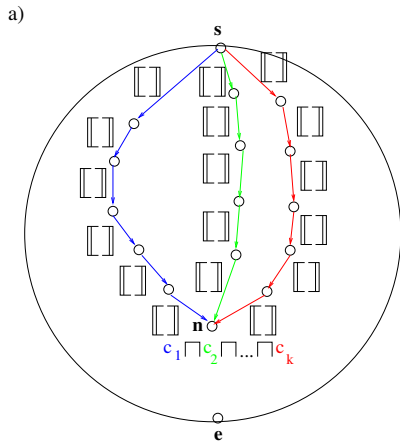
Kap. 10

Kap. 11

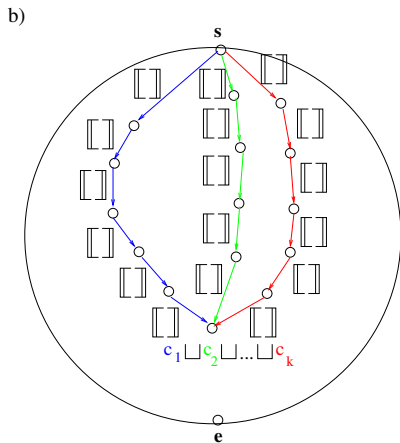
Kap. 12

342/863

# Veranschaulichung



a) Traditionelle DFA-Sicht:  
'Schneiden' v. Infos: *MOP*  
(Allquantifizierung)



b) Traditionelle AI-Sicht:  
"Vereinigen" v. Infos: *JOP*  
(Ex. Quantifizierung)

# Wermutstropfen

...ist die Unentscheidbarkeit der *MOP*-Lösung im allgemeinen:

## Theorem (6.3.3, Unentscheidbarkeit)

(John B. Kam and Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. *Acta Informatica* 7:305-317, 1977.)

*Es gibt keinen Algorithmus A mit folgenden Eigenschaften:*

### 1. *Eingabe für A sind*

1.1 *Algorithmen zur Berechnung von Schnitt, Gleichheitstest und Anwendung von Funktionen auf Verbandselemente eines monotonen Datenflussanalyserahmens*

1.2 *eine durch  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  gegebene Instanz I dieses Rahmens*

### 2. *Ausgabe von A ist die MOP-Lösung von I.*

Deshalb betrachten wir eine **zweite Globalisierungsstrategie**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

344/863



# Kapitel 6.4

## *MaxFP-Ansatz*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

**6.4**

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Der *MaxFP*-Ansatz

Grundlegend für die *MaxFP*-Strategie:

Definition (6.4.1, *MaxFP*-Gleichungssystem)

Das *MaxFP*-Gleichungssystem ist gegeben durch:

$$\mathit{inf}(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{ \llbracket (m, n) \rrbracket (\mathit{inf}(m)) \mid m \in \mathit{pred}(n) \} & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

**6.4**

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

346/863

# Die *MaxFP*-Lösung

## Definition (6.4.2, *MaxFP*-Lösung)

Sei  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  die Spezifikation eines DFA-Problems. Dann ist für alle Knoten  $n \in N$  die *MaxFP-Lösung* definiert durch:

$$\text{MaxFP}_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}(n) =_{df} \text{inf}_{c_s}^*(n)$$

wobei  $\text{inf}_{c_s}^*$  die größte Lösung des *MaxFP*-Gleichungssystems 6.4.1 bezüglich  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  bezeichnet.

### Zentral:

- ▶ Die *MaxFP-Lösung* ist die (unter geeigneten Voraussetzungen, siehe Terminierungstheorem 6.6.2) **effektiv berechenbare** Lösung eines durch  $(\hat{C}, \llbracket \cdot \rrbracket, c_s)$  gegebenen intraprozeduralen DFA-Problems.

# Kapitel 6.5

## Koinzidenz- und Sicherheitstheorem

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

**6.5**

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Hauptresultate: Korrektheit u. Vollständigkeit

## Grundsätzlich:

- ▶ Zusammenhang von *MOP*-Lösung und *MaxFP*-Lösung

## Genauer:

- ▶ Korrektheit und Vollständigkeit der *MaxFP*-Lösung bzgl. der *MOP*-Lösung

## Im Detail:

- ▶ Korrektheitsfrage:

Gilt stets  $MaxFP_{(\hat{c}, [ ], c_s)} \sqsubseteq MOP_{(\hat{c}, [ ], c_s)}$  ?

- ▶ Vollständigkeitsfrage:

Gilt stets  $MaxFP_{(\hat{c}, [ ], c_s)} \sqsupseteq MOP_{(\hat{c}, [ ], c_s)}$  ?

## Theorem (6.5.1, Sicherheit)

Die MaxFP-Lösung ist eine *sichere* (konservative), d.h. untere Approximation der MOP-Lösung, d.h.,

$$\forall c_s \in \mathcal{C} \quad \forall n \in \mathbb{N}. \text{MaxFP}_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}(n) \sqsubseteq \text{MOP}_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}(n)$$

falls das DFA-Funktional  $\llbracket \cdot \rrbracket$  monoton ist.

# Vollständigkeit (und zugleich Korrektheit)

## Theorem (6.5.2, Koinzidenz)

Die MaxFP-Lösung *stimmt* mit der MOP-Lösung *überein*, d.h.,

$$\forall c_s \in \mathcal{C} \quad \forall n \in \mathbb{N}. \text{MaxFP}_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}(n) = \text{MOP}_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}(n)$$

*falls das DFA-Funktional  $\llbracket \cdot \rrbracket$  distributiv ist.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

351/863

# Bemerkung

Statt von

- ▶ Korrektheit
- ▶ Vollständigkeit

spricht man im Zusammenhang mit DFA traditionell von

- ▶ Sicherheit
- ▶ Koinzidenz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

**6.5**

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12



# Kapitel 6.6

## Generischer Fixpunktalgorithmus und Terminierungstheorem

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

**6.6**

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Hauptresultat: Effektivität

## Grundsätzlich:

- ▶ Zusammenhang von *MaxFP*-Lösung und Generischem Fixpunktalgorithmus 6.6.1

## Genauer:

- ▶ Effektivität des Generischen Fixpunktalgorithmus 6.6.1 für die *MaxFP*-Lösung

## Im Detail:

- ▶ Effektivitätsfrage:  
Terminiert der Generische Fixpunktalgorithmus 6.6.1 stets mit der  $MaxFP_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}$ -Lösung?

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

**6.6**

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Generischer Fixpunktalgorithmus 6.6.1 (1)

**Eingabe:** (1) Ein Flussgraph  $G = (N, E, s, e)$ , (2) ein DFA-Problem gegeben durch eine DFA-Spezifikation mit (lokaler) abstrakter Semantik bestehend aus einem DFA-Verband  $\hat{C}$ , einem DFA-Funktional  $\llbracket \cdot \rrbracket : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$ , und (3) einer Anfangsinformation  $c_s \in \mathcal{C}$ .

**Ausgabe:** Unter den Voraussetzungen des Terminierungstheorems 6.6.2 die  $MaxFP_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}$ -Lösung. Abhängig von den Eigenschaften des DFA-Funktionals gilt dann:

- (i)  $\llbracket \cdot \rrbracket$  ist **distributiv**: Variable *inf* enthält für jeden Knoten die stärkste Nachbedingung bezüglich der Anfangsinformation  $c_s$ .
- (ii)  $\llbracket \cdot \rrbracket$  ist **monoton**: Variable *inf* enthält für jeden Knoten eine sichere (d.h. untere) Approximation der stärksten Nachbedingung bezüglich der Anfangsinformation  $c_s$ .

# Generischer Fixpunktalgorithmus 6.6.1 (2)

## Bemerkung:

- ▶ Die **stärkste Nachbedingung** ist durch die  $MOP_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}$ -Lösung gegeben.
- ▶ Die Variable *workset* steuert den iterativen Prozess. Ihre Elemente sind Knoten aus  $G$ , deren Annotation jüngst aktualisiert worden ist, was möglicherweise zu einer verbandsmäßig kleineren Annotation an ihren Nachfolgerknoten führt.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

**6.6**

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

356/863

# Generischer Fixpunktalgorithmus 6.6.1 (3)

( Prolog: Initialisierung von *inf* und *workset* )

FORALL  $n \in N \setminus \{s\}$  DO  $inf[n] := \top$  OD;

$inf[s] := c_s$ ;

$workset := N$ ;

( Hauptprozess: Iterative Fixpunktberechnung )

WHILE  $workset \neq \emptyset$  DO

    CHOOSE  $m \in workset$ ;

$workset := workset \setminus \{m\}$ ;

    ( Aktualisiere die Nachfolgerumgebung von Knoten  $m$  )

    FORALL  $n \in succ(m)$  DO

$meet := \llbracket (m, n) \rrbracket (inf[m]) \sqcap inf[n]$ ;

        IF  $inf[n] \sqsupset meet$

            THEN

$inf[n] := meet$ ;

$workset := workset \cup \{n\}$

    FI

OD ES00HC OD.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

**6.6**

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

357/863

## Theorem (6.6.2, Terminierung)

Der *Generische Fixpunktalgorithmus 6.6.1* terminiert mit der  $\text{MaxFP}_{(\hat{C}, \llbracket \cdot \rrbracket, c_s)}$ -Lösung, falls

- das *DFA-Funktional*  $\llbracket \cdot \rrbracket$  *monoton* ist
- der *DFA-Verband*  $\hat{C}$  die *absteigende Kettenbedingung* erfüllt.

# Bemerkung

Der [Generische Fixpunktalgorithmus 6.6.1](#) ist formuliert für

- ▶ allquantifizierte (“distributive”) Vorwärtsprobleme

Die anderen [drei DFA-Problemvarietäten](#)

- ▶ existenziell quantifizierter (“additiver”) Probleme
- ▶ Rückwärtsprobleme

können mithilfe des [Generischen Fixpunktalgorithmus 6.6.1](#) gelöst werden, indem

- ▶ der Verband (durch Vertauschen von  $\sqsubseteq$  mit  $\supseteq$ )
- ▶ der Flussgraph (durch [Umdrehen](#) aller Kanten)

auf “[den Kopf gestellt](#)” wird.

# Kapitel 6.7

## Zusammenfassung und Überblick

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

**6.7**

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

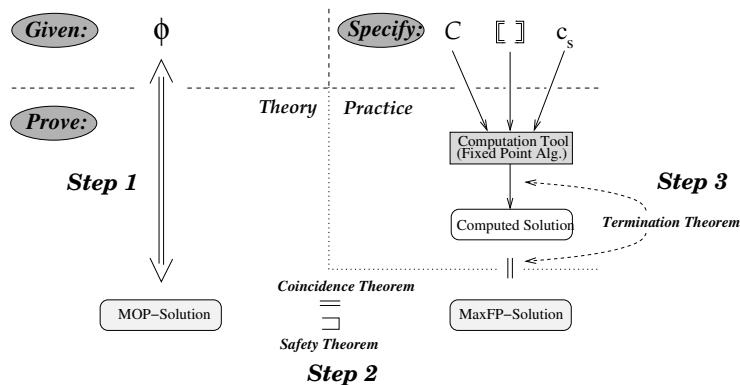
Kap. 11

Kap. 12



# Intraprozedurale DFA im Überblick (1)

Ein Bild sagt mehr als 1000 Worte:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

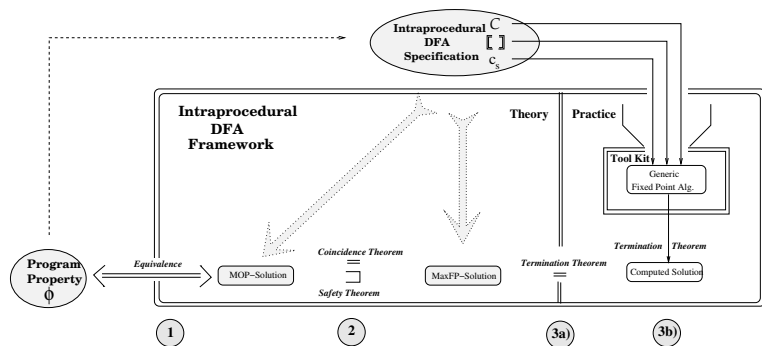
Kap. 10

Kap. 11

Kap. 12

# Intraprozedurale DFA im Überblick (2)

Fokussiert auf die Rahmen-/Werkzeugkistensicht:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Kapitel 6.8

Beispiele: Verfügbare Ausdrücke, einfache  
Konstanten

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

**6.8**

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Zwei prototypische DFA-Probleme

- ▶ **Verfügbare Ausdrücke**  
     $\rightsquigarrow$  kanonisches Bsp. eines **distributiven** DFA-Problems
- ▶ **Einfache Konstanten**  
     $\rightsquigarrow$  kanonisches Bsp. eines **monotonen** DFA-Problems

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

**6.8**

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Kapitel 6.6.1

## Verfügbare Ausdrücke

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

**6.8.1**

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Verfügbare Ausdrücke (1)

...ein typisches distributives DFA-Problem.

Informell:

- ▶ Ein Ausdruck  $t$  ist **verfügbar** an einem Knoten  $n$  eines Flussgraphen, wenn  $t$  auf allen Pfaden  $p$  von  $s$  nach  $n$  berechnet wird und nach der letzten Berechnung von  $t$  auf  $p$  kein Operand von  $t$  mehr verändert wird (d.h. nicht mehr linksseitig in einer Zuweisung vorkommt).

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

366/863

# Verfügbare Ausdrücke (2)

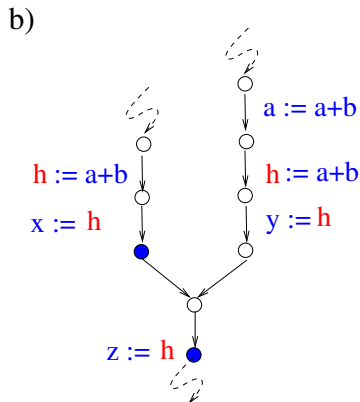
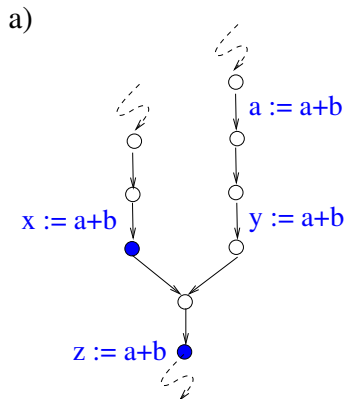
## Bemerkung:

- ▶  $t$  wird oft als **Kandidatenausdruck** bezeichnet.
- ▶ Ist  $t$  verfügbar an einem Knoten  $n$ , so kann eine Wiederberechnung an  $n$  durch Rückgriff auf seinen zuvor berechneten (und zu diesem Zweck gespeicherten) Wert vermieden werden; dadurch entsteht bestimmter Berechnungsaufwand zur Laufzeit überhaupt nicht, er wird vermieden:  
**Performanzverbesserung**  $\rightsquigarrow$  **Optimierung**

# Veranschaulichung

a) Vor der Optimierung

b) Nach der Optimierung



Die Optimierung wird bezeichnet als

- Elimination total redundanter Berechnungen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

368/863



# Spezifikation als DFA-Problem

- ▶ (Lokale) abstrakte Semantik für verfügbare Ausdrücke:

1. DFA-Verband:

$$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top) =_{df} (\mathbb{B}, \wedge, \vee, \leq, \mathbf{falsch}, \mathbf{wahr})$$

2. DFA-Funktional:  $\llbracket \cdot \rrbracket_{av} : E \rightarrow (\mathbb{B} \rightarrow \mathbb{B})$  definiert durch

$$\forall e \in E. \llbracket e \rrbracket_{av} =_{df} \begin{cases} Cst_{\mathbf{wahr}} & \text{falls } Comp_e \wedge Transp_e \\ Id_{\mathbb{B}} & \text{falls } \neg Comp_e \wedge Transp_e \\ Cst_{\mathbf{falsch}} & \text{sonst} \end{cases}$$

- ▶ Anfangsinformation:  $b_s \in \mathbb{B}$

# Bezeichnungen (1)

Dabei bezeichnen:

- ▶  $\hat{\text{IB}} =_{df} (\text{IB}, \wedge, \vee, <, \mathbf{falsch}, \mathbf{wahr})$ : Verband der Wahrheitswerte mit  $\perp = \mathbf{falsch} < \mathbf{wahr} = \top$  und dem logischen “und” und “oder” als Schnitt- bzw. Vereinigungsoperation  $\sqcap$  und  $\sqcup$ .
- ▶  $Cst_{\mathbf{wahr}}, Cst_{\mathbf{falsch}}$ : die konstanten Funktionen “wahr” und “falsch” auf IB
- ▶  $Id_{\text{IB}}$ : die Identität auf IB.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

## Bezeichnungen (2)

...sowie relativ zu einem fest gewählten **Kandidatenausdruck**  $t$ :

- ▶  $Comp_e$ :  $t$  wird von der Anweisung an Kante  $e$  **berechnet** (d.h.  $t$  kommt rechtsseitig als (Teil-) Ausdruck vor)
- ▶  $Transp_e$ : kein Operand von  $t$  erhält durch die Anweisung an Kante  $e$  einen neuen Wert (d.h. kein Operand von  $t$  kommt linksseitig vor:  $e$  ist **transparent** für  $t$ )

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

371/863

# Hauptergebnisse

## Lemma (6.8.1.1, Distributivität u. Kettenbedingung)

1.  $\llbracket \cdot \rrbracket_{av}$  ist distributiv.
2.  $\hat{I}\hat{B}$  erfüllt die absteigende Kettenbedingung.

## Korollar (6.8.1.2, Koinzidenz und Terminierung)

*Für das DFA-Problem verfügbarer Ausdrücke*

1. stimmen die MOP-Lösung und die MaxFP-Lösung überein, d.h. die MaxFP-Lösung ist korrekt und vollständig bezgl. der MOP-Lösung.
2. terminiert der Generische Fixpunktalgorithmus 6.6.1 mit der MaxFP-Lösung.

# Kapitel 6.6.2

## Einfache Konstanten

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

**6.8.2**

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Einfache Konstanten

...ein typisches monotones (nicht distributives) DFA-Problem.

## Informell:

- ▶ Ein Ausdruck  $t$  ist **einfache Konstante** an einem Knoten  $n$  eines Flussgraphen, wenn jeder Operand von  $t$  auf allen Pfaden  $p$  von  $s$  nach  $n$  vom selbem konstanten Wert ist.

## Bemerkung:

- ▶ Ist  $t$  eine einfache Konstante am Knoten  $n$ , so kann der Wert von  $t$  schon zur Übersetzungszeit berechnet werden und somit Berechnungsaufwand von der Laufzeit in die Übersetzungszeit eines Programms verschoben werden:  
**Performanzverbesserung  $\rightsquigarrow$  Optimierung**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

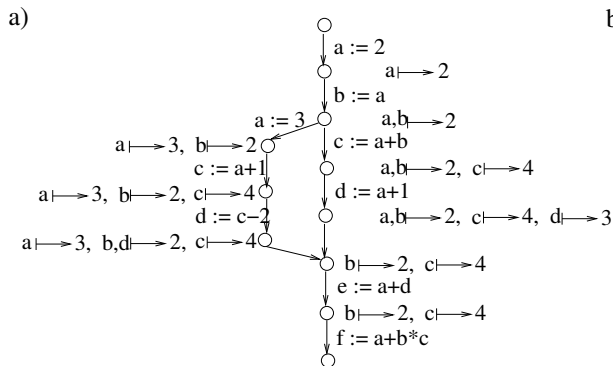
Kap. 11

Kap. 12

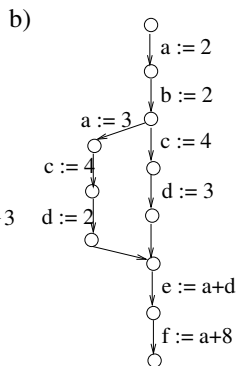
374/863

# Veranschaulichung

## a) Vor der Optimierung



## b) Nach der Optimierung



Die Optimierung wird bezeichnet als

- Konstantenausbreitung und Faltung

# Spezifikation als DFA-Problem

► (Lokale) abstrakte Semantik für einfache Konstanten:

1. DFA-Verband:

$$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top) =_{df} (\Sigma, \sqcap, \sqcup, \sqsubseteq, \sigma_{\perp}, \sigma_{\top})$$

2. DFA-Funktional:

$\llbracket \_ \rrbracket_{sc} : E \rightarrow (\Sigma \rightarrow \Sigma)$  definiert durch

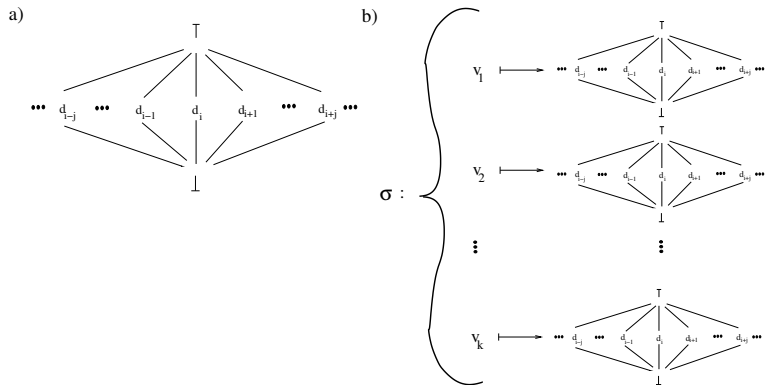
$$\forall e \in E. \llbracket e \rrbracket_{sc} =_{df} \theta_e$$

► Anfangsinformation:  $\sigma_s \in \Sigma$



# DFA-Verband für einfache Konstanten

Der "kanonische" Verband  $f$ . Konstantenausbreitung/Faltung:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Zusätzlich nötige Begriffe und Definitionen

...um die Definition der lokalen abstrakten Semantik für das DFA-Problem einfacher Konstanten abzuschließen:

- ▶ Termsyntax
- ▶ Interpretation
- ▶ Zustand
- ▶ Termsemantik
- ▶ Zustandstransformationsfunktion

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Termsyntax (1)

## Variablen, Konstanten, Operatoren

Sei

- ▶ **V** eine Menge von Variablen
- ▶ **Op** eine Menge von  $n$ -stelligen Operatoren,  $n \geq 0$ , sowie **C**  $\subseteq$  **Op** die Menge der 0-stelligen Operatoren aus **Op**, die sog. Konstanten.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

379/863

# Termsyntax (2)

## Zusammengesetzte Terme

### Definition (6.8.2.1, Termsyntax)

Wir legen fest:

1. Jede Variable  $v \in \mathbf{V}$  und jede Konstante  $c \in \mathbf{C}$  ist ein **Term**.
2. Ist  $op \in \mathbf{Op}$  ein  $n$ -stelliger Operator,  $n \geq 1$ , und sind  $t_1, \dots, t_n$  Terme, dann ist auch  $op(t_1, \dots, t_n)$  ein **Term**.
3. Es gibt keine weiteren **Terme** außer den nach den obigen beiden Regeln konstruierbaren.

Die Menge aller Terme bezeichnen wir mit  $\mathbf{T}$ .

## Definition (6.8.2.2, Interpretation)

Sei  $\mathbf{D}'$  ein geeigneter Datenbereich (z.B. die Menge der ganzen Zahlen), seien  $\perp$  und  $\top$  zwei ausgezeichnete Elemente mit  $\perp, \top \notin \mathbf{D}'$  und sei  $\mathbf{D} =_{df} \mathbf{D}' \cup \{\perp, \top\}$ .

Eine **Interpretation** über  $\mathbf{T}$  und  $\mathbf{D}$  ist ein Paar  $I \equiv (\mathbf{D}, I_0)$ , wobei

- ▶  $I_0$  eine Funktion ist, die mit jedem 0-stelligen Operator  $c \in \mathbf{Op}$  ein Datum  $I_0(c) \in \mathbf{D}'$  und mit jedem  $n$ -stelligen Operator  $op \in \mathbf{Op}$ ,  $n \geq 1$ , eine totale Funktion  $I_0(op) : \mathbf{D}^n \rightarrow \mathbf{D}$  assoziiert, die als **strikt** angenommen wird (d.h.  $I_0(op)(d_1, \dots, d_n) = \perp$ , wann immer es ein  $j \in \{1, \dots, n\}$  gibt mit  $d_j = \perp$ )

# Zustand, Zustandsmenge, undef. Zustand

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

## Definition (6.8.2.3, Zustand, Zustandsmenge)

- ▶ Ein **Zustand**  $\sigma : \mathbf{V} \rightarrow \mathbf{D}$  ist eine Abbildung von der Menge der Programmvariablen  $\mathbf{V}$  auf den Datenbereich  $\mathbf{D}$ .

- ▶ Mit

$$\Sigma =_{df} \{ \sigma \mid \sigma : \mathbf{V} \rightarrow \mathbf{D} \}$$

bezeichnen wir die **Menge aller Zustände**.

- ▶ Mit  $\sigma_{\perp}$  bezeichnen wir den **(total) undefinierten** Zustand aus  $\Sigma$ , für den gilt:  $\forall v \in \mathbf{V}. \sigma_{\perp}(v) = \perp$

## Definition (6.8.2.4, Termsemantik)

Die **Semantik** von Termen  $t \in \mathbf{T}$  ist durch die induktiv definierte **Evaluationsfunktion**  $\mathcal{E}$  gegeben:

$\mathcal{E} : \mathbf{T} \rightarrow (\Sigma \rightarrow \mathbf{D})$  definiert durch

$$\forall t \in \mathbf{T} \forall \sigma \in \Sigma. \mathcal{E}(t)(\sigma) \stackrel{df}{=} \begin{cases} \sigma(x) & \text{falls } t \equiv x \in \mathbf{V} \\ l_0(c) & \text{falls } t \equiv c \in \mathbf{C} \\ l_0(op)(\mathcal{E}(t_1)(\sigma), \dots, \mathcal{E}(t_r)(\sigma)) & \text{falls } t \equiv op(t_1, \dots, t_r) \end{cases}$$

# Zustandstransformationsfunktion

Definition (6.2.8.5, Zustandstransformationsfkt.)

Die Zustandstransformationsfunktion  $\theta_\iota$

$$\theta_\iota : \Sigma \rightarrow \Sigma, \quad \iota \equiv x := t$$

ist definiert durch:

$$\forall \sigma \in \Sigma \forall y \in \mathbf{V}. \theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y \equiv x \\ \sigma(y) & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

384/863



# Hauptergebnisse

## Lemma (6.8.2.6, Monotonie und Kettenbedingung)

1.  $\llbracket \cdot \rrbracket_{sc}$  ist monoton.
2.  $\hat{\Sigma}$  erfüllt die absteigende Kettenbedingung.




**Beachte:** Distributivität von  $\llbracket \cdot \rrbracket_{sc}$  ist i.a. verletzt!

## Korollar (6.8.2.7, Sicherheit und Terminierung)




*Für das DFA-Problem einfacher Konstanten*

1. stimmen die MOP-Lösung und die MaxFP-Lösung i.a. nicht überein; die MaxFP-Lösung ist aber stets eine sichere, d.h. untere Approximation der MOP-Lösung.
2. terminiert der Generische Fixpunktalgorithmus 6.6.1 mit der MaxFP-Lösung.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (1)

-  Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. Addison-Wesley, 2nd edition, 2007. (Chapter 1, Introduction; Chapter 9.2, Introduction to Data-Flow Analysis; Chapter 9.3, Foundations of Data-Flow Analysis)
-  Randy Allen, Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2002. (Chapter 4.4, Data Flow Analysis)
-  Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004. (Chapter 1, Overview of Compilation; Chapter 8, Introduction to Code Optimization; Chapter 9, Data Flow Analysis)

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (2)

-  Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.
-  John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.
-  Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8




Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (3)

-  Jens Knoop. *From DFA-frameworks to DFA-generators: A unifying multiparadigm approach*. In Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), Springer-V., LNCS 1579, 360-374, 1999.
-  Jens Knoop, Bernhard Steffen. *The Interprocedural Coincidence Theorem*. In Proceedings of the 4th International Conference on Compiler Construction (CC'92), Springer-V., LNCS 641, 125-140, 1992.
-  Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Efficient and Optimal Bitvector Analyses for Parallel Programs*. ACM Transactions on Programming Languages and Systems 18(3):268-299, 1996.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8





Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (4)

-  Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009. (Chapter 7, What can one tell about a Program without its Execution: Static Analysis)
-  Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 20:121-163, 1990.
-  Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.
-  Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997. (Chapter 1, Introduction to Advanced Topics; Chapter 4, Intermediate Representations; Chapter 7, Control-Flow Analysis; Chapter 8, Data Flow Analysis; Chapter 11, Introduction to Optimization; Chapter 12, Early Optimizations)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8




Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 6 (5)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 5, Static Program Analysis)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 7, Program Analysis; Chapter 8, More on Program Analysis; Appendix B, Implementation of Program Analysis)
-  Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2nd edition, Springer-V., 2005. (Chapter 1, Introduction; Chapter 2, Data Flow Analysis; Chapter 6, Algorithms)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

6.1

6.2

6.3

6.4

6.5

6.6

6.7

6.8

6.8.1

6.8.2

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

# Kapitel 7

## Programmverifikation vs. Programmanalyse

# Programmverifikation vs. -analyse: SPC-Sicht

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

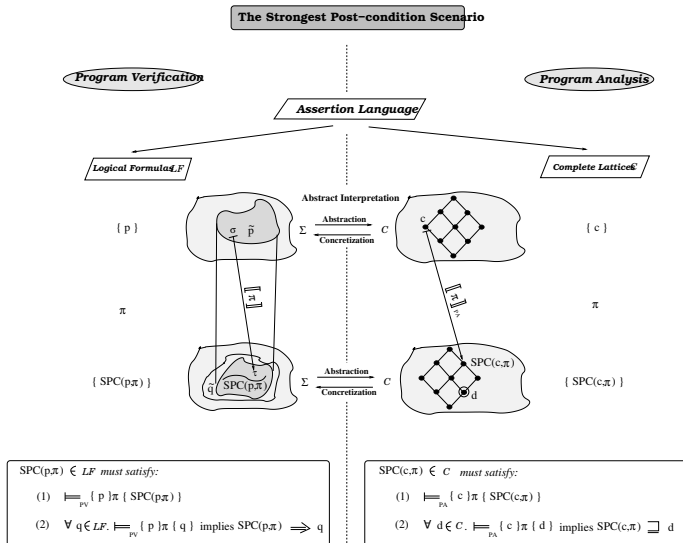
Kap. 13

Kap. 14

Kap. 15

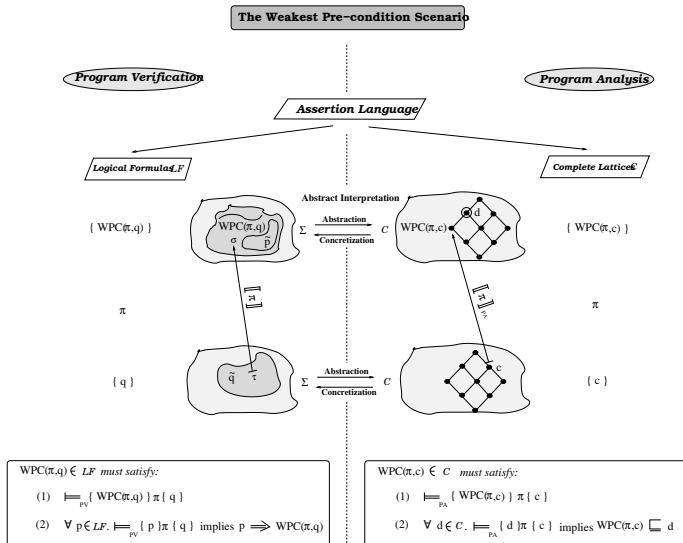
Kap. 16

Kap. 17





# Programmverifikation vs. -analyse: WPC-Sicht



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

**Kap. 7**

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Kapitel 8

## Reverse Datenflussanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

**Kap. 8**

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Kapitel 8.1

## Grundlagen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

**8.1**

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Vorbereitung (1)

Erweiterung des DFA-Verbands  $\hat{\mathcal{C}}$  um ein *failure*-Element:

Sei  $(\hat{\mathcal{C}}, \llbracket \cdot \rrbracket, c_s)$  eine DFA-Spezifikation, sei *failure* ein neues Element nicht aus  $\mathcal{C}$ . Wir erweitern  $\hat{\mathcal{C}}$  (bzw.  $\mathcal{C}$ ) um das Element *failure* zum erweiterten Verband

$$\hat{\mathcal{C}}_f =_{df} (\mathcal{C}_f =_{df} \mathcal{C} \dot{\cup} \{\text{failure}\}, \sqsubseteq_f, \sqcap_f, \sqcup_f, \perp, \text{failure})$$

so dass *failure* größtes Element in  $\hat{\mathcal{C}}_f$  ist, d.h.

1.  $\forall c \in \mathcal{C}_f. c \sqsubseteq_f \text{failure}$
2.  $\forall c, c' \in \mathcal{C}. c \sqsubseteq_f c' \text{ gdw } c \sqsubseteq c'$

Beachte:

- ▶ Durch die Festlegung von  $\sqsubseteq_f$  sind auch  $\sqcap_f$  und  $\sqcup_f$  eindeutig festgelegt.
- ▶ Das neue Element *failure* repräsentiert eine unerfüllbare DFA-Information.

# Vorbereitung (2)

Ausdehnung des DFA-Funktional  $\llbracket \cdot \rrbracket$  auf  $\mathcal{C}$  zum DFA-Funktional  $\llbracket \cdot \rrbracket_f$  auf  $\mathcal{C}_f$ :

Hierzu legen wir fest:

$$\forall c \in \mathcal{C}_f \forall e \in E. \llbracket e \rrbracket_f(c) =_{df} \begin{cases} \llbracket e \rrbracket(c) & \text{falls } c \neq \text{failure} \\ \text{failure} & \text{sonst} \end{cases}$$

## Lemma (8.1.1)

- ▶  $\llbracket \cdot \rrbracket_f$  ist monoton gdw  $\llbracket \cdot \rrbracket$  ist monoton.
- ▶  $\llbracket \cdot \rrbracket_f$  ist distributiv gdw  $\llbracket \cdot \rrbracket$  ist distributiv.

# Reverse abstrakte Semantik

Sei  $\llbracket \cdot \rrbracket : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$  eine (lokale) abstrakte Semantik auf  $\mathcal{C}$ .  
Dann ist die durch die Erweiterung  $\llbracket \cdot \rrbracket_f$  von  $\llbracket \cdot \rrbracket$  definierte reverse (lokale) abstrakte Semantik wie folgt festgelegt:

## Definition (8.1.2, Reverse abstrakte Semantik)

Die reverse abstrakte Semantik zu einer abstrakten Semantik  $(\hat{\mathcal{C}}, \llbracket \cdot \rrbracket)$  ist gegeben durch:

1. (Reverser) DFA-Verband  $\hat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \perp, failure)$
2. Reverses DFA-Funktional

$\llbracket \cdot \rrbracket_R : E \rightarrow (\mathcal{C}_f \rightarrow \mathcal{C}_f)$  definiert durch

$$\forall e \in E \forall c \in \mathcal{C}. \llbracket e \rrbracket_R(c) =_{df} \sqcap \{ c' \mid \llbracket e \rrbracket_f(c') \sqsupseteq c \}$$

# DFA vs. rDFA

## Intuitiv:

- ▶ DFA zielt für jede Programmstelle auf die Berechnung des **stärkst möglichen** Datenflussfakts (relativ zu einer gegebenen Startinformation).
- ▶ rDFA zielt für jede Programmstelle auf die Berechnung eines **schwächst möglichen** Datenflussfakts, so dass ein 'gewünschter' Datenflussfakt an einer bestimmten Programmstelle gültig ist.
- ▶ Reverses Gegenstück der **meet-over-all-paths** Globalisierung einer abstrakten Semantik ist daher die **reverse join-over-all-paths** Globalisierung der zugehörigen reversen abstrakten Semantik.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

399/863

# Grapherweiterung um Anfrageknoten

Der Übergang von DFA zu rDFA ist geradlinig, die Globalisierung der lokalen reversen abstrakten Semantik erfordert aber die folgende Grapherweiterung:

- ▶ Sei  $G' = (N', E', \mathbf{s}', \mathbf{e}')$  ein Flussgraph und  $q \in N'$  der interessierende Programmpunkt, der sog. **Anfrageknoten (query node)**.
- ▶ Ist  $q$  von  $\mathbf{s}'$  verschieden, dann wird  $G'$  durch eine Kopie  $\mathbf{q}$  von  $q$  zu  $G = (N, E, \mathbf{s}, \mathbf{e})$  erweitert, wobei der neue Knoten  $\mathbf{q}$  dieselben Vorgänger wie  $q$  besitzt, aber keine Nachfolger.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

400/863



# Beobachtung

Es gilt:

- ▶ Die Hinzunahme von  $\mathbf{q}$  hat keinen Einfluss auf die *MOP*-Lösung irgendeines der ursprünglichen Knoten von  $G$ .
- ▶ Die *MOP*-Lösungen von  $\mathbf{q}$  und  $q$  stimmen überein.

Die folgenden drei Lemmata fassen diese Beobachtungen zusammen und formalisieren sie.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

401/863

# Zusammenhang von $\llbracket \cdot \rrbracket_f$ und $\llbracket \cdot \rrbracket_R$ (1)

## Lemma (8.1.3)

Sei  $\llbracket \cdot \rrbracket_f$  ein (erweitertes) DFA-Funktional. Dann gilt für jede Kante  $e \in E$ :

1.  $\llbracket e \rrbracket_R$  ist wohldefiniert und monoton.
2.  $\llbracket e \rrbracket_R$  ist additiv, falls  $\llbracket \cdot \rrbracket_f$  distributiv ist.

# Zusammenhang von $\llbracket \cdot \rrbracket_f$ und $\llbracket \cdot \rrbracket_R$ (2)

## Lemma (8.1.4)

Sei  $\llbracket \cdot \rrbracket_f$  ein (erweitertes) DFA-Funktional. Dann gilt für jede Kante  $e \in E$ :

1.  $\llbracket e \rrbracket_R \circ \llbracket e \rrbracket_f \sqsubseteq Id_{C_f}$ , falls  $\llbracket e \rrbracket$  monoton ist.
2.  $\llbracket e \rrbracket_f \circ \llbracket e \rrbracket_R \sqsupseteq Id_{C_f}$ , falls  $\llbracket e \rrbracket$  distributiv ist.

Sprechweise in der Theorie “**Abstrakter Interpretation**”:

- ▶  $\llbracket e \rrbracket_f$  und  $\llbracket e \rrbracket_R$  bilden eine **Galois-Verbindung**.

# Zusammenhang von $\llbracket \cdot \rrbracket_f$ und $\llbracket \cdot \rrbracket_R$ (3)

## Lemma (8.1.5)

1.  $\forall n \in N' \cap N. \mathbf{P}_{G'}[\mathbf{s}, n] = \mathbf{P}_G[\mathbf{s}, n]$
2.  $\forall q \in N' \setminus \{\mathbf{s}\}. \mathbf{P}_{G'}[\mathbf{s}, q] = \mathbf{P}_G[\mathbf{s}, \mathbf{q}]$
3.  $\forall c_s \in \mathcal{C} \forall n \in N' \cap N. MOP_{(G', c_s)}(n) = MOP_{(G, c_s)}(n)$
4.  $MOP_{(G, c_s)}(q) = MOP_{(G, c_s)}(\mathbf{q})$

wobei  $\mathbf{q}$  (*“query node”*) Kopie von  $q$  ist mit  $\text{pred}(\mathbf{q}) = \text{pred}(q)$  und  $\text{succ}(\mathbf{q}) = \emptyset$ .

# rDFA-Spezifikation und rDFA-Problem

Mit den vorherigen Festlegungen und Beobachtungen ist es sinnvoll festzulegen:

## Definition (8.1.6, rDFA-Spezifikation)

Eine **rDFA-Spezifikation** zu einer abstrakten Semantik  $(\hat{\mathcal{C}}, \llbracket \cdot \rrbracket)$  ist ein Tripel  $(\hat{\mathcal{C}}_f, \llbracket \cdot \rrbracket_R, c_q)$  gegeben durch

- ▶ eine **reverse (lokale) abstrakte Semantik** bestehend aus
  1. einem **DFA-Verband**  $\hat{\mathcal{C}}_f =_{df} (\mathcal{C}_f, \sqsubseteq_f, \sqcap_f, \sqcup_f, \perp, failure)$
  2. einem **reversen DFA-Funktional**  $\llbracket \cdot \rrbracket_R : E \rightarrow (\mathcal{C}_f \rightarrow \mathcal{C}_f)$
- ▶ einer **DFA-Frage**:  $c_q \in \mathcal{C}$

## Definition (8.1.7, rDFA-Problem)

Eine rDFA-Spezifikation legt ein **rDFA-Problem** fest.

# Kapitel 8.2

## *rJOP*-Ansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

**8.2**

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Ausdehnung von $\llbracket \cdot \rrbracket_R$ auf Pfade

Grundlegend für die *rJOP*-Strategie:

## Definition (8.2.1, Pfadausdehnung von $\llbracket \cdot \rrbracket_R$ )

Die Ausdehnung einer reversen (lokalen) abstrakten Semantik auf Pfade  $p = \langle e_1, \dots, e_{q-1}, e_q \rangle$  ist definiert durch:

$$\llbracket p \rrbracket_R \stackrel{df}{=} \begin{cases} Id_C & \text{falls } \lambda_p < 1 \\ \llbracket \langle e_1, \dots, e_{q-1} \rangle \rrbracket_R \circ \llbracket e_q \rrbracket_R & \text{sonst} \end{cases}$$

**Beachte:** Die obige Ausdehnung bedeutet einen Rückwärtsdurchlauf von Pfad  $p$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

407/863

# Der $rJOP$ -Ansatz

## Definition (8.3.2, $rJOP$ -Lösung)

Sei  $(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)$  die Spezifikation eines rDFA-Problems. Dann ist für alle Knoten  $n \in N$  die  $rJOP$ -Lösung definiert durch:

$$rJOP_{(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)}(n) =_{df} \sqcup \{ \llbracket p \rrbracket_R(c_q) \mid p \in \mathbf{P}[n, \mathbf{q}] \}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

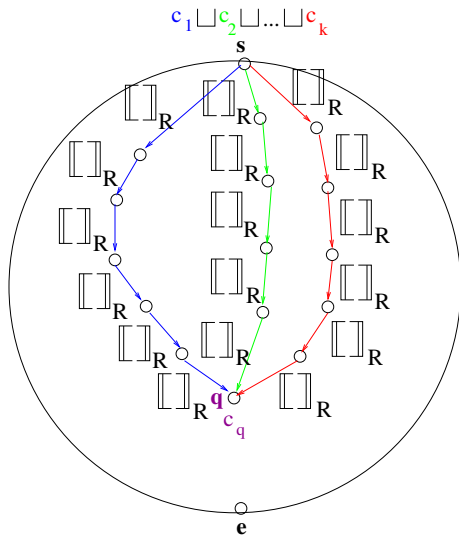
Kap. 12

Kap. 13

408/863



# Veranschaulichung



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

**8.2**

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

409/863

# Kapitel 8.3

## *rMinFP*-Ansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

**8.3**

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Der *rMinFP*-Ansatz

Grundlegend für die *rMinFP*-Strategie:

Definition (8.3.1, *rMinFP*-Gleichungssystem)

Das *rMinFP*-Gleichungssystem ist gegeben durch:

$reqInf(n) =$

$$\begin{cases} c_q & \text{falls } n = \mathbf{q} \\ \sqcup \{ \llbracket (n, m) \rrbracket_R(reqInf(m)) \mid m \in succ(n) \} & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

411/863

# Die *rMinFP*-Lösung

## Definition (8.3.2, *rMinFP*-Lösung)

Sei  $(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)$  die Spezifikation eines rDFA-Problems. Dann ist für alle Knoten  $n \in N$  die *rMinFP-Lösung* definiert durch:

$$rMinFP_{(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)}(n) =_{df} reqInf_{c_q}^*(n)$$

wobei  $reqInf_{c_q}^*$  die kleinste Lösung des *rMinFP*-Gleichungssystems 8.3.1 bzgl.  $(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)$  bezeichnet.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

**8.3**

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

412/863

# Kapitel 8.4

## Reverses Koinzidenz- und Sicherheitstheorem

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

**8.4**

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Hauptresultate: Korrektheit u. Vollständigkeit

## Grundsätzlich:

- ▶ Zusammenhang von  $rJOP$ -Lösung und  $rMinFP$ -Lösung

## Genauer:

- ▶ Korrektheit und Vollständigkeit der  $rMinFP$ -Lösung bzgl. der  $rJOP$ -Lösung

## Im Detail:

- ▶ Korrektheitsfrage:

Gilt stets  $rMinFP_{(\hat{c}_f, \llbracket \rrbracket_{R, C_q})} \supseteq rJOP_{(\hat{c}_f, \llbracket \rrbracket_{R, C_q})}$  ?

- ▶ Vollständigkeitsfrage:

Gilt stets  $rMinFP_{(\hat{c}_f, \llbracket \rrbracket_{R, C_q})} \sqsubseteq rJOP_{(\hat{c}_f, \llbracket \rrbracket_{R, C_q})}$  ?

## Theorem (8.4.1, Reverse Sicherheit)

Die *rMinFP*-Lösung ist eine *sichere* (konservative), d.h. obere Approximation der *rJOP*-Lösung, d.h.,

$$\forall c_q \in \mathcal{C} \forall n \in \mathbb{N}. rMinFP_{(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)}(n) \supseteq rJOP_{(\hat{C}_f, \llbracket \cdot \rrbracket_R, c_q)}(n)$$

### Beachte:

- ▶ Die reversen Semantikfunktionen sind stets monoton (vgl. Lemma 8.1.3(1)); deshalb braucht diese Voraussetzung in Theorem 8.4.1 nicht explizit genannt zu werden.

# Vollständigkeit (und zugleich Korrektheit)

## Theorem (8.4.2, Reverse Koinzidenz)

Die *rMinFP*-Lösung *stimmt* mit der *rJOP*-Lösung *überein*, d.h.,

$$\forall c_q \in \mathcal{C} \quad \forall n \in \mathbb{N}. \text{rMinFP}_{(\hat{c}_f, [\ ]_R, c_q)}(n) = \text{rJOP}_{(\hat{c}_f, [\ ]_R, c_q)}(n)$$

falls das  $[\ ]_R$  induzierende DFA-Funktional  $[\ ]$  distributiv ist.

### Beachte:

- ▶ Distributivität von  $[\ ]$  (und deshalb auch  $[\ ]_f$ ) impliziert Additivität von  $[\ ]_R$  (vgl. Lemma 8.1.1 und 8.1.3(2)). Somit ist Additivität von  $[\ ]_R$  implizit in Theorem 8.4.2 gefordert, wird aber auf die Distributivitätseigenschaft des Ursprungproblems zurückgeführt.



# Kapitel 8.5

## Generischer Fixpunktalgorithmus und Reverses Terminierungstheorem

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

**8.5**

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Hauptresultat: Effektivität

## Grundsätzlich:

- ▶ Zusammenhang von  $rMinFP$ -Lösung und Generischem Fixpunktalgorithmus 8.5.1

## Genauer:

- ▶ Effektivität des Generischen Fixpunktalgorithmus 8.5.1 für die  $rMinFP$ -Lösung

## Im Detail:

- ▶ Effektivitätsfrage:  
Terminiert der Generische Fixpunktalgorithmus 8.5.1 stets mit der  $rMinFP_{(\hat{C}_f, \llbracket \cdot \rrbracket_{R^3, C_q})}$ -Lösung?

# Der generische $rMinFP$ -Alg. 8.5.1 (1)

**Eingabe:** (1) Ein Flussgraph  $G = (N, E, s, e)$ , (2) ein (Anfrage-) Programmpunkt  $q$ , (3) ein reverses DFA-Problem gegeben durch eine reverse (lokale) abstrakte Semantik bestehend aus einem erweiterten DFA-Verband  $\hat{\mathcal{C}}_f$  und einem reversen DFA-Funktional  $\llbracket \rrbracket_R : E \rightarrow (\mathcal{C}_f \rightarrow \mathcal{C}_f)$  induziert von einem DFA-Funktional  $\llbracket \rrbracket : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$ , und (4) eine Anfrageinformation  $c_q \in \mathcal{C}$  am Anfragepunkt  $q$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

## Der generische $rMinFP$ -Alg. 8.5.1 (2)

**Ausgabe:** Unter den Voraussetzungen des Terminierungstheorems 8.5.2 die  $rMinFP_{(\hat{c}_f, [\ ]_R, c_q)}$ -Lösung. Abhängig von den Eigenschaften des reversen DFA-Funktionalis gilt dann:

(i)  $[\ ]_R$  ist **additiv**: Variable  $reqInf[s]$  enthält die schwächste Vorbedingung für  $c_q$  an  $q$ , d.h. den kleinsten DFA-Fakt, der beim Programmaufruf gelten muss, um die Gültigkeit von  $c_q$  an  $q$  zu garantieren. Ist dies *failure*, so gilt  $c_q$  an  $q$  definitiv nicht.

(ii)  $[\ ]_R$  ist **monoton**: Variable  $reqInf[s]$  enthält eine obere Approximation der schwächsten Vorbedingung für  $c_q$  an  $q$ , d.h. eine obere Schranke für den DFA-Fakt, der beim Programmaufruf gelten muss, um die Gültigkeit von  $c_q$  an  $q$  zu garantieren. Ist dies *failure*, so schließt dies aufgrund der Überapproximation nicht aus, dass  $c_q$  an  $q$  gilt.

# Der generische $rMinFP$ -Alg. 8.5.1 (3)

## Bemerkung:

- ▶ Die **schwächste Vorbedingung** ist durch die  $rJOP_{(\hat{C}_f, \mathbb{I}_R, c_q)}$ -Lösung gegeben.
- ▶ Die Variable **workset** steuert den iterativen Prozess. Ihre Elemente sind Knoten aus dem erweiterten Flussgraphen  $G'$ , deren Annotation jüngst aktualisiert worden ist, was möglicherweise zu einer verbandsmäßig größeren Annotation an ihren Vorgängerknoten führt.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

**8.5**

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

421/863

# Der generische $rMinFP$ -Alg. 8.5.1 (4)

( Prolog: Initialisierung von  $reqInf$  und  $workset$  )

FORALL  $n \in N \setminus \{\mathbf{q}\}$  DO  $reqInf[n] := \perp$  OD;

$reqInf[\mathbf{q}] := c_q$ ;

$workset := \{\mathbf{q}\}$ ;

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

**8.5**

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Der generische $rMinFP$ -Alg. 8.5.1 (5)

(Hauptprozess: Iterative Fixpunktberechnung)

WHILE  $workset \neq \emptyset$  DO

    CHOOSE  $m \in workset$ ;

$workset := workset \setminus \{m\}$ ;

    (Aktualisierung d. Vorgängerumgebung von Knoten  $m$ )

    FORALL  $n \in pred(m)$  DO

$join := \llbracket (n, m) \rrbracket_R(reqInf[m]) \sqcup_f reqInf[n]$ ;

        IF  $reqInf[n] \sqsubset_f join$

            THEN

$reqInf[n] := join$ ;

$workset := workset \cup \{n\}$

    FI

OD

ESOOHC

OD.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

**8.5**

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

423/863

# Effektivität

Aufgrund der stets gegebenen Monotonie der reversen Semantikfunktionen (vgl. Lemma 8.1.2(1)) erhalten wir:

## Theorem (8.5.2, Reverse Terminierung)

Der *Generische Fixpunktalgorithmus 8.5.1* terminiert mit der  $r\text{MinFP}_{(\hat{C}_f, \llbracket \cdot \rrbracket_{R, Cq})}$ -Lösung, falls der DFA-Verband  $\hat{C}$  die aufsteigende Kettenbedingung erfüllt.

Beachte:

- ▶  $\hat{C}_f$  erfüllt die aufsteigende Kettenbedingung gdw  $\hat{C}$  die aufsteigende Kettenbedingung erfüllt. Wie bei Theorem 8.4.2 wird hier die entscheidende Voraussetzung auf die entsprechende Eigenschaft des Ursprungsproblems zurückgeführt.



# Kapitel 8.6

## Analyse und Verifikation: Analogien und Gemeinsamkeiten

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

**8.6**

8.7

8.8

Kap. 9

Kap. 10

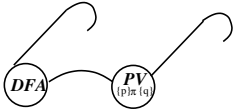
Kap. 11

Kap. 12

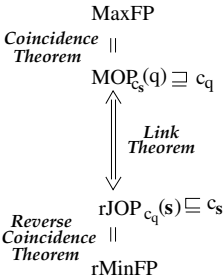
Kap. 13

# DFA/rDFA vs. Verifikation

- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- Kap. 5
- Kap. 6
- Kap. 7
- Kap. 8
  - 8.1
  - 8.2
  - 8.3
  - 8.4
  - 8.5
  - 8.6
  - 8.7
  - 8.8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13



## Data-flow Analysis



## Program Verification

$$\llbracket e \rrbracket \in \llbracket C \xrightarrow{\text{distributive}} C \rrbracket$$



$$\llbracket e \rrbracket_R(c) \stackrel{\text{def}}{=} \bigsqcap \{c' \mid \llbracket e \rrbracket(c') \sqsupseteq c\}$$

*Strongest Postcondition View*

$$\{p\} \pi \{?\}$$

$$\{?\} \pi \{q\}$$

*Weakest Precondition View*

# Zusammenhang DFA und rDFA

...für **distributive** DFA-Probleme.

## Theorem (8.6.1, Zusammenhangs-Theorem)

*Es gilt:*

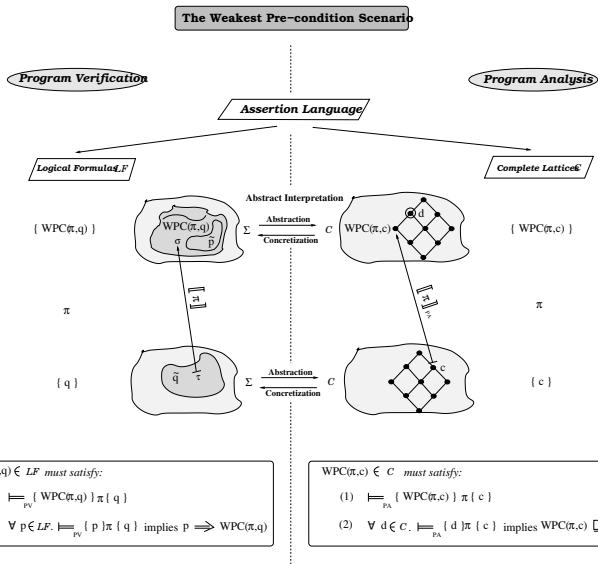
$$\text{MaxFP}_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}(n) = \text{MOP}_{(\hat{c}, \llbracket \cdot \rrbracket, c_s)}(n) \sqsupseteq c_q$$

$$\iff$$

$$\text{rMinFP}_{(\hat{c}_f, \llbracket \cdot \rrbracket_R, c_q)}(n) = \text{rJOP}_{(\hat{c}_f, \llbracket \cdot \rrbracket_R, c_q)}(n) \sqsubseteq c_s$$

*falls das DFA-Funktional  $\llbracket \cdot \rrbracket$  distributiv ist.*

# Insgesamt: WPC-Analogie Verifikation/(r)DFA



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

**8.6**

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

428/863

# Drei unterschiedliche Problemsichten (1)

The *specification problem*:

$\{?\} \pi \{q\}$

... the domain of *reverse DFA*

The *verification problem*:

$\{p\} \pi \{q\} ?$

... the domain of *demand-driven DFA*

The *implementation problem*:

$\{p\} \pi \{?\}$

... the domain of *exhaustive DFA*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

**8.6**

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

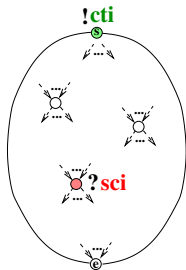
Kap. 12

Kap. 13

429/863

# Drei unterschiedliche Problemsichten (2)

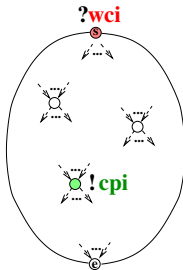
## Implementation Problem



**! Given:** Context Information **cti**

**? Sought:** Strongest Component Information **sci**

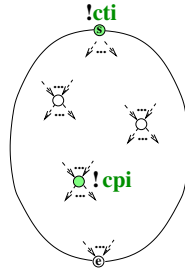
## Specification Problem



**! Given:** Component Information **cpi**

**? Sought:** Weakest Context Information **wci**

## Verification Problem



**! Given:** Context Information **cti**

Component Information **cpi**

**? Sought:** Validity of **cpi** with respect of **cti**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

**8.6**

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

430/863

# Kapitel 8.7

## Anwendungen reverser Datenflussanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

**8.7**

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Beispiele

## Anwendungen reverser DFA

- ▶ Einfache **Analysatoren** und **Optimierer**
- ▶ **“Hot Spot”** Programmanalyse und -optimierung
- ▶ **Debugger**
- ▶ ...

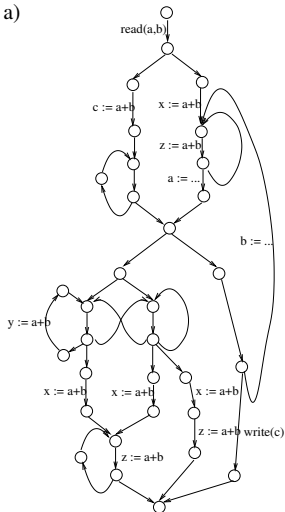
## Insbesondere

- ▶ **Anforderungsgetriebene Datenflussanalyse**  
(demand-driven data-flow analysis)



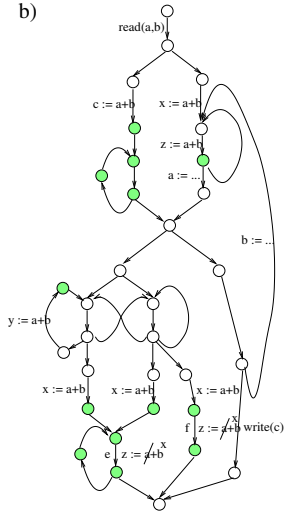
# Einfacher Analysator und Optimierer

a)



Flow graph

b)



Data-flow information

Program points

satisfying **availability** ●

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

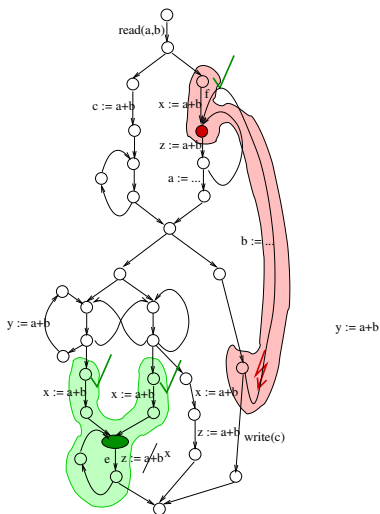
Kap. 10

Kap. 11



Kap. 12

Kap. 13

# "Hot Spot"-Analysator und Optimierer



## "Hot Spot" Optimizer

Program point  ✓  
satisfies **availability**  
while  ⚡ does not!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

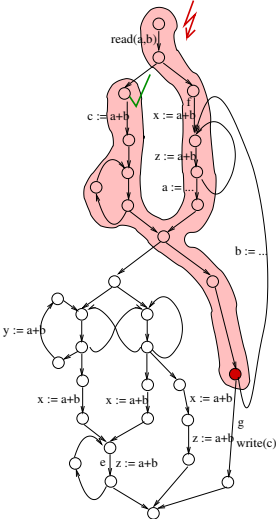
Kap. 11

Kap. 12

Kap. 13

434/863

# Debugger



## Debugger

Variable `c` is not initialized  
along some paths reaching  
program point ●

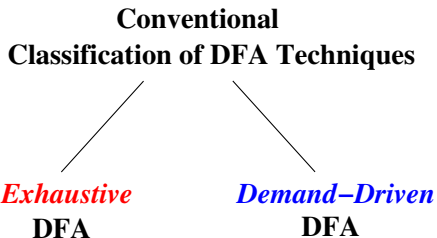
- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- Kap. 5
- Kap. 6
- Kap. 7
- Kap. 8
- 8.1
- 8.2
- 8.3
- 8.4
- 8.5
- 8.6
- 8.7**
- 8.8

- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13

# Erschöpfende vs. anforderungsgetriebene DFA

Erschöpfende (xDFA) vs.

anforderungsgetriebene DFA (ddDFA):



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

**8.7**

8.8

Kap. 9

Kap. 10

Kap. 11

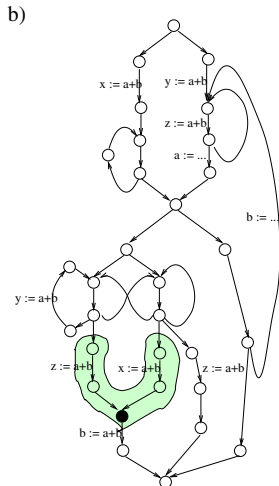
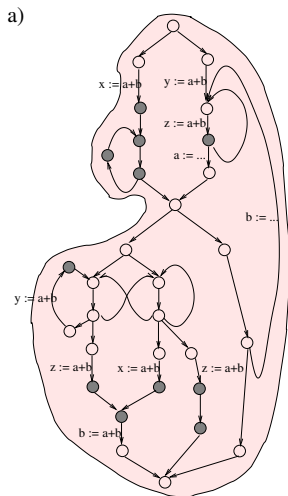
Kap. 12

Kap. 13

436/863

# Aufwand xDFA und ddDFA im Vergleich (1)

Verfügbarkeit an einem Punkt: Vergleich Berechnungsaufwand erschöpfend (*rosa*), anforderungsgetrieben (*grün*)



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

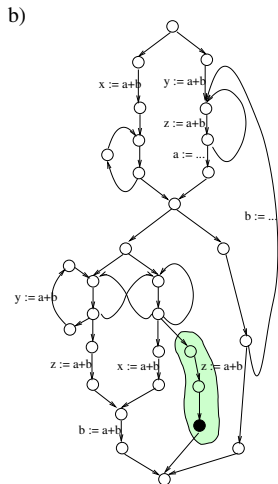
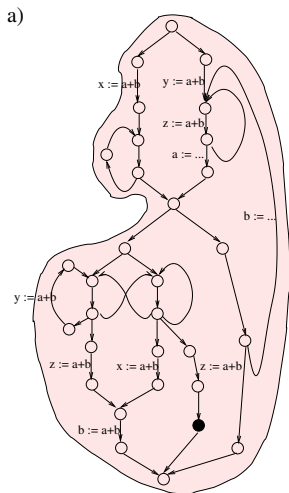
Kap. 11

Kap. 12

Kap. 13

# Aufwand xDFA und ddDFA im Vergleich (2)

Verfügbarkeit an einem Punkt: Vergleich Berechnungsaufwand erschöpfend (*rosa*), anforderungsgetrieben (*grün*)



# Reverse Verfügbarkeit (1)

## Reverse Verfügbarkeit im Detail

Erforderliche Hilfsfunktionen:

$rCst_{\text{wahr}}$ ,  $rCst_{\text{falsch}}$ , und  $rId_{\text{IB}_f}$ :

$$\forall b \in \text{IB}_f. rCst_{\text{wahr}}(b) =_{df} \begin{cases} \text{falsch} & \text{falls } b \in \text{IB} \\ \text{failure} & \text{sonst (d.h. falls } b = \text{failure)} \end{cases}$$

$$\forall b \in \text{IB}_f. rCst_{\text{falsch}}(b) =_{df} \begin{cases} \text{falsch} & \text{falls } b = \text{falsch} \\ \text{failure} & \text{sonst} \end{cases}$$

$$rId_{\text{IB}_f} =_{df} Id_{\text{IB}_f}$$

wobei  $\text{IB}_f =_{df} \{\text{falsch}, \text{wahr}, \text{failure}\}$ .

# Reverse Verfügbarkeit (2)

Reverse abstrakte Semantik für Verfügbarkeit:

## 1. DFA-Verband:

$(\mathcal{C}, \sqcap, \sqcup, \sqsubseteq, \perp, \top) =_{df} (\mathbb{B}_f, \wedge_f, \vee_f, \leq_f, \mathbf{falsch}, \mathbf{failure})$   
mit  $\mathbf{falsch} \leq_f \mathbf{wahr} \leq_f \mathbf{failure}$

## 2. Reverses DFA-Funktional:

$\llbracket \cdot \rrbracket_{av_R} : E \rightarrow (\mathbb{B}_f \rightarrow \mathbb{B}_f)$  definiert durch

$$\forall e \in E. \llbracket e \rrbracket_{av_R} =_{df} \begin{cases} rCst_{\mathbf{wahr}} & \text{falls } \llbracket e \rrbracket_{av} = Cst_{\mathbf{wahr}} \\ rId_{\mathbb{B}_f} & \text{falls } \llbracket e \rrbracket_{av} = Id_{\mathbb{B}} \\ rCst_{\mathbf{falsch}} & \text{falls } \llbracket e \rrbracket_{av} = Cst_{\mathbf{falsch}} \end{cases}$$

**Hinweis:** Siehe auch Ergänzungsfolien zu “Hot Spot”-Programmanalyse und -optimierung auf der Webseite zur LVA.



# Kapitel 8.8

## Zusammenfassung

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

**8.8**

Kap. 9

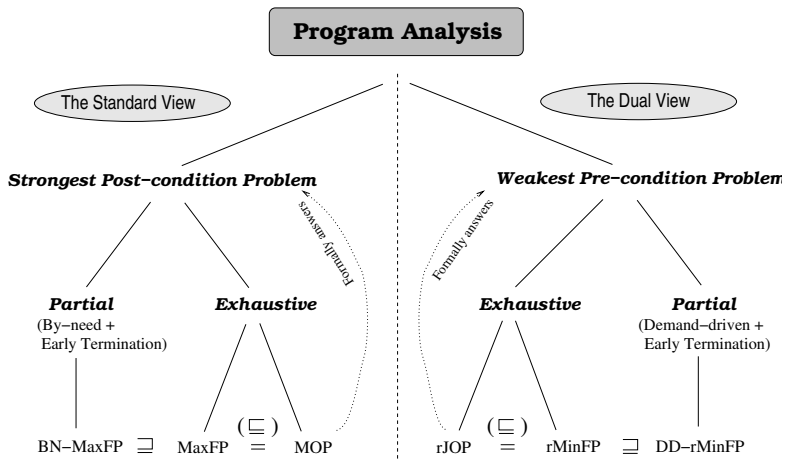
Kap. 10

Kap. 11

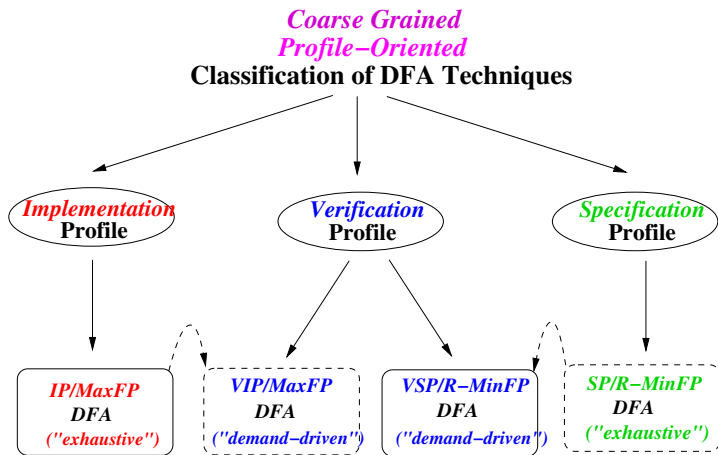
Kap. 12

Kap. 13

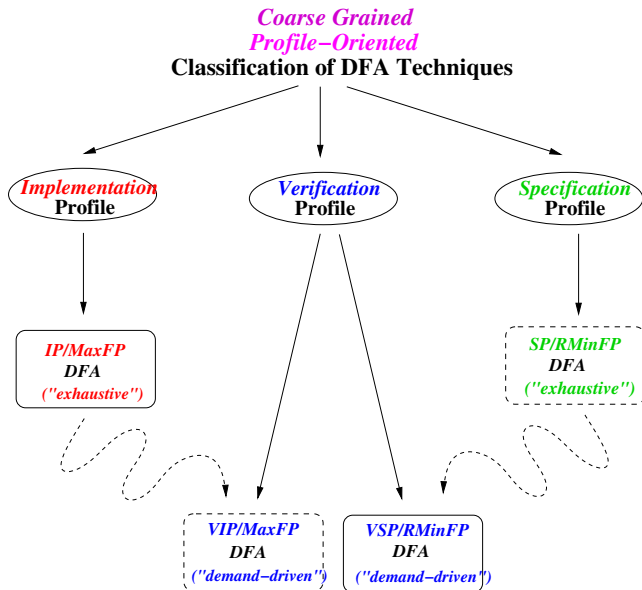
# Gegenüberstellung von xDFA und ddDFA



# Induzierte Lösungsstrategien (1)

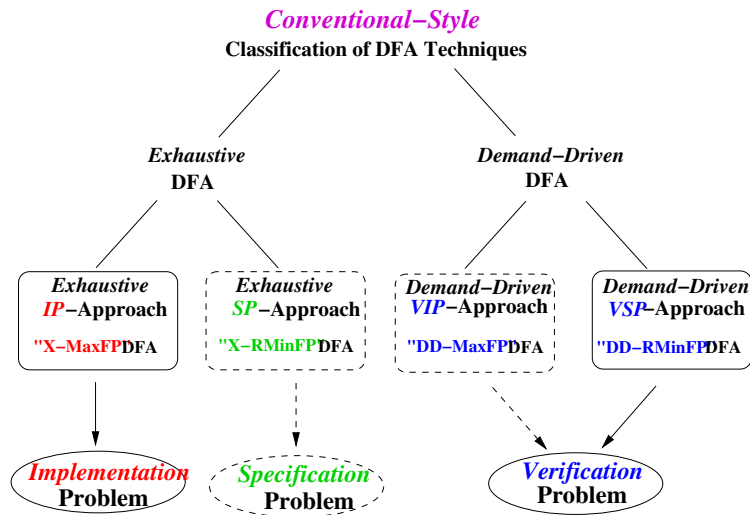


# Induzierte Lösungsstrategien (2)



# Algorithmenorientierte Sicht (1)

...zur Klassifikation von DFA-Problemen.



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

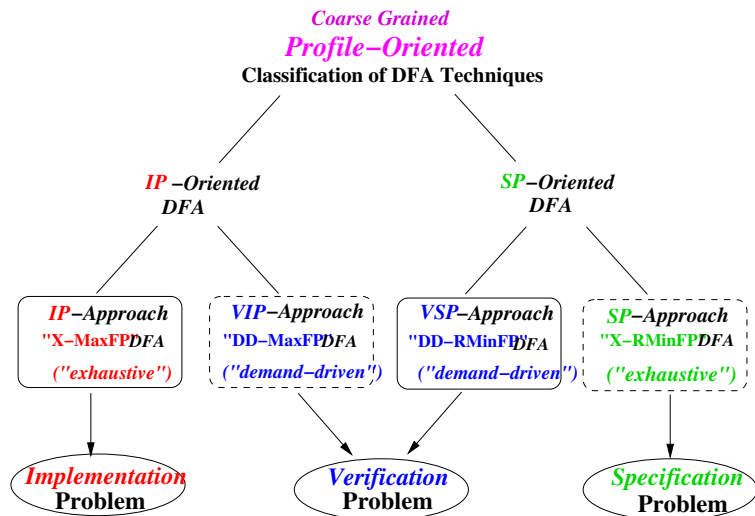
Kap. 12

Kap. 13

445/863

# Algorithmenorientierte Sicht (2)

...zur Klassifikation von DFA-Problemen.



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

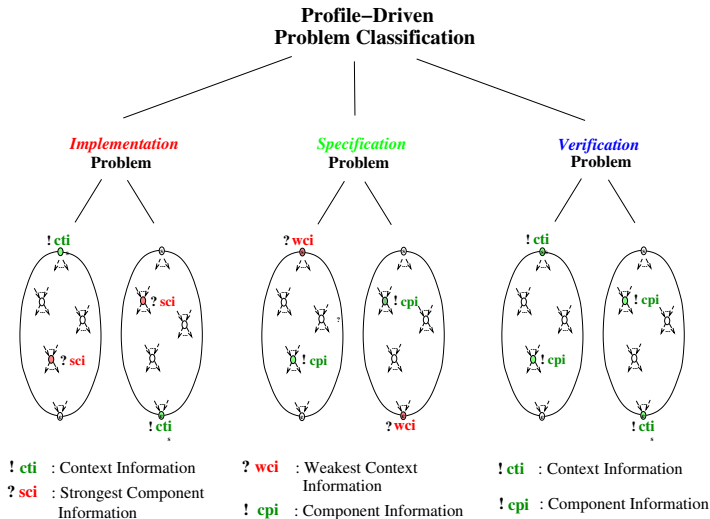
Kap. 12

Kap. 13

446/863

# Programm- und problemorientierte Sicht

...zur Klassifikation von DFA-Problemen.



... where "!" and "?" means **given** and **sought**, respectively.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10




Kap. 11

Kap. 12

Kap. 13

447/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (1)

-  Gagan Agrawal. *Demand-driven Construction of Call Graphs*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 125-140, 2000.
-  Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part I - Exhaustive Analysis*. Acta Informatica 10(3):245-264, 1978.
-  Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

**8.8**

Kap. 9

Kap. 10

Kap. 11


Kap. 12

Kap. 13

448/863



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (2)

-  Ras Bodík, Rajiv Gupta, Vivek Sarkar. *ABCD: Eliminating Array Bounds Check on Demand*. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2000), ACM SIGPLAN Notices 35(5):321-333, 2000.
-  Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, 1996.
-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *Demand-driven Computation of Interprocedural Data Flow*. In Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95), 37-48, 1995.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

**8.8**

Kap. 9




Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (3)

-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Demand-driven Analyzer for Data Flow Testing at the Integration Level*. In Proceedings of the IEEE Conference on Software Engineering (CoSE'96), 575-586, 1996.
-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Practical Framework for Demand-driven Interprocedural Data Flow Analysis*. ACM Transactions on Programming Languages and Systems (TOPLAS) 19(6):992-1030, 1997.
-  Susan Horwitz, Thomas Reps, Mooly Sagiv. *Demand Interprocedural Data Flow Analysis*. In Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-3), 104-115, 1995.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

**8.8**

Kap. 9




Kap. 10

Kap. 11



Kap. 12

Kap. 13




# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (4)

-  John Hughes, John Launchbury. *Reversing Abstract Interpretations*. In Proceedings of the 4th European Symposium on Programming (ESOP'92), Springer-V., LNCS 582, 269-286, 1992.
-  John Hughes, John Launchbury. *Reversing Abstract Interpretations*. Science of Computer Programming 22:307-326, 1994.
-  Jens Knoop. *Demand-driven Analysis of Explicitly Parallel Programs: An Approach based on Reverse Data-Flow Analysis*. In Proceedings of the 9th International Workshop on Compilers for Parallel Computers (CPC 2001), 151-162, 2001.



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (5)

-  Jens Knoop. *Data-Flow Analysis for Hot-Spot Program Optimization*. In Proceedings of the 14th Biennial Workshop on “Programmiersprachen und Grundlagen der Programmierung” (KPS 2007). Bericht A-07-07 der Institute für Mathematik und Informatik, Universität Lübeck, Germany, 124-131, 2007.
-  Yuan Lin, David A. Padua. *Demand-driven Interprocedural Array Property Analysis*. In Proceedings of the 12th International Conference on Languages and Compilers for Parallel Computing (LCPC'99), Springer-V., LNCS, 1999.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (6)

-  Thomas Reps. *Solving Demand Versions of Interprocedural Analysis Problems*. In Proceedings of the 5th International Conference on Compiler Construction (CC'95), Springer-V., LNCS 786, 389-403, 1994.
-  Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Ed.), Kluwer Academic Publishers, 1994.
-  Mary Lou Soffa. *Tutorial: Techniques to improve the Scalability and Precision of Data Flow Analysis*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 355-356, 1999.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 8 (7)

-  Peng Tu, David A. Padua. *Gated SSA-based Demand-driven Symbolic Analysis for Parallelizing Computers*. In Proceedings of the International Conference on Supercomputing (SC'95), 414-423, 1995.
-  X. Yuan, Rajiv Gupta, R. Melham. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

8.1

8.2

8.3

8.4

8.5

8.6

8.7

8.8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

# Kapitel 9

## Chaotische Fixpunktiteration

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

**Kap. 9**

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Kapitel 9.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

**9.1**

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15



# Motivation

Viele **praktisch relevante Probleme** in der **Informatik** lassen sich durch die

- ▶ **kleinste (größte) gemeinsame Lösung**

von **Systemen rekursiver Gleichungen** beschreiben:

$$\begin{aligned}x &= f_1(x) \\ &\vdots \\ x &= f_n(x)\end{aligned}$$

# System rekursiver Gleichungen

Sei

$$\begin{aligned}x &= f_1(x) \\ &\vdots \\ x &= f_n(x)\end{aligned}$$

ein System rekursiver Gleichungen, wobei

$$\mathcal{F} =_{df} \{f_k : D \rightarrow D \mid 1 \leq k \leq n\}$$

eine Familie monotoner Funktionen auf einer wohlfundierten partiellen Ordnung  $(D, \sqsubseteq)$  ist.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

458/863

# Fixpunkte vs. Lösungen

## Fixpunkte von Funktionen vs. Lösungen von Gleichungen

Wir wollen einsehen, dass das

- ▶ Lösen eines Systems rekursiver Gleichungen

$$x = f_1(x)$$

$$\vdots$$

$$x = f_n(x)$$

der

- ▶ Berechnung eines Fixpunktes von  $\mathcal{F}$ , d.h. eines gemeinsamen Fixpunkts  $x = f_k(x)$  für alle  $f_k$ .

entspricht.

# Chaotische Iteration

## Fixpunktberechnung mittels chaotischer Iteration:

- ▶ Ein typischer **Iterationsalgorithmus** beginnt mit dem initialen Wert  $\perp$  für  $x$ , dem kleinsten Element von  $D$ , und aktualisiert sukzessive den Wert von  $x$  durch Anwendung der Funktionen  $f_k$  in einer beliebigen Reihenfolge, um so den kleinsten gemeinsamen Fixpunkt von  $\mathcal{F}$  zu approximieren.

Diese Vorgehensweise wird als **chaotische Iteration** bezeichnet.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

460/863

# Bekannte Fixpunkttheoreme aus der Literatur

Möglicherweise am bekanntesten das

- ▶ **Fixpunkttheorem von Tarski [1955]**
  - ▶ Garantiert Existenz kleinster Fixpunkte für monotone Funktionen über vollständigen partiellen Ordnungen
  - ▶ Iteration:  $\vec{x}_0 = \perp, \vec{x}_1 = \vec{f}(\vec{x}_0), \vec{x}_2 = \vec{f}(\vec{x}_1), \dots$ , wobei  $\vec{x}_i$  den Wert von  $\vec{x}$  nach der  $i$ -ten Iteration bezeichnet.
  - ▶ Vielfach anwendbar, aber dennoch oft zu speziell.

Verallgemeinerungen:

- ▶ **Vektor-Iterationen:** Robert [1976]
- ▶ **Asynchrone Iterationen:** Baudet [1978], Cousot [1977], Üresin/Dubois [1989], Wei [1993]
- ▶ ...

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

461/863

# Verallgemeinerungen

## ▶ Vektor-Iterationen (Robert [1976])

### ▶ Gegeben:

Eine monotone Vektorfunktion  $\vec{f} = (f^1, \dots, f^m)$

### ▶ Gesucht:

Kleinster Fixpunkt  $\vec{x} = (x^1, \dots, x^m) \in D^m$  von  $\vec{f}$

### ▶ Iteration:

$\vec{x}_0 = \vec{1}, \vec{x}_1 = \vec{f}_{J_0}(\vec{x}_0), \vec{x}_2 = \vec{f}_{J_1}(\vec{x}_1), \dots$ , wobei

$J_i \subseteq \{1, \dots, n\}$  und die  $k$ -te Komponente  $\vec{f}_{J_i}(\vec{x}_i)^k$  von  $\vec{f}_{J_i}(\vec{x}_i)$  ist  $f^k(\vec{x}_i)$ , falls  $k \in J_i$ , und  $\vec{x}_i^k$  sonst.

## ▶ Asynchrone Iterationen (Baudet 1978], Cousot [1977], Üresin/Dubois [1989], Wei [1993])

- ▶  $\vec{f}_{J_i}$  kann auf Komponenten früherer Vektoren der Iterationsfolge zurückgreifen  $\vec{x}_j, j \leq i$ .

# Zum Nachschlagen und -lesen

- ▶ Alfred Tarski. *A Lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics, Vol. 5, 285-309, 1955.
- ▶ F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.

Ein historischer Abriss findet sich in:

- ▶ Jean-Louis Lassez, V.L. Nguyen, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

463/863

# In diesem Kapitel

Vorstellung eines weiteren **Fixpunkttheorems**, das **ohne Monotonie** auskommt!

- ▶ Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15



# Kapitel 9.2

## Chaotisches Fixpunktiterationstheorem

# Vorbereitungen (1)

- ▶ Eine **partielle Ordnung**  $(D, \sqsubseteq)$  ist ein Paar aus einer Menge  $D$  und einer reflexiven, antisymmetrischen und transitiven zweistelligen Relation  $\sqsubseteq \subseteq D \times D$ .
- ▶ Eine Folge  $(d_i)_{i \in \mathbb{N}}$  von Elementen  $d_i \in D$  heißt **(aufsteigende) Kette**, falls  $\forall i \in \mathbb{N}. d_i \sqsubseteq d_{i+1}$ .
- ▶ Eine Kette  $T =_{df} (d_i)_{i \in \mathbb{N}}$  heißt **stationär**, falls  $\{d_i \mid i \in \mathbb{N}\}$  endlich ist.
- ▶ Eine partielle Ordnung  $\sqsubseteq$  heißt **wohlfundiert**, falls jede Kette stationär ist.

# Vorbereitungen (2)

- ▶ Eine Funktion  $f : D \rightarrow D$  auf  $D$  heißt
  - ▶ **monoton**, falls  $\forall d, d' \in D. d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$ .
  - ▶ **inflationär (vergrößernd)**, falls  $\forall d \in D. d \sqsubseteq f(d)$
- ▶ Ist  $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$  eine Familie von Funktionen und  $s = (s_1, \dots, s_n) \in \mathbb{N}^*$ , dann ist  $f_s$  definiert durch die Komposition

$$f_s =_{df} f_{s_n} \circ \dots \circ f_{s_1}$$

# Strategien, Iterationsfolgen, Fairness

## Definition (9.2.1, Strategie, Chaotische Iterationsfolge, Fairness)

Sei  $(D, \sqsubseteq)$  eine partielle Ordnung und  $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$  eine Familie inflationärer Funktionen  $f_k : D \rightarrow D$ .

- ▶ Eine **Strategie** ist eine beliebige Funktion  $\gamma : \mathbb{N} \rightarrow \mathbb{N}$ .
- ▶ Eine Strategie  $\gamma$  und ein Element  $d \in D$  induzieren eine **chaotische Iteration**  $f_\gamma(d) = (d_i)_{i \in \mathbb{N}}$  von Elementen  $d_i \in D$ , die induktiv definiert sind durch  $d_0 = d$  und  $d_{i+1} = f_{\gamma(i)}(d_i)$ .
- ▶ Eine Strategie  $\gamma$  heißt **fair** gdw

$$\forall i, k \in \mathbb{N}. (f_k(d_i) \neq d_i \text{ impliziert } \exists j > i. d_j \neq d_i)$$

# Ein abgeschwächter Monotoniebegriff

## Definition (9.2.2, Verzögerungsmonotonie)

Sei  $(D, \sqsubseteq)$  eine partielle Ordnung und  $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$  eine Familie von Funktionen  $f_k : D \rightarrow D$ . Dann heißt  $\mathcal{F}$  **verzögert-monoton**, falls für alle  $k \in \mathbb{N}$  gilt:

$$d \sqsubseteq d' \text{ impliziert } \exists s \in \mathbb{N}^*. f_k(d) \sqsubseteq f_s(d')$$

## Lemma (9.2.3)

$\mathcal{F}$  ist verzögert-monoton, wenn alle  $f_k$  im üblichen Sinn monoton sind.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

**9.2**

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

469/863

# Das “monotoniefreie” Fixpunkttheorem

## Theorem (9.2.4, Chaotische Fixpunktiteration)

Sei  $(D, \sqsubseteq)$  eine wohlfundierte partielle Ordnung mit kleinstem Element  $\perp$ ,  $\mathcal{F} =_{df} (f_k)_{k \in \mathbb{N}}$  eine verzögert-monotone Familie inflationärer Funktionen und  $\gamma : \mathbb{N} \rightarrow \mathbb{N}$  eine faire Strategie.

Dann gilt:

1. Der kleinste gemeinsame Fixpunkt  $\mu\mathcal{F}$  von  $\mathcal{F}$  existiert und ist gegeben durch  $\bigsqcup f_\gamma(\perp)$ .
2.  $\mu\mathcal{F}$  wird stets in einer endlichen Zahl von Iterationsschritten erreicht.

# Generischer Fixpunktalgorithmus 9.2.5

Nichtdeterministischer 'Rumpf'-Algorithmus:

```
 $d := \perp;$   
while  $\exists k \in IN. d \neq f_k(d)$  do  
    choose  $k \in IN$  where  $d \sqsubset f_k(d)$  in  
         $d := f_k(d)$   
    ni  
od
```

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

**9.2**

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

471/863

# Kapitel 9.3

## Anwendungen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

**9.3**

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15



# Kapitel 9.3.1

## Vektor-Iteration

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

**9.3.1**

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Vorbereitungen (1)

- ▶ Sei  $(C, \sqsubseteq_C)$  eine wohlfundierte partielle Ordnung und  $D = C^n$  für ein  $n \in \mathbb{N}$ , geordnet durch die punktweise Ausdehnung von  $\sqsubseteq_C$ .
- ▶ Sei  $f : D \rightarrow D$  eine monotone Funktion.
- ▶ Anstelle der Iteration  $d_1 = f(\perp)$ ,  $d_2 = f(d_1)$ ,  $\dots$  im Stil von Tarskis Fixpunkttheorem, können wir zu einer Zerlegung von  $f$  in seine Komponenten  $f^k$  übergehen, d.h.  $f(d) = (f^1(d), \dots, f^n(d))$  unter Ausführung selektiver Aktualisierungen
- ▶ Hier und in der Folge benutzen wir obere Indizes  $i$ , um die  $i$ -te Komponente eines Vektors der Länge  $n$  zu bezeichnen.

# Vorbereitungen (2)

## Definition (9.3.1.1, Vektor-Iteration)

Eine **Vektor-Iteration** ist eine Iteration der Form

$d_1 = f_{J_0}(\perp), d_2 = f_{J_1}(d_1), \dots$ , wobei  $J_i \subseteq \{1, \dots, n\}$  und

$$f_J(d)^i =_{df} \begin{cases} f^i(d) & \text{falls } i \in J \\ d^i & \text{sonst} \end{cases}$$

eine selektive Aktualisierung der durch  $J$  spezifizierten Komponenten durchführt.

Es gilt:

- ▶ Die Menge der gemeinsamen Fixpunkte der Funktionenfamilie  $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \dots, n\}\}$  ist gleich der Menge der Fixpunkte von  $f$ .
- ▶ Jedes  $f_J$  ist monoton, da  $f$  monoton ist.

# Anwendung von Fixpunkttheorem 9.2.4

...zur Modellierung der Vektor-Iteration.

**Erforderlich:** Verallgemeinerung des Strategiebegriffs auf einen Strategiebegriff für **Mengen**.

## Definition (9.3.1.2, (Faire) Mengenstrategie)

- ▶ Eine **Mengenstrategie** ist eine (beliebige) Funktion  $\gamma : \mathbb{N} \rightarrow \mathcal{P}(\{1, \dots, n\})$ .

**Intuition:**  $\gamma(i)$  liefert eine Menge  $J_i$  von Indizes aus  $\{1, \dots, n\}$ , deren zugehörige Komponenten in Schritt  $i$  aktualisiert werden sollen.

- ▶ Eine Mengenstrategie heißt **fair** gdw

$$\forall i \in \mathbb{N}, J \subseteq \mathbb{N}. (f_J(d_i) \neq d_i \text{ impliziert } \exists j > i. d_j \neq d_i)$$

# Modellierungsergebnisse (1)

## Lemma (9.3.1.3, Vektor-Iteration)

Sei  $(C, \sqsubseteq_C)$  eine wohlfundierte partielle Ordnung mit kleinstem Element  $\perp_C$ , sei  $n \in \mathbb{N}$  und sei  $D = C^n$  geordnet durch die punktweise Ausdehnung von  $\sqsubseteq_C$  auf  $\sqsubseteq$ . Sei  $f = (f^1, \dots, f^n)$  eine monotone Funktion auf  $D$ , sei  $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \dots, n\}\}$  mit Funktionen  $f_J : D \rightarrow D$  wie zuvor definiert und sei  $\gamma : \mathbb{N} \rightarrow \mathcal{P}(\{1, \dots, n\})$  eine Mengenstrategie.

Dann gilt: Jede chaotische Iteration  $f_\gamma(\perp)$  liefert eine Kette.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Modellierungsergebnisse (2)

## Korollar (9.3.1.4, Chaotische Vektor-Iteration)

Sei  $(C, \sqsubseteq_C)$  eine wohlfundierte partielle Ordnung mit kleinstem Element  $\perp_C$ , sei  $n \in \mathbb{N}$  und sei  $D = C^n$  geordnet durch die punktweise Erweiterung von  $\sqsubseteq_C$  auf  $\sqsubseteq$ . Sei  $f = (f^1, \dots, f^n)$  eine monotone Funktion auf  $D$ , sei  $\mathcal{F} =_{df} \{f_J \mid J \subseteq \{1, \dots, n\}\}$  und sei  $\gamma$  eine faire Mengenstrategie.

Dann gilt:

- ▶  $\sqcup f_\gamma(\perp)$  ist der kleinste Fixpunkt  $\mu\mathcal{F}$  von  $\mathcal{F}$ .
- ▶  $\mu\mathcal{F} = \mu f$ .
- ▶  $\mu\mathcal{F}$  ist stets in einer endlichen Zahl von Iterationsschritten erreicht.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Modellierungsergebnisse (3)

## Bemerkungen:

- ▶ Korollar 9.3.1.4 ist ein Spezialfall von Fixpunktheorem 9.2.4 für Vektor-Iterationen und folgt zusammen mit Lemma 9.3.1.3.
- ▶ Für  $|\mathcal{F}| = 1$  reduziert sich Korollar 9.3.1.4 auf Tarskis Fixpunktheorem im Fall wohlfundierter partieller Ordnungen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Kapitel 9.3.2

## Datenflussanalyse

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

**9.3.2**

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15



# Anwendung von Fixpunkttheorem 9.2.4

...auf intraprozedurale DFA.

Erinnerung:

Das *MaxFP*-Gleichungssystem:

$$\text{inf}(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigsqcap \{ \llbracket (m, n) \rrbracket (\text{inf}(m)) \mid m \in \text{pred}(n) \} & \text{sonst} \end{cases}$$

Die *MaxFP*-Lösung:

Die größte Lösung des *MaxFP*-Gleichungssystems, bezeichnet mit  $\text{inf}_{c_s}^*$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

481/863

# Dual dazu: *MinFP*-Gleichungssystem 9.3.2.1

Das *MinFP*-Gleichungssystem:

$$\mathit{inf}(n) = \begin{cases} c_s & \text{falls } n = s \\ \sqcup \{ \llbracket (m, n) \rrbracket (\mathit{inf}(m)) \mid m \in \mathit{pred}(n) \} & \text{sonst} \end{cases}$$

Die *MinFP*-Lösung:

Die kleinste Lösung des *MinFP*-Gleichungssystems, bezeichnet mit  $\mathit{inf}_{c_s}^*$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

## Generischer *MinFP*-Fixpunktalg. 9.3.2.2

```
inf[s] :=  $c_s$ ;  
forall  $n \in N \setminus \{s\}$  do inf[ $n$ ] :=  $\perp$  od;  
workset :=  $N$ ;  
while workset  $\neq \emptyset$  do  
  choose  $n \in$  workset in  
    workset := workset  $\setminus \{n\}$ ;  
    new := inf[ $n$ ]  $\sqcup \bigsqcup \{ \llbracket (m, n) \rrbracket (\mathbf{inf}[m]) \mid m \in \text{pred}_G(n) \}$ ;  
    if new  $\sqsupset$  inf[ $n$ ] then  
      inf[ $n$ ] := new;  
      workset := workset  $\cup \text{succ}_G(n)$   
    fi  
  ni  
od
```

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

483/863

# Zur Fixpunktcharakt. der *MinFP*-Lösung (1)

## Vorbereitung

- ▶ Sei  $G = (N, E, \mathbf{s}, \mathbf{e})$  der betrachtete Flussgraph.
- ▶ Sei  $\llbracket \cdot \rrbracket : E \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$  ein monotones DFA-Funktional, das die lokale abstrakte Semantik von  $G$  festlegt.
- ▶ Die Menge der Knoten  $N$  werde mit der Menge der natürlichen Zahlen  $\{1, \dots, n\}$  identifiziert, wobei  $n$  die Anzahl der Knoten von  $N$  bezeichnet.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

484/863

# Zur Fixpunktcharakt. der *MinFP*-Lösung (2)

Sei  $D =_{df} \mathcal{C}^n$  versehen mit der punktweisen Ausdehnung v.  $\sqsubseteq_{\mathcal{C}}$ .

Dann gilt:

- ▶  $(D, \sqsubseteq)$  ist eine wohlfundierte partielle Ordnung.
- ▶ Ein Wert  $d = (d^1, \dots, d^n)$  stellt eine Annotation des Flussgraphen dar, wobei dem Knoten  $k$  der Wert  $d^k$  zugewiesen ist.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

485/863

## Zur Fixpunktcharakt. der *MinFP*-Lösung (3)

Für jeden Knoten  $k$  des Flussgraphen definieren wir jetzt eine Funktion  $f^k : D \rightarrow C$  durch

$$f^k(d^1, \dots, d^n) =_{df} d'^k$$

wobei

$$d'^k = d^k \sqcup \bigsqcup \{ \llbracket (m, k) \rrbracket (d^m) \mid m \in \text{pred}_G(k) \}$$

**Intuitiv:**  $f^k$  beschreibt den Effekt der Berechnung der lokalen abstrakten Semantik am Knoten  $k$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

# Charakterisierungsergebnisse (1)

## Lemma (9.3.2.3)

*Für alle  $d \in D$  gilt:  $d$  ist eine Lösung des MinFP-Gleichungssystems gdw  $d$  ist Fixpunkt von  $f =_{df} (f^1, \dots, f^n)$ .*

## Theorem (9.3.2.4, Korrektheit und Terminierung)

*Jeder Lauf des generischen MinFP-Fixpunktalgorithmus 9.3.2.2 terminiert mit der MinFP-Lösung.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

9.3.2

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

487/863

# Charakterisierungsergebnisse (2)

## Anmerkungen zum Beweis von Theorem 9.3.2.4

- ▶ Der *MinFP*-Fixpunktalgorithmus 9.3.2.2 folgt dem Muster von Rumpfalgorithmus 9.2.5 mit  $\mathcal{F} = \{f_{\{k\}} \mid 1 \leq k \leq n\}$ .
- ▶ Die Verwendung von Variable *workset*, die die Invariante  $workset \supseteq \{k \mid f_{\{k\}}(d) \neq d\}$  erfüllt, trägt zu höherer Effizienz bei.
- ▶ Offenbar gilt:  $f$  ist monoton.
- ▶ Somit sind insgesamt die Voraussetzungen von Korollar 9.3.1.4 erfüllt, womit Theorem 9.3.2.4 folgt.



Weitere Anwendungen des “monotoniefreien” Fixpunkttheorems 9.2.4 in Kapitel 11 und 12 zum Beweis der **Optimalität** von



- ▶ Partially Dead-Code Elimination
- ▶ Partially Redundant-Assignment Elimination

Andere Fixpunkttheoreme sind dafür nicht anwendbar.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 9 (1)

-  Patrick Cousot, Radhia Cousot. *Constructive Versions of Tarski's Fixed Point Theorems*. Pacific Journal of Mathematics 82(1):43-57, 1979.
-  Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.
-  Jean-Louis Lassez, V.L. Nguyen, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 9 (2)

-  F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.
-  Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5:285-309, 1955.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

9.1

9.2

9.3

9.3.1

**9.3.2**

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

491/863

# Kapitel 10

## Basisblock- vs. Einzelanweisungsgraphen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

**Kap. 10**

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Kapitel 10.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

**10.1**

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Basisblock- vs. Einzelanweisungsgraphen

In diesem Kapitel untersuchen wir die **Zweckmäßigkeit** unterschiedlicher Programmrepräsentationen.

Dazu betrachten und vergleichen wir Programme in Form von **knoten- und kantenbenannten Flussgraphen** mit **Basisblöcken** und **Einzelanweisungen** und untersuchen ihre jeweiligen

- ▶ **Vor- und Nachteile für die Programmanalyse**

...und gehen somit der Frage nach:

- ▶ **Basisblock vs. Einzelanweisungsgraphen: Eine Geschmacksfrage?**

Nebenbei werden wir kennenlernen:

- ▶ Einige weitere Beispiele konkreter **Datenflussanalyseprobleme** und **Datenflussanalysen**

# Basisblöcke: Vermeintliche Vorteile

...und die ihnen allgemein zugeschriebenen Anwendungsvorteile (**“Folk Knowledge”**):

**Bessere Skalierbarkeit**, da

- ▶ weniger Knoten in die (potentiell) berechnungsaufwändige iterative Fixpunktberechnung involviert sind und somit
- ▶ größere Programme in den Hauptspeicher passen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

495/863

# Basisblöcke: Sichere Nachteile

...und die nachfolgend gezeigten Anwendungsnachteile:

- ▶ **Höhere konzeptuelle Komplexität:** Basisblöcke führen zu einer unerwünschten **Hierarchisierung**, die sowohl theoretische Überlegungen wie praktische Implementierungen erschwert.
- ▶ **Notwendigkeit von Prä- und Postprozessen:** Sind i.a. erforderlich, um die hierarchie-induzierten Zusatzprobleme zu behandeln (z.B. für **dead code elimination**, **constant propagation**, ...); oder **“trickbehaftete”** Formulierungen nötig macht, um sie zu vermeiden (z.B. für **partial redundancy elimination**).
- ▶ **Eingeschränkte Allgemeinheit:** Bestimmte praktisch relevante Analysen und Optimierungen sind nur schwer oder gar nicht auf der Ebene von Basisblöcken auszudrücken (z.B. **faint variable elimination**).

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

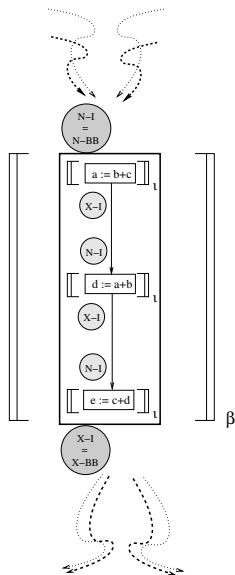
10.5

Kap. 11



# Hierarchisierung durch Basisblöcke

Veranschaulichung:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

**10.1**

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# In der Folge

## Untersuchung von

- ▶ Vor- und Nachteilen von **Basisblock- (BB)** gegenüber **Einzelanweisungsgraphen (EA)**

## anhand von Beispielen

- ▶ einiger bereits von uns betrachteter
  - ▶ **Verfügbarkeit von Ausdrücken**
  - ▶ **Einfache Konstanten**und neuer Datenflussanalyseprobleme
  - ▶ **Schattenhafte (faint) Variablen**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

498/863

# Kapitel 10.1.1

## Kantenbenannter Einzelanweisungsansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

**10.1.1**

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# MOP<sub>EA</sub>-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die MOP-Lösung:

$$\forall c_s \in \mathcal{C} \forall n \in \mathbb{N}. \text{MOP}_{(\llbracket \cdot \rrbracket_l, c_s)}(n) =_{df} \bigsqcap \{ \llbracket p \rrbracket_l(c_s) \mid p \in \mathbf{P}_G[s, n] \}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

**10.1.1**

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# MaxFP<sub>EA</sub>-Ansatz

...für kantenbenannte Einzelanweisungsgraphen.

Die MaxFP-Lösung:

$$\forall c_s \in \mathcal{C} \forall n \in N. \text{MaxFP}_{(\llbracket \cdot \rrbracket_\ell, c_s)}(n) =_{df} \text{inf}_{c_s}^*(n)$$

wobei  $\text{inf}_{c_s}^*$  die größte Lösung des MaxFP-Gleichungssystems bezeichnet:

$$\text{inf}(n) = \begin{cases} c_s & \text{falls } n = s \\ \bigcap \{ \llbracket (m, n) \rrbracket_\ell(\text{inf}(m)) \mid m \in \text{pred}_G(n) \} & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

501/863

# Kapitel 10.1.2

## Knotenbenannter Basisblockansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

**10.1.2**

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Bezeichnungen

In der Folge bezeichnen wir

- ▶ **Basisblockknoten** mit fett gesetzten Buchstaben (**m**, **n**,...)
- ▶ **Einzelanweisungsknoten** mit normal gesetzten Buchstaben (**m**, **n**,...)

Weiters bezeichnen wir mit

- ▶  $\llbracket \cdot \rrbracket_\beta$  und
- ▶  $\llbracket \cdot \rrbracket_\iota$

(lokale) abstrakte Datenflussanalysefunktionale auf **Basisblock-**  
bzw. **Einzelanweisungs-/Instruktions-Ebene**.

# $MOP_{BB}$ -Ansatz (1)

...für knotenbenannte Basisblockgraphen.

Die  $MOP$ -Lösung auf BB-Ebene:

$$\forall c_s \in \mathcal{C} \quad \forall \mathbf{n} \in \mathbf{N}. \quad MOP_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\mathbf{n}) =_{df} \\ (N-MOP_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\mathbf{n}), X-MOP_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\mathbf{n}))$$

mit

$$N-MOP_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\mathbf{n}) =_{df} \bigcap \{ \llbracket p \rrbracket_{\beta}(c_s) \mid p \in \mathbf{P}_G[\mathbf{s}, \mathbf{n}] \}$$

$$X-MOP_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\mathbf{n}) =_{df} \bigcap \{ \llbracket p \rrbracket_{\beta}(c_s) \mid p \in \mathbf{P}_G[\mathbf{s}, \mathbf{n}] \}$$



# MOP<sub>BB</sub>-Ansatz (2)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall c_s \in \mathcal{C} \quad \forall n \in \mathbb{N}. \quad \text{MOP}_{(\mathbb{I}_L, c_s)}(n) =_{df} \\ (N\text{-MOP}_{(\mathbb{I}_L, c_s)}(n), X\text{-MOP}_{(\mathbb{I}_L, c_s)}(n))$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

**10.1.2**

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# MOP<sub>BB</sub>-Ansatz (3)

...mit

$$N\text{-MOP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n) \stackrel{df}{=} \begin{cases} N\text{-MOP}_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\text{block}(n)) \\ \quad \text{falls } n = \text{start}(\text{block}(n)) \\ \\ \llbracket p \rrbracket_{\ell}(N\text{-MOP}_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\text{block}(n))) \\ \quad \text{sonst } (p \text{ Präfixpfad} \\ \quad \text{von } \text{start}(\text{block}(n)) \\ \quad \text{bis (ausschließlich) } n) \end{cases}$$

$$X\text{-MOP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n) \stackrel{df}{=} \llbracket p \rrbracket_{\ell}(N\text{-MOP}_{(\llbracket \cdot \rrbracket_{\beta}, c_s)}(\text{block}(n))) \\ (p \text{ Präfix von } \text{start}(\text{block}(n)) \\ \text{bis (einschließlich) } n)$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# MaxFP<sub>BB</sub>-Ansatz (1)

...für knotenbenannte Basisblockgraphen:

Die *MaxFP*-Lösung auf BB-Ebene:

$$\forall c_s \in \mathcal{C} \forall \mathbf{n} \in \mathbf{N}. \text{MaxFP}_{(\llbracket \cdot \rrbracket_{\beta, c_s})}(\mathbf{n}) =_{df} \\ (N\text{-MFP}_{(\llbracket \cdot \rrbracket_{\beta, c_s})}(\mathbf{n}), X\text{-MFP}_{(\llbracket \cdot \rrbracket_{\beta, c_s})}(\mathbf{n}))$$

mit

$$N\text{-MFP}_{(\llbracket \cdot \rrbracket_{\beta, c_s})}(\mathbf{n}) =_{df} \text{pre}_{c_s}^{\beta}(\mathbf{n}) \quad \text{und} \\ X\text{-MFP}_{(\llbracket \cdot \rrbracket_{\beta, c_s})}(\mathbf{n}) =_{df} \text{post}_{c_s}^{\beta}(\mathbf{n})$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

507/863

## MaxFP<sub>BB</sub>-Ansatz (2)

...wobei  $pre_{c_s}^\beta$  und  $post_{c_s}^\beta$  die größten Lösungen des Gleichungssystems

$$pre(\mathbf{n}) = \begin{cases} c_s & \text{falls } \mathbf{n} = \mathbf{s} \\ \sqcap \{ post(\mathbf{m}) \mid \mathbf{m} \in pred_G(\mathbf{n}) \} & \text{sonst} \end{cases}$$

$$post(\mathbf{n}) = \llbracket \mathbf{n} \rrbracket_\beta(pre(\mathbf{n}))$$

bezeichnen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

**10.1.2**

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

508/863

# MaxFP<sub>BB</sub>-Ansatz (3)

...und ihre Fortsetzung auf EA-Ebene:

$$\forall c_s \in \mathcal{C} \forall n \in N. \text{MaxFP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n) =_{df} \\ (N\text{-MFP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n), X\text{-MFP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n))$$

mit

$$N\text{-MFP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n) =_{df} \text{pre}_{c_s}^{\ell}(n) \quad \text{und}$$

$$X\text{-MFP}_{(\llbracket \cdot \rrbracket_{\ell}, c_s)}(n) =_{df} \text{post}_{c_s}^{\ell}(n)$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

509/863

## BB-*MaxFP*-Ansatz (4)

...wobei  $pre_{c_s}^l$  und  $post_{c_s}^l$  die größten Lösungen des Gleichungssystems

$$pre(n) = \begin{cases} pre_{c_s}^\beta(\text{block}(n)) \\ \quad \text{falls } n = \text{start}(\text{block}(n)) \\ \\ post(m) \\ \quad \text{sonst (} m \text{ ist hier der eindeutig} \\ \quad \text{bestimmte Vorgänger von } n \\ \quad \text{in } \text{block}(n)) \end{cases}$$

$$post(n) = \llbracket n \rrbracket_l(pre(n))$$

bezeichnen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

510/863

# Kapitel 10.2

## Verfügbarkeit von Ausdrücken

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

**10.2**

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Kapitel 10.2.1

## Knotenbenannter Basisblockansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

**10.2.1**

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11



# Verfügbarkeit von Ausdrücken (1)

...für knotenbenannte BB-Graphen.

## Phase I: Die Basisblockebene

Lokale Prädikate (assoziiert mit BB-Knoten):

- ▶  $\text{BB-XCOMP}_\beta(t)$ :  $\beta$  enthält eine Anweisung  $\iota$ , die  $t$  berechnet, und weder  $\iota$  noch eine auf  $\iota$  folgende Anweisung in  $\beta$  modifiziert einen Operanden von  $t$ .
- ▶  $\text{BB-TRANSP}_\beta(t)$ :  $\beta$  enthält keine Anweisung, die einen Operanden von  $t$  modifiziert.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Verfügbarkeit von Ausdrücken (2)

Das BB-Gleichungssystem von Phase I:

$$\text{BB-N-AVAIL}_\beta = \begin{cases} \text{falsch} & \text{falls } \beta = \mathbf{s} \\ \prod_{\hat{\beta} \in \text{pred}(\beta)} \text{BB-X-AVAIL}_{\hat{\beta}} & \text{sonst} \end{cases}$$

$$\text{BB-X-AVAIL}_\beta = \text{BB-N-AVAIL}_\beta \cdot \text{BB-TRANSP}_\beta + \text{BB-XCOMP}_\beta$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

**10.2.1**

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Verfügbarkeit von Ausdrücken (3)

## Phase II: Die Anweisungsebene

Lokale Prädikate (assoziiert mit EA-Knoten):

- ▶  $COMP_{\iota}(t)$ :  $\iota$  berechnet  $t$ .
- ▶  $TRANSP_{\iota}(t)$ :  $\iota$  modifiziert keinen Operanden von  $t$ .
- ▶  $BB-N-AVAIL^*$ ,  $BB-X-AVAIL^*$ : größte Lösung des BB-Gleichungssystem von Phase I.

Das EA-Gleichungssystem von Phase II:

$$N-AVAIL_{\iota} = \begin{cases} BB-N-AVAIL_{block(\iota)}^* & \text{falls } \iota = start(block(\iota)) \\ X-AVAIL_{pred(\iota)} & \text{sonst} \end{cases} \quad (\text{beachte: } |pred(\iota)| = 1)$$

$$X-AVAIL_{\iota} = \begin{cases} BB-X-AVAIL_{block(\iota)}^* & \text{falls } \iota = end(block(\iota)) \\ (N-AVAIL_{\iota} + COMP_{\iota}) \cdot TRANSP_{\iota} & \text{sonst} \end{cases}$$

# Kapitel 10.2.2

## Knotenbenannter Einzelanweisungsansatz

# Verfügbarkeit von Ausdrücken

...für knotenbenannte EA-Graphen.

Lokale Prädikate (assoziiert mit EA-Knoten):

- ▶  $COMP_\iota(t)$ :  $\iota$  berechnet  $t$ .
- ▶  $TRANSP_\iota(t)$ :  $\iota$  modifiziert keinen Operanden von  $t$ .

Das EA-Gleichungssystem:

$$N-AVAIL_\iota = \begin{cases} \mathbf{falsch} & \text{falls } \iota = s \\ \prod_{\hat{\iota} \in pred(\iota)} X-AVAIL_{\hat{\iota}} & \text{sonst} \end{cases}$$

$$X-AVAIL_\iota = (N-AVAIL_\iota + COMP_\iota) \cdot TRANSP_\iota$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

517/863

# Kapitel 10.2.3

## Kantenbenannter Einzelanweisungsansatz

# Verfügbarkeit von Ausdrücken

...für kantenbenannte EA-Graphen.

Lokale Prädikate (assoziiert mit EA-Kanten):

- ▶  $COMP_{\varepsilon}(t)$ : Anweisung  $\iota$  von Kante  $\varepsilon$  berechnet  $t$ .
- ▶  $TRANSP_{\varepsilon}(t)$ : Anweisung  $\iota$  von Kante  $\varepsilon$  modifiziert keinen Operanden von  $t$ .

Das EA-Gleichungssystem:

$$Avail_n = \begin{cases} \text{falsch} & \text{falls } n = s \\ \prod_{m \in pred(n)} (Avail_m + COMP_{(m,n)}) \cdot TRANSP_{(m,n)} & \\ \text{sonst} & \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

519/863

# In den folgenden Kapiteln

...zwei weitere Beispiele zur Veranschaulichung des Einflusses der Flussgraphdarstellungsvariante:

- ▶ Konstantenausbreitung und -faltung (Constant Propagation and Folding)
- ▶ Schattenvariablenelimination (Faint Variable Elimination)

Dabei Formulierung dieser Probleme für die Varianten:

- ▶ knotenbenannte Basisblockgraphen
- ▶ kantenbenannte Einzelanweisungsgraphen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

**10.2.3**

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11



# Kapitel 10.3

## Konstantenausbreitung und -faltung

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

**10.3**

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Konstantenausbreitung und -faltung

... am Beispiel "Einfacher Konstanten".

Wir benötigen dazu zwei Hilfsfunktionen:

- ▶ Rückwärtssubstitution
- ▶ Zustandstransformation(sfunktion)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

**10.3**

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Rückwärtssubstitution und Zustandstransformation auf Anweisungen

Sei  $\iota \equiv (x := t)$  eine Anweisung. Dann definieren wir:

- ▶ Rückwärtssubstitution

$\delta_\iota : \mathbf{T} \rightarrow \mathbf{T}$  durch  $\delta_\iota(s) =_{df} s[t/x]$  für alle  $s \in \mathbf{T}$ , wobei  $s[t/x]$  die simultane Ersetzung aller Vorkommen von  $x$  in  $s$  durch  $t$  bezeichnet.

- ▶ Zustandstransformation

$$\theta_\iota(\sigma)(y) =_{df} \begin{cases} \mathcal{E}(t)(\sigma) & \text{falls } y = x \\ \sigma(y) & \text{sonst} \end{cases}$$

# Zusammenhang von $\delta$ und $\theta$

Bezeichne  $\mathcal{I}$  die Menge aller Anweisungen.

Lemma (10.1, Substitutionslemma)

$$\forall t \in \mathbf{T} \forall \sigma \in \Sigma \forall l \in \mathcal{I}. \mathcal{E}(\delta_l(t))(\sigma) = \mathcal{E}(t)(\theta_l(\sigma))$$

**Beweis** induktiv über den Aufbau von  $t$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

**10.3**

10.3.1

10.3.2

10.4

10.5

Kap. 11

524/863

# Kapitel 10.3.1

## Kantenbenannter Einzelanweisungsansatz

# Einfache Konstanten für den EA-Fall

...für kantenbenannte Einzelanweisungsgraphen.

- ▶  $CP_n \in \Sigma$
- ▶  $\sigma_0 \in \Sigma$  Anfangszusicherung

Das EA-Gleichungssystem:

$$\forall v \in \mathbf{V}. CP_n = \begin{cases} \sigma_0(v) & \text{falls } n = s \\ \prod \{ \mathcal{E}(\delta_{(m,n)}(v))(CP_m) \mid m \in \text{pred}(n) \} & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

**10.3.1**

10.3.2

10.4

10.5

Kap. 11

# Kapitel 10.3.2

## Knotenbenannter Basisblockansatz

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

**10.3.2**

10.4

10.5

Kap. 11

# Rückwärtssubstitution und Zustandstransformation auf Pfaden

Ausdehnung von  $\delta$  und  $\theta$  auf Pfade (und somit insbesondere auch auf Basisblöcke):

- ▶  $\Delta_p : \mathbf{T} \rightarrow \mathbf{T}$  definiert durch  $\Delta_p =_{df} \delta_{n_q}$  für  $q = 1$  und durch  $\Delta_{(n_1, \dots, n_{q-1})} \circ \delta_{n_q}$  für  $q > 1$
- ▶  $\Theta_p : \Sigma \rightarrow \Sigma$  definiert durch  $\Theta_p =_{df} \theta_{n_1}$  für  $q = 1$  und durch  $\Theta_{(n_2, \dots, n_q)} \circ \theta_{n_1}$  für  $q > 1$ .



# Zusammenhang von $\Delta$ und $\Theta$

Bezeichne  $\mathcal{B}$  die Menge aller Basisblöcke.

Lemma (10.2, Verallgemeinertes Substitutionslemma)

$$\forall t \in \mathbf{T} \forall \sigma \in \Sigma \forall \beta \in \mathcal{B}. \mathcal{E}(\Delta_\beta(t))(\sigma) = \mathcal{E}(t)(\Theta_\beta(\sigma))$$

**Beweis** induktiv über die Länge von  $p$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

**10.3.2**

10.4

10.5

Kap. 11

529/863

# Einfache Konstanten für den BB-Fall (1)

...für knotenbenannte Basisblockgraphen.

## Phase I: Basisblockebene

Bemerkung:

- ▶  $\Delta_\beta(v) =_{df} \delta_{\iota_1} \circ \dots \circ \delta_{\iota_q}(v)$ , wobei  $\beta \equiv \iota_1; \dots; \iota_q$ .
- ▶  $\text{BB-N-CP}_\beta, \text{BB-X-CP}_\beta, \text{N-CP}_\iota, \text{X-CP}_\iota \in \Sigma$
- ▶  $\sigma_0 \in \Sigma$  Anfangszusicherung

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

**10.3.2**

10.4

10.5

Kap. 11

# Einfache Konstanten für den BB-Fall (2)

Das BB-Gleichungssystem von Phase I:

$$\text{BB-N-CP}_\beta = \begin{cases} \sigma_0 & \text{falls } \beta = \mathbf{s} \\ \prod \{\text{BB-X-CP}_{\hat{\beta}} \mid \hat{\beta} \in \text{pred}(\beta)\} & \\ \text{sonst} & \end{cases}$$

$$\forall \nu \in \mathbf{V}. \text{BB-X-CP}_\beta(\nu) = \mathcal{E}(\Delta_\beta(\nu))(\text{BB-N-CP}_\beta)$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

**10.3.2**

10.4

10.5

Kap. 11

# Einfache Konstanten für den BB-Fall (3)

## Phase II: Anweisungsebene

Vorberechnete Resultate (aus Phase I):

- ▶  $BB-N-CP^*$ ,  $BB-X-CP^*$ : die **größte Lösung** des Gleichungssystems von Phase I.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

**10.3.2**

10.4

10.5

Kap. 11

# Einfache Konstanten für den BB-Fall (4)

Das EA-Gleichungssystem von Phase II:

$$\text{N-CP}_\iota = \begin{cases} \text{BB-N-CP}_{\text{block}(\iota)}^* & \text{falls } \iota = \text{start}(\text{block}(\iota)) \\ \text{X-CP}_{\text{pred}(\iota)} & \text{sonst (beachte: } |\text{pred}(\iota)| = 1) \end{cases}$$

$$\forall v \in \mathbf{V}. \text{X-CP}_\iota(v) = \begin{cases} \text{BB-X-CP}_{\text{block}(\iota)}^*(v) & \text{falls } \iota = \text{end}(\text{block}(\iota)) \\ \mathcal{E}(\delta_\iota(v))(\text{N-CP}_\iota) & \text{sonst} \end{cases}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

533/863

# Kapitel 10.4

## Schattenvariablen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

**10.4**

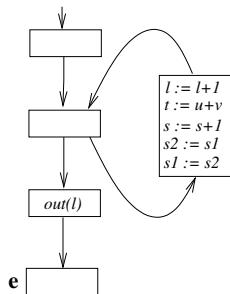
10.5

Kap. 11

# Schattenvariablen

## Anweisung

- ▶  $l := l + 1$  ist lebendig.
- ▶  $t := u + v$  ist tot.
- ▶  $s := s + 1$  sowie  $s1 := s2; s2 := s1$  sind lebendig, aber schattenhaft (faint).



- ▶ **faint**: schwach, kraftlos, ohnmächtig  
     $\rightsquigarrow$  "ein Schatten seiner selbst"  $\rightsquigarrow$  Schattenvariable

# Schattenvariablenanalyse (1)

...für kantenbenannte Einzelanweisungsgraphen.

Lokale Prädikate (assoziiert mit **Einzelanweisungskanten**):

- ▶  $USED_{\varepsilon}(v)$ : Anweisung  $\iota$  von Kante  $\varepsilon$  benutzt  $v$ .
- ▶  $MOD_{\varepsilon}(v)$ : Anweisung  $\iota$  von Kante  $\varepsilon$  modifiziert  $v$ .
- ▶  $REL-USED_{\varepsilon}(v)$ :  $v$  ist eine Variable, die in der Anweisung  $\iota$  von Kante  $\varepsilon$  vorkommt und von dieser Anweisung **“zu leben gezwungen”** wird (z.B. für  $\iota$  Ausgabeanweisung).
- ▶  $ASS-USED_{\varepsilon}(v)$ :  $v$  ist eine in der Zuweisung  $\iota$  von Kante  $\varepsilon$  rechtsseitig vorkommende Variable.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

536/863



# Schattenvariablenanalyse (2)

Das EA-Gleichungssystem:

$$\begin{aligned} \text{FAINT}_n(v) = & \\ & \prod_{m \in \text{succ}(n)} \overline{\text{REL-USED}_{(n,m)}(v)} * \\ & (\text{FAINT}_m(v) + \text{MOD}_{(n,m)}(v)) * \\ & (\text{FAINT}_m(\text{LhsVar}_{(n,m)}) + \overline{\text{ASS-USED}_{(n,m)}(v)}) \end{aligned}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

**10.4**

10.5

Kap. 11

# Schattenvariablenanalyse (3)

...ein Beispiel für ein DFA-Problem, für das eine Formulierung

- ▶ auf (knoten- und kantenbenannten) Einzelanweisungsgraphen offensichtlich ist,
- ▶ auf (knoten- und kantenbenannten) Basisblockgraphen alles andere als ersichtlich, nicht möglich ist.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

**10.4**

10.5

Kap. 11

# Kapitel 10.5

## Fazit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

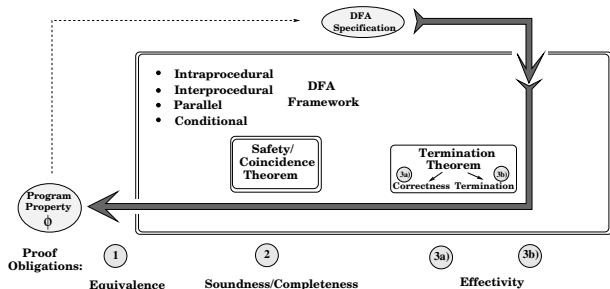
**10.5**

Kap. 11

# Fazit

Alle 4 Flussgraphrepräsentationen sind grundsätzlich **gleichwertig**.

Konzeptuell reicht deshalb die allgemeine **Rahmen-** bzw. **Werkzeugkistensicht**



und das Wissen, dass sie je nach Aufgabe unterschiedlich zweckmäßig sind und **unterschiedlich aufwändige Spezifikations-, Implementierungs- und Beweisverpflichtungen** bewirken.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 10



Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block graphs: Living dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65 - 79, 1998.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

10.5

Kap. 11

# Teil III

## Transformation und Optimalität

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

10.1

10.1.1

10.1.2

10.2

10.2.1

10.2.2

10.2.3

10.3

10.3.1

10.3.2

10.4

**10.5**

Kap. 11

Optimalität = Korrektheit + Vollständigkeit

# Kapitel 11

## Elimination partiell toter Anweisungen



# Kapitel 11.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

**11.1**

11.2

11.3

Kap. 12

Kap. 13

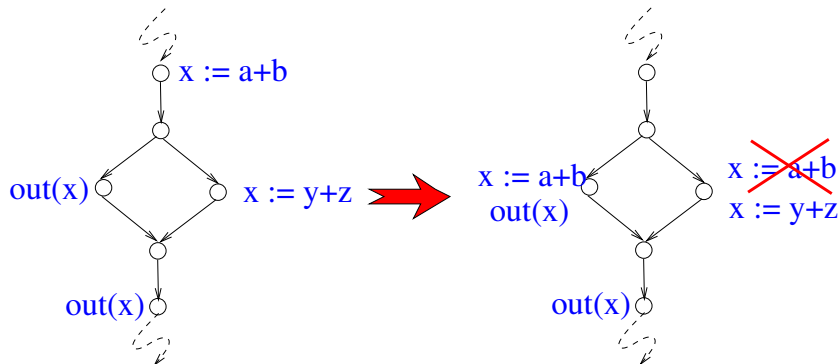
Kap. 14

Kap. 15

Kap. 16

# Elimination partiell toter Anweisungen

Veranschaulichendes Beispiel:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

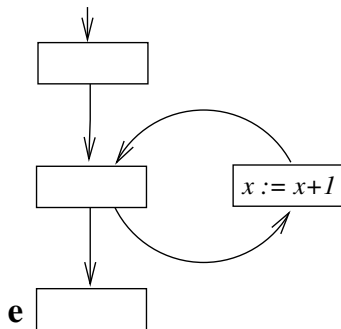
Kap. 15

Kap. 16

546/863

# Schattenhafter Code

Die Anweisung  $x := x + 1$  ist **schattenhaft**, aber nicht **tot**.



# Elimination partiell toten/schattenhaften Codes

Toter und schattenhafter Code induzieren zwei unterschiedliche (Optimierungs-) Transformationen:

- ▶ Elimination partiell toten Codes  
     $\rightsquigarrow$  Partial Dead-Code Elimination (PDCE)
- ▶ Elimination partiell schattenhaften Codes  
     $\rightsquigarrow$  Partial Faint-Code Elimination (PFCE)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

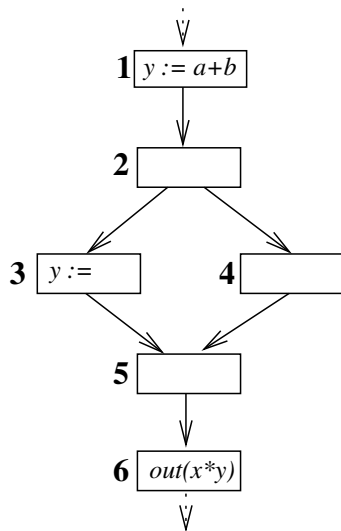
Kap. 14

Kap. 15

Kap. 16

# Elimination partiell toten Codes (1)

Ausgangsprogramm:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

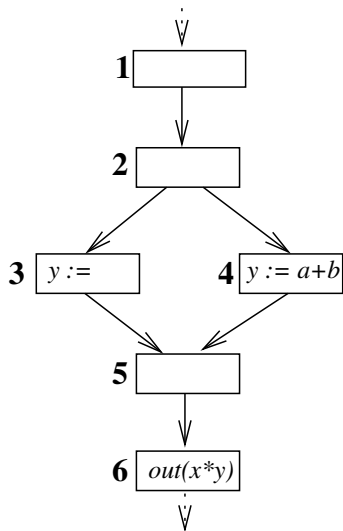
Kap. 14

Kap. 15

Kap. 16

# Elimination partiell toten Codes (2)

Transformiertes/optimiertes Programm:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

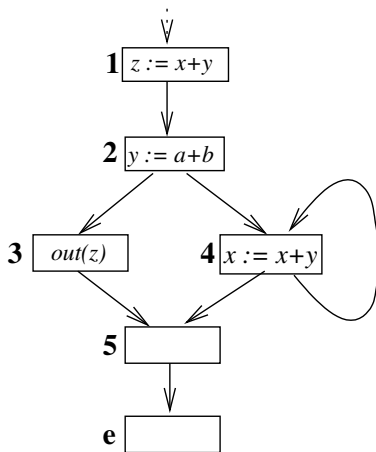
Kap. 14

Kap. 15

Kap. 16

# Elimination partiell schattenhaften Codes (1)

Ausgangsprogramm:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

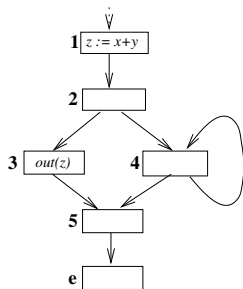
Kap. 16

551/863

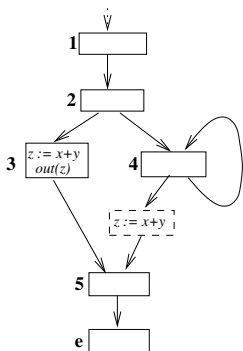
# Elimination partiell schattenhaften Codes (2)

Transformiertes/optimiertes Programm:

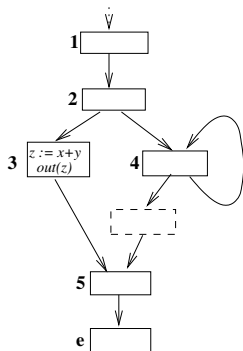
Elimination  
schattenhaften Codes



Anweisungssenkung



Elimination  
toten Codes





# Elimination partiell schattenhaften Codes (3)

Beachte:

- ▶ “Echt” partiell schattenhaften Code gibt es nicht.

Die Elimination schattenhaften Codes

- ▶ kann aber durch die Beseitigung von Codesenkungsblockaden die Elimination weiteren partiell toten Codes ermöglichen.

In diesem Sinne ist

- ▶ Elimination partiell schattenhaften Codes

zu verstehen.

# Kapitel 11.2

## PDCE/PFCE: Transformation und Optimalität

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

**11.2**

11.3

Kap. 12

Kap. 13

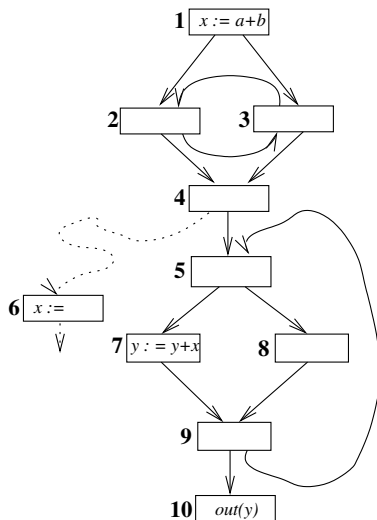
Kap. 14

Kap. 15

Kap. 16

# Kritische Kanten (1)

...be-/verhindern die Transformation:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

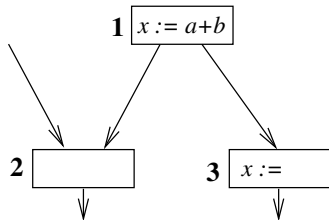
555/863

# Kritische Kanten (2)

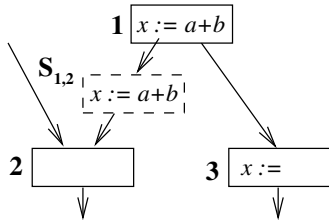
## Definition (11.2.1, Kritische Kanten)

Eine Kante heißt **kritisch** gdw sie von einem Knoten mit mehr als einem Nachfolger zu einem Knoten mit mehr als einem Vorgänger führt.

a)



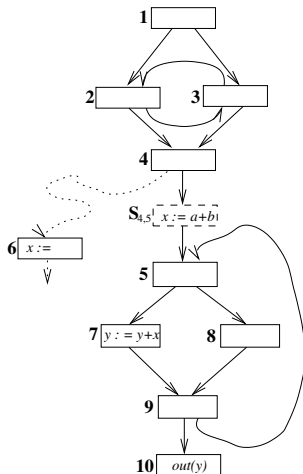
b)



# Kritische Kanten (3)

Für **bestmögliche** Transformationsresultate, insbesondere **kein** Schieben von Anweisungen in Schleifen:

- **Kritische Kanten spalten!**



# Anweisungsmuster

In der Folge bezeichne:

- ▶  $\alpha$  ein sog. **Anweisungsmuster**:  $\alpha \equiv x := t$
- ▶  $\mathcal{AP}$  die **Menge aller Anweisungsmuster**

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

**11.2**

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Senken von Anweisungen

## Definition (11.2.2, Anweisungssenkung)

Eine  $\alpha$ -Anweisungssenkung (assignment sinking) ist eine Programmtransformation, die

- ▶ einige  $\alpha$ -Vorkommen **eliminiert**,
- ▶ neue  $\alpha$ -Vorkommen am Eingang oder Ausgang einiger von einem Basisblock mit einem eliminierten  $\alpha$ -Vorkommen aus erreichbaren Basisblöcke **einsetzt**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

559/863

# Senkungskandidat

...in Basisblöcken:

```
⋮  
y := a+b  
a := c  
x := 3*y  
y := a+b  
x := d
```

```
⋮  
y := a+b  
a := c  
x := 3*y  
y := a+b  
a := d
```



Senkungskandidat

Beachte:

- ▶ Nur das markierte Vorkommen von  $y := a + b$  ist ein Senkungskandidat; die drei anderen Vorkommen von  $y := a + b$  sind lokal blockiert.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

560/863



## Definition (11.2.3, Lokale Barrieren)

Eine  $\alpha$ -Anweisungssenkung wird **blockiert** von einer Anweisung, die

- ▶ einen Operanden von  $t$  modifiziert oder
- ▶ die Variable  $x$  liest oder modifiziert.

# Zulässiges Senken von Anweisungen

## Definition (11.2.4, Zulässige Anweisungssenkung)

Ein  $\alpha$ -Anweisungssenkung ist **zulässig** gdw

1. Die eliminierten  $\alpha$ -Vorkommen werden **ersetzt**, d.h. auf jedem von einem Knoten  $n$  mit eliminiertem  $\alpha$ -Vorkommen ausgehenden Pfad zum Endknoten  $e$  gibt es einen Knoten  $m$ , an dem ein neues  $\alpha$ -Vorkommen **eingesetzt** wird und  $\alpha$  von keiner Anweisung zwischen  $n$  and  $m$  blockiert ist.
2. Jedes neue Vorkommen von  $\alpha$  ist **gerechtfertigt**, d.h., auf jedem vom Startknoten  $s$  aus einen Knoten  $n$  mit neu eingesetztem  $\alpha$ -Vorkommen erreichenden Pfad gibt es einen Knoten  $m$ , an dem ein (ursprüngliches)  $\alpha$ -Vorkommen **eliminiert** worden ist und  $\alpha$  von keiner Anweisung zwischen  $m$  and  $n$  blockiert ist.

# Elimination von Anweisungen

## Definition (11.2.5, Anweisungselimination)

Eine  $\alpha$ -Anweisungselimination (assignment elimination) ist eine Programmtransformation, die einige  $\alpha$ -Vorkommen aus dem Argumentprogramm streicht.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

**11.2**

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

563/863

# Zulässige Elimination von Anweisungen

## Definition (11.2.6, Zulässige Anweisungselimination)

Eine  $\alpha$ -Anweisungselimination ist **zulässig** gdw sie streicht einige

- ▶ tote
- ▶ schattenhafte

$\alpha$ -Vorkommen aus dem Argumentprogramm.

Wir bezeichnen diese beiden **Transformationen** als

- ▶ **Elimination toten Codes**  
     $\rightsquigarrow$  Dead-Code Elimination (**DCE**)
- ▶ **Elimination schattenhaften Codes**  
     $\rightsquigarrow$  Faint-Code Elimination (**FCE**)

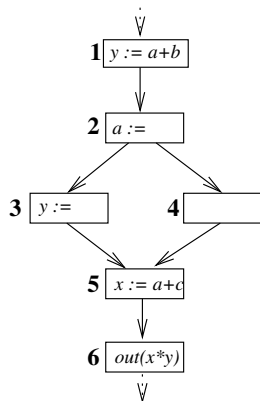
# Effekte zweiter Ordnung

Effekte zweiter Ordnung (engl. second-order effects):

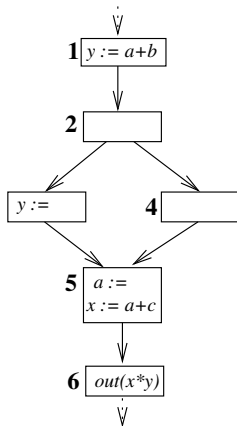
- ▶ Senkungs-Eliminations-Effekte (Zieleffekt)  
     $\rightsquigarrow$  Sinking-Elimination effects (SE)
- ▶ Senkungs-Senkungs-Effekte  
     $\rightsquigarrow$  Sinking-Sinking effects (SS)
- ▶ Eliminations-Senkungs-Effekte  
     $\rightsquigarrow$  Elimination-Sinking effects (ES)
- ▶ Eliminations-Eliminations-Effekte (Zieleffekt)  
     $\rightsquigarrow$  Elimination-Elimination effects (EE)

# Beispiel eines Senkungs-Senkungs-Effekts

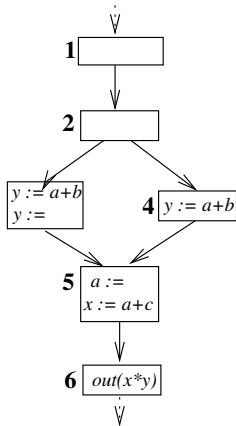
Ausgangsprogramm



1. Senkung

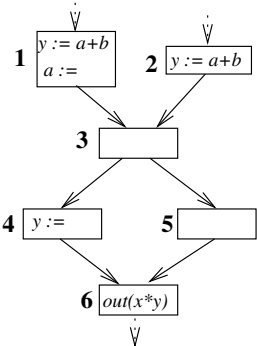


2. Senkung

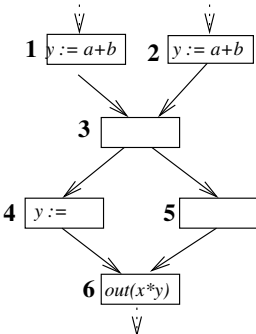


# Beispiel eines Eliminations-Senkungs-Effekt

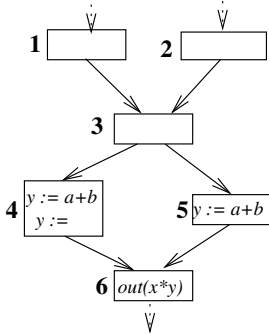
Ausgangsprogramm



Elimination

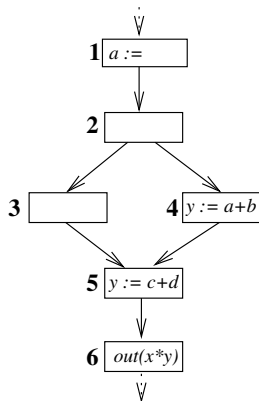


Senkung

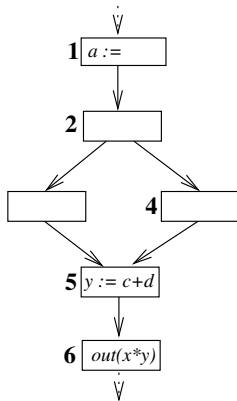


# Beispiel eines Eliminations-Eliminations-Effekt

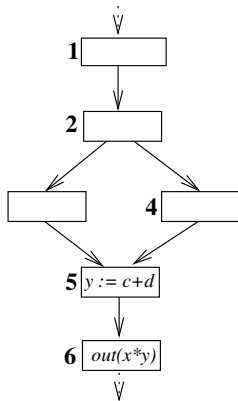
Ausgangsprogramm



1. Elimination



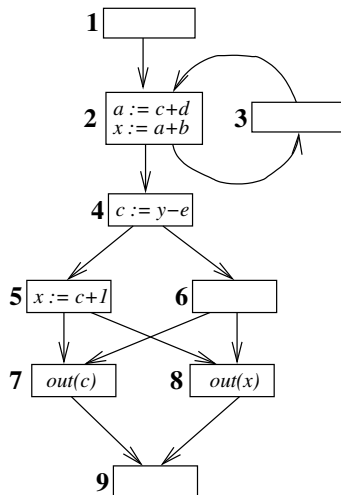
2. Elimination





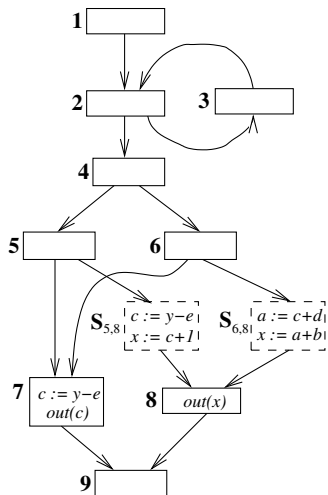
# Kombinationswirkung von Effekten zweiter Ordnung (1)

Ausgangsprogramm:



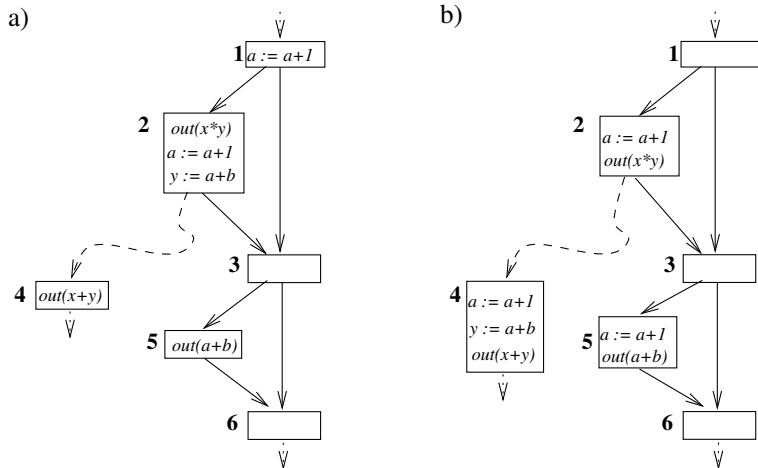
# Kombinationswirkung von Effekten zweiter Ordnung (2)

Transformiertes/optimiertes Programm:



# Kombinationswirkung von Effekten zweiter Ordnung (3)

Im allgemeinen m2n-Senkungs-Eliminationswirkungen:



# Die PDCE/PDFE-Transformationen

## Definition (11.2.7, PDCE/PDFE-Transformationen)

Die Elimination partiell toten/schattenhaften Codes (partial dead (faint) code elimination) PDCE/PFCE ist eine beliebige Abfolge zulässiger

- ▶ Anweisungssenkungen und
- ▶ Anweisungseliminierungen von
  - ▶ toten
  - ▶ schattenhaften

Anweisungen.

# Bezeichnungen und Schreibweisen (1)

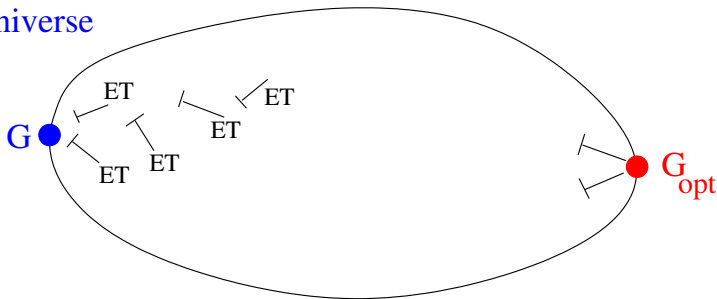
- ▶  $G \vdash_{PDCE} G'$  bzw.  $G \vdash_{PFCE} G'$ :  
 $G'$  resultiert aus  $G$  durch Anwendung einer zulässigen Anweisungssenkungs- oder Eliminationstransformation (tot bzw. schattenhaft).
- ▶  $\tau \in \{PDCE, PFCE\}$ :  
Bezeichner für PDCE bzw. PFCE.
- ▶  $\mathcal{G}_\tau =_{df} \{ G' \mid G \vdash_\tau^* G' \}$ :  
Das aus  $G$  durch sukzessive Anwendung von PDCE- bzw. PFCE-Elementartransformationen aufgespannte Universum.

# Bezeichnungen und Schreibweisen (2)

Veranschaulichung des Universums:

- ▶ ET steht für Elementar-Transformation, d.h. Anweisungssenkung und Anweisungselimination.

Universe



# Vergleichsrelation “besser” für Programme

## Definition (11.2.8, besser)

Seien  $G', G'' \in \mathcal{G}_\tau$ . Dann heißt  $G'$  besser als  $G''$ , in Zeichen  $G'' \sqsubseteq G'$ , gdw

$$\forall p \in \mathbf{P}[\mathbf{s}, \mathbf{e}] \forall \alpha \in \mathcal{AP}. \alpha\#(p_{G'}) \leq \alpha\#(p_{G''})$$

wobei  $\alpha\#(p_{G'})$  und  $\alpha\#(p_{G''})$  jeweils die Anzahl der  $\alpha$ -Vorkommen auf  $p$  in  $G'$  bzw.  $G''$  bezeichnen.

Beachte:

- ▶ **Anweisungssenkungen** und **-eliminationen** erhalten die Verzweigungsstruktur eines Programms  $G$ . Die einem Pfad in  $G$  entsprechenden Pfade in  $G'$  und  $G''$  können deshalb einfach identifiziert werden.

# Eigenschaften der Relation “besser”

## Lemma (11.2.9)

Die Vergleichsrelation  $\preceq$  ist eine

- ▶ *Quasiordnung* (d.h. *reflexiv* und *transitiv*, aber nicht *antisymmetrisch*).



# Der Optimalitätsbegriff

## Definition (11.2.10, PDCE/PFCE-Optimalität)

Ein Programm  $G^* \in \mathcal{G}_T$  ist **optimal** gdw  $G^*$  besser ist als jedes andere Programm aus  $\mathcal{G}_T$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

**11.2**

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

577/863

# Monotonie und Dominanz

Sei

- ▶  $\vec{\sqsubseteq}_\tau =_{df} (\vec{\sqsubseteq} \cap \vdash_\tau)^*$
- ▶  $\mathcal{F}_\tau \subseteq \{f \mid f : \mathcal{G}_\tau \rightarrow \mathcal{G}_\tau\}$  eine endliche Familie von Funktionen mit
  1. Monotonie:  
 $\forall G', G'' \in \mathcal{G}_\tau \forall f \in \mathcal{F}_\tau.$   
 $G' \vec{\sqsubseteq}_\tau G'' \Rightarrow f(G') \vec{\sqsubseteq}_\tau f(G'')$
  2. Dominanz:  
 $\forall G', G'' \in \mathcal{G}_\tau. G' \vdash_\tau G'' \Rightarrow \exists f \in \mathcal{F}_\tau. G'' \vec{\sqsubseteq}_\tau f(G')$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

578/863

## Theorem (11.2.11, Existenz optimalen Programms)

$\mathcal{G}_\tau$  besitzt ein *optimales* Element (bezüglich  $\preceq$ ), das von jeder Folge von Funktionsanwendungen berechnet wird, die alle Elemente aus  $\mathcal{F}_\tau$  'hinreichend' oft enthält.

**Beweis** mithilfe von Monotonie, Dominanz und Fixpunkttheorem 9.2.4.

# Anwendung auf PDCE und PFCE

- ▶ PDCE und PFCE erfüllen die Voraussetzungen von Optimalitätstheorem 11.2.11.
- ▶ Das optimale Programm in  $\mathcal{G}_T$  bezüglich PDCE und PFCE ist bis auf irrelevante Umreihungen von Anweisungen in Basisblöcken eindeutig bestimmt.

Insgesamt ergibt sich daraus die

- ▶ Korrektheit und Optimalität

von PDCE- und PDFE-Transformation.

# Zweiter Korrektheits- und Optimalitätsbeweis

## PDCE/PFCE-Transformationsidee:

Konzeptuell können wir die **Elimination partiell toter/schattenhafter Anweisungen (PDCE/PFCE)** in folgender Weise verstehen:

- ▶  $PDCE = (AS + DCE)^*$
- ▶  $PFCE = (AS + FCE)^*$

# Bezeichnungen (1)

In der Folge bezeichnen wir die aus vorstehender Transformationsidee **abgeleiteten Algorithmen** für die **Elimination partiell**

- ▶ **toter** und
- ▶ **schattenhafter**

**Anweisungen** mit

- ▶ **pdce** und
- ▶ **pfce**.

## Bezeichnungen (2)

Wir bezeichnen weiters die aus einem Programm  $G$  durch Anwendung von  $pdce$  und  $pfce$  resultierenden Programme mit

- ▶  $G_{pdce}$  und
- ▶  $G_{pfce}$

und die von den Elementartransformationen von  $pdce$  und  $pfce$  aufgespannten Universen für  $G$  mit

- ▶  $\mathcal{G}_{PDCE}$  und
- ▶  $\mathcal{G}_{PFCE}$ .

# Veranschaulichung

PDCE/PFCE-Ableitungsrelation  $\vdash$ :

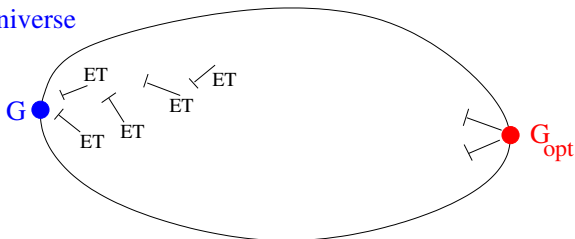
- ▶ PDCE:  $G \vdash_{AS,DCE} G'$  (d.h.  $ET =_{df} \{AS, DCE\}$ )
- ▶ PFCE:  $G \vdash_{AS,FCE} G'$  (d.h.  $ET =_{df} \{AS, FCE\}$ )

Korrektheit und Optimalität von  $pdce/pfce$  folgen aus:

Theorem (11.2.14, Konfluenz und Terminierung)

Die PDCE/PFCE-Ableitungsrelationen  $\vdash_{AS,DCE}$  und  $\vdash_{AS,FCE}$  sind *konfluent* und *terminierend*.

Universe





# PDCE/PDFE-Korrektheit und -Optimalität

## Theorem (11.2.12, Korrektheit)

1.  $G_{pdce} \in \mathcal{G}_{PDCE}$
2.  $G_{pfce} \in \mathcal{G}_{PFCE}$

## Theorem (11.2.13, Optimalität)

1.  $G_{pdce}$  ist optimal in  $\mathcal{G}_{PDCE}$ .
2.  $G_{pfce}$  ist optimal in  $\mathcal{G}_{PFCE}$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

**11.2**

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Kapitel 11.3

## Implementierung von PDCE/PFCE

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

**11.3**

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Vorbereitungen zur DFA-Spezifikation

Wir benötigen 6 Hilfsprädikate für die Spezifikation der benötigten **lokalen abstrakten Semantiken** und der darauf aufbauenden DFAs.

Im einzelnen folgende 6 **lokale Prädikate**:

- ▶ USED, REL-USED, ASS-USED und MOD
- ▶ LOC-DELAYED und LOC-BLOCKED

Darauf aufbauende DFAs:

- ▶ **Elimination toten Codes**
  - ↔ Dead-Code Elimination (**DCE**)
- ▶ **Elimination schattenhaften Codes**
  - ↔ Faint-Code Elimination (**FCE**)
- ▶ **Anweisungssenkung**
  - ↔ Assignment Sinking (**AS** ↔ **Delayability**)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

587/863

# Bedeutung der lokalen DCE/FCE-Prädikate

Zur Bedeutung der vier lokalen DCE/FCE-Prädikate:

- ▶  $USED_\ell(x)$ :  $x$  wird rechtsseitig in Anweisung  $\ell$  benutzt.
- ▶  $REL-USED_\ell(x)$ :  $x$  wird rechtsseitig in der “relevanten”, d.h. “zum Leben zwingenden” Anweisung  $\ell$  benutzt.
- ▶  $ASS-USED_\ell(x)$ :  $x$  wird rechtsseitig in der Zuweisung  $\ell$  benutzt.
- ▶  $MOD_\ell(x)$ :  $x$  wird linksseitig in der Anweisung  $\ell$  benutzt.

# Die DCE-Analyse

## Die Analyse toter Variablen (DCE):

$$\text{N-DEAD}_\iota = \overline{\text{USED}_\iota} * (\text{X-DEAD}_\iota + \text{MOD}_\iota)$$

$$\text{X-DEAD}_\iota = \prod_{\hat{\iota} \in \text{succ}(\iota)} \text{N-DEAD}_{\hat{\iota}}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

**11.3**

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

589/863

# Die FCE-Analyse

## Die Analyse schattenhafter Variablen (FCE): (Simultan für alle Variablen $x$ )

$$\text{N-FAINT}_\iota(x) = \overline{\text{REL-USED}_\iota(x)} * \\ (\text{X-FAINT}_\iota(x) + \text{MOD}_\iota(x)) * \\ (\text{X-FAINT}_\iota(\text{LhsVar}_\iota) + \overline{\text{ASS-USED}_\iota(x)})$$

$$\text{X-FAINT}_\iota(x) = \prod_{\hat{\iota} \in \text{succ}(\iota)} \text{N-FAINT}_{\hat{\iota}}(x)$$

wobei  $\text{LhsVar}_\iota$  die linksseitige Variable von Zuweisung  $\iota$  bezeichnet.

# Bedeutung der lokalen AS-Prädikate

Zur Bedeutung der zwei lokalen AS-Prädikate:

- ▶  $\text{LOC-DELAYED}_n(\alpha)$ : Es gibt einen  $\alpha$ -Anweisungsenkungs-Kandidaten (sinking candidate) in  $n$ .
- ▶  $\text{LOC-BLOCKED}_n(\alpha)$ : Die Senkung von  $\alpha$  ist durch eine Anweisung an  $n$  blockiert.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

591/863

# Die AS-Analyse

## Das Anweisungssenkungs-Gleichungssystem:

$$\text{N-DELAYED}_n = \begin{cases} \text{falsch} & \text{if } n = \mathbf{s} \\ \prod_{m \in \text{pred}(n)} \text{X-DELAYED}_m & \text{otherwise} \end{cases}$$

$$\text{X-DELAYED}_n = \text{LOC-DELAYED}_n + \text{N-DELAYED}_n * \overline{\text{LOC-BLOCKED}_n}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

11.3

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

592/863



# Einsetzungspunkte gesenkter Anweisungen

Die sich aus der AS-Analyse ergebenden Einsetzungspunkte:



$$\text{N-INSERT}_n \stackrel{df}{=} \text{N-DELAYED}_n^* * \text{LOC-BLOCKED}_n$$

$$\text{X-INSERT}_n \stackrel{df}{=} \text{X-DELAYED}_n^* * \sum_{m \in \text{succ}(n)} \overline{\text{N-DELAYED}_m^*}$$

wobei  $\text{N-DELAYED}_n^*$  und  $\text{X-DELAYED}_n^*$  die größten Lösungen des Anweisungssenkungs-Gleichungssystems bezeichnen.

**Beachte:** Die Berechnung der Einsetzungspunkte erfordert keine DFA!

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (1)

-  Ras Bodik, Rajiv Gupta. *Partial Dead Code Elimination using Slicing Transformations*. In Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI'97), ACM SIGPLAN Notices 32(6):159-170, 1997.
-  L. Feigen, D. Klappholz, R. Casazza, X. Xue. *The Revival Transformation*. In Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94), 1994.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

11.1

11.2

**11.3**

Kap. 12

Kap. 13




Kap. 14

Kap. 15

Kap. 16

594/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 11 (2)

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Partial Dead Code Elimination*. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94), ACM SIGPLAN Notices 29(6):147-158, 1994.
-  Ronald J. Mintz, Gerald A. Fisher, Micha Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SoCC'79), ACM SIGPLAN Notices 14(8):226-234, 1979.
-  Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.

# Kapitel 12

## Elimination partiell redundanter Anweisungen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

**Kap. 12**

12.1

12.2

12.3

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Kapitel 12.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

**12.1**

12.2

12.3

Kap. 13

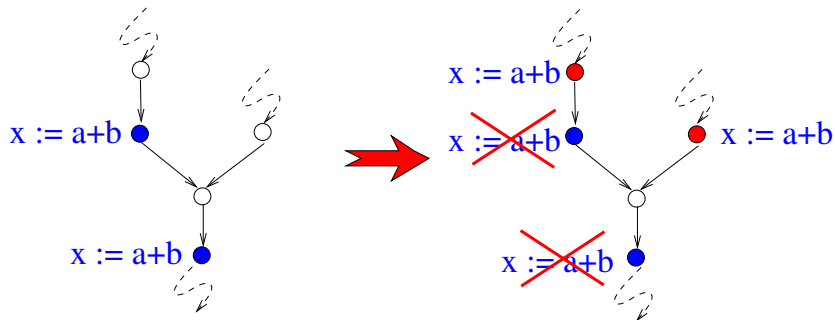
Kap. 14

Kap. 15

Kap. 16

# Elimination partiell redundanter Anweisungen

Veranschaulichendes Beispiel:



# Transformationsidee

Konzeptuell können wir die **Elimination partiell redundanter Anweisungen (PRAE)** in folgender Weise verstehen:

- ▶  $PRAE = (AH + RAE)^*$

Analog zu den **PDCE/PFCE**-Transformationen **pdce** und **pfce** gilt auch für die **PRAE**-Transformation **prae** ein

- ▶ **Korrektheits-** und
- ▶ **Optimalitätsresultat.**

# Veranschaulichung

PRAE-Ableitungsrelation  $\vdash$ :

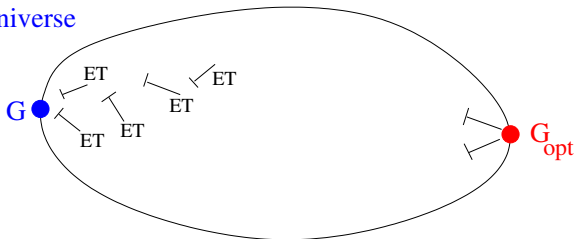
► PRAE:  $G \vdash_{AH,RAE} G'$  (d.h.  $ET =_{df} \{AH,RAE\}$ )

Korrektheit und Optimalität von *prae* folgen aus:

Theorem (12.1.1, Konfluenz und Terminierung)

Die PRAE-Ableitungsrelation  $\vdash_{AH,RAE}$  ist *konfluent* und *terminierend*.

Universe





# PRAE-Korrektheit und -Optimalität

Theorem (12.1.2, Korrektheit)

$$G_{prae} \in \mathcal{G}_{PRAE}$$

Theorem (12.1.3, Optimalität)

$G_{prae}$  ist optimal in  $\mathcal{G}_{PRAE}$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

**12.1**

12.2

12.3

Kap. 13

Kap. 14

Kap. 15

Kap. 16

# Kapitel 12.2

## EAM: Einheitliche PREE/PRAE-Behandlung

# Grundtransformationen: PREE und PRAE

## Zwei Grundtransformationen zur Redundanzelimination:

- ▶ **Elimination partiell redundanter Ausdrücke**
  - ↪ Partially Redundant Expression Elimination (**PREE**)
  - ↪ Expression Motion (**EM**)
- ▶ **Elimination partiell redundanter Anweisungen**
  - ↪ Partially Redundant Assignment Elimination (**PRAE**)
  - ↪ Assignment Motion (**AM**)

# Kombinierte PRE/AE-Transformation: EAM

## Kombinierte Transformation zur Redundanzelimination:

- ▶ Elimination partiell redundanter Ausdrücke und Anweisungen
  - ↪ Partially Redundant Expression and Assignment Elimination (**PREAE**)
  - ↪ Expression and Assignment Motion (**EAM**)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

12.1

**12.2**

12.3

Kap. 13

Kap. 14

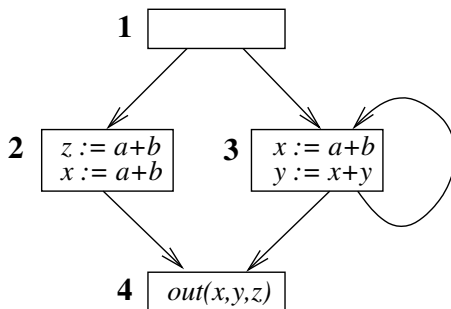
Kap. 15

Kap. 16

604/863

# In der Folge

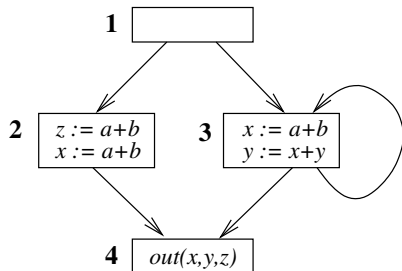
...illustrieren wir die **unterschiedlichen Effekte** dieser Transformationen anhand eines **gemeinsamen Beispiels**:



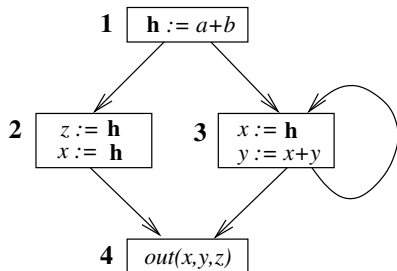
# Elimination partiell redundanter Ausdrücke

Der PREE-Effekt auf das laufende Beispiel:

a)



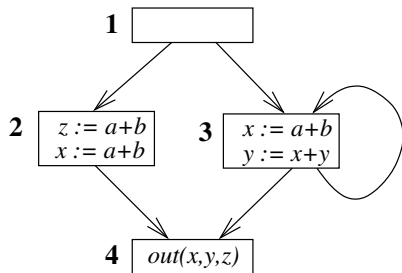
b)



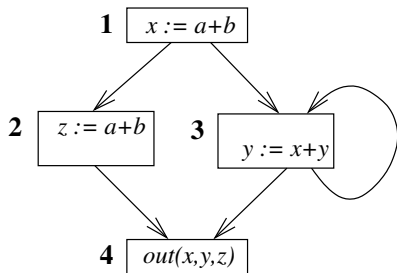
# Elimination partiell redundanter Anweisungen

Der PRAE-Effekt auf das laufende Beispiel:

a)



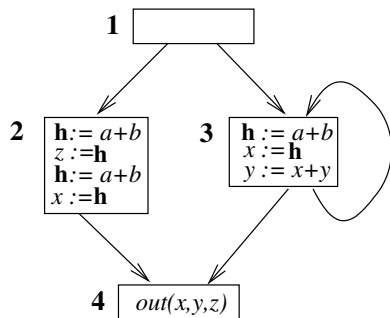
b)



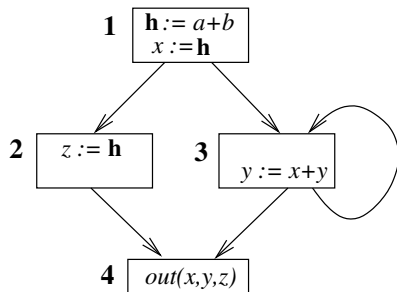
# Kombinierte Elimination partiell redundanter Ausdrücke und Anweisungen

Der EAM(=PREAE)-Effekt auf das laufende Beispiel:

a)



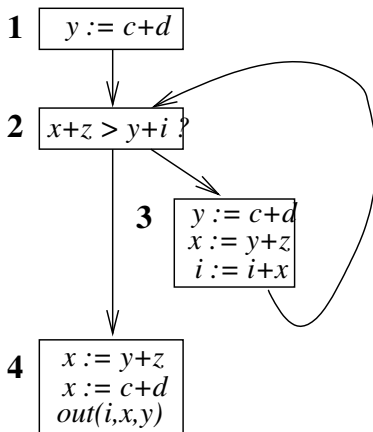
b)





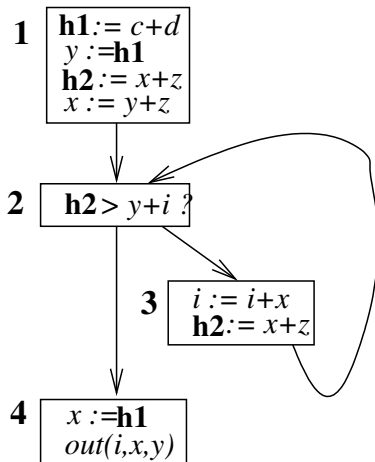
# EAM anhand eines größeren Beispiels (1)

Ausgangsprogramm:



# EAM anhand eines größeren Beispiels (2)

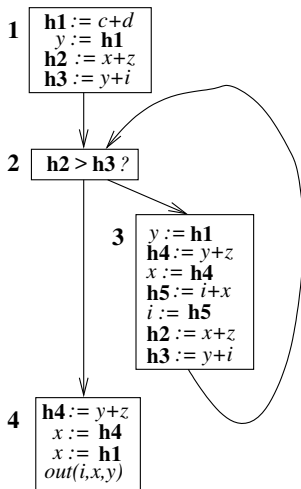
Effekt der EAM-Optimierung:



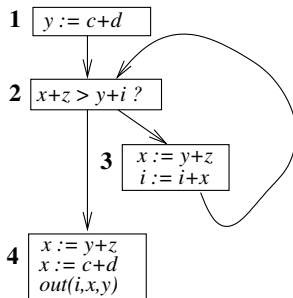
# PREE- und PRAE-Effekte zum Vergleich

Die (schwächeren) Effekte von PREE (a) und PRAE (b):

a)



b)



# Kapitel 12.3

## EAM: Transformation und Optimalität

# Der EAM-Algorithmus eam

eam: Ein dreistufiges Verfahren

## ▶ Präprozess

Ersetze jedes Vorkommen einer Anweisung  $x := t$  durch die Anweisungssequenz  $h_t := t; x := h_t$ .

## ▶ Hauptprozess

Wende die Transformationen

### ▶ Heben von Anweisungen

↔ Assignment Hoisting (AH)

### ▶ Eliminieren (total) redundanter Anweisungen

↔ (Totally) Redundant Assignment Elimination (RAE)

wiederholt so lange an bis Stabilität eintritt.

## ▶ Postprozess

Aufräumen isolierter Initialisierungen.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

12.1

12.2

12.3

Kap. 13

Kap. 14

Kap. 15

Kap. 16

613/863

# Wichtig

Der **Präprozess** bewirkt

- ▶ dass der 3-stufige **EAM-Algorithmus** die Effekte von **PREE** und **PRAE** einheitlich erfasst und abdeckt!

Dabei gilt:

- ▶ “Das Ganze ist mehr als die Summe seiner Teile”:

$$\text{EAM} > \text{PREE} + \text{PRAE}$$

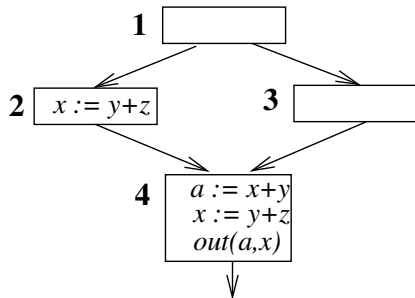
# Effekte zweiter Ordnung für (E)AM

Effekte zweiter Ordnung (engl. *second order effects*) im (E)AM-Fall:

- ▶ Hebungs-Hebungs-Effekte  
     $\rightsquigarrow$  Hoisting-Hoisting effects (HH)
- ▶ Hebungs-Eliminations-Effekte (Zieleffekt)  
     $\rightsquigarrow$  Hoisting-Elimination effects (HE)
- ▶ Eliminations-Hebungs-Effekte  
     $\rightsquigarrow$  Elimination-Hoisting effects (EH)
- ▶ Eliminations-Eliminations-Effekte (Zieleffekt)  
     $\rightsquigarrow$  Elimination-Elimination effects (EE)

# Veranschaulichung von Effekten 2. Ordnung

Ausgangsprogramm:

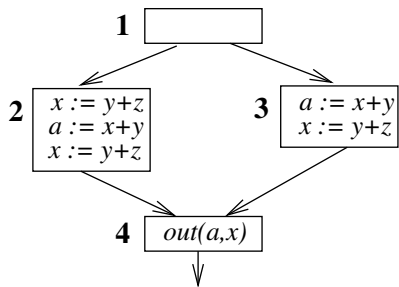




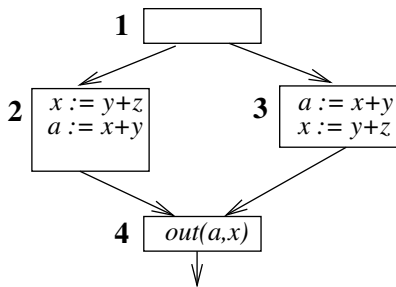
# Veranschaulichung von Effekten 2. Ordnung

Transformiertes/optimiertes Programm:

a)

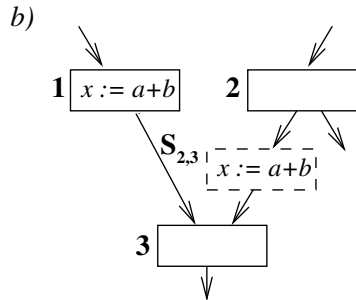
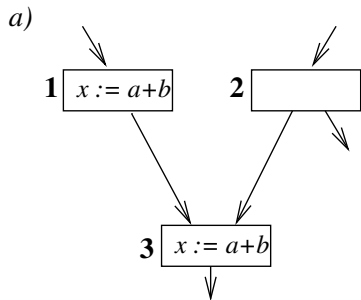


b)



# Wie für PDCE/PFCE

Spalten kritischer Kanten:



# Analog zu PDCE/PFCE

Hebungskandidat in Basisblöcken:

```
x := d
y := a+b
x := 3*y
a := c
y := a+b
⋮
⋮
⋮
```

```
a := d
y := a+b
x := 3*y
a := c
y := a+b
⋮
⋮
⋮
```



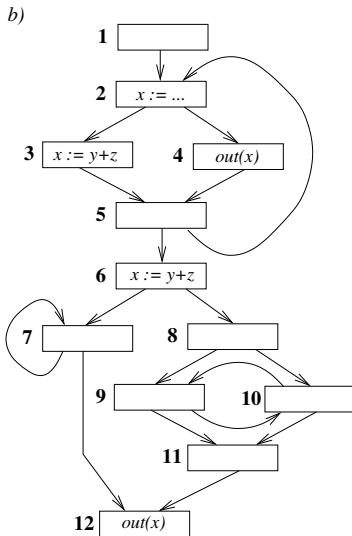
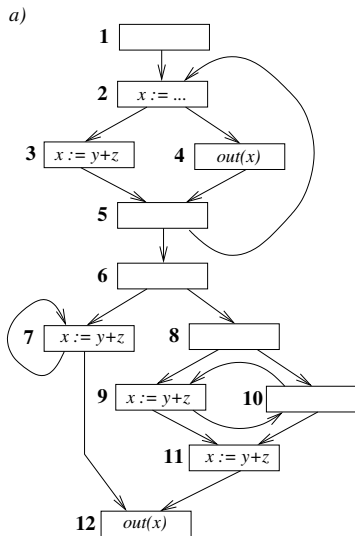
Hebungskandidat

Beachte:

- ▶ Nur das markierte Vorkommen von  $y := a + b$  ist ein Hebungskandidat; die drei anderen Vorkommen von  $y := a + b$  sind lokal blockiert.

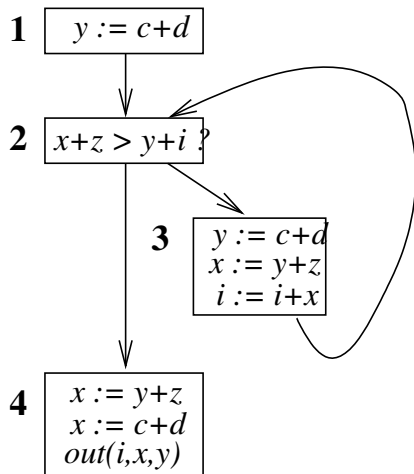
# EAM für schleifenbehaftete Programme

Kein Schieben von Anweisungen in Schleifen:



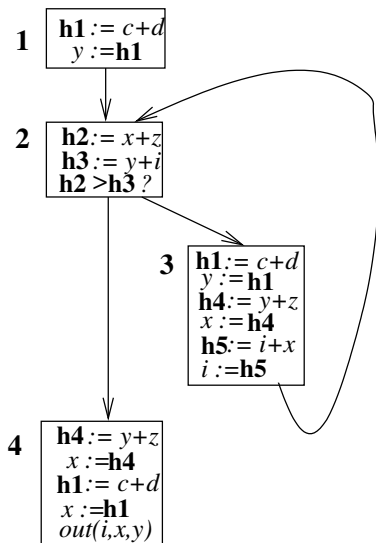
# Die EAM-Transformation im Detail (1)

Ausgangsprogramm:



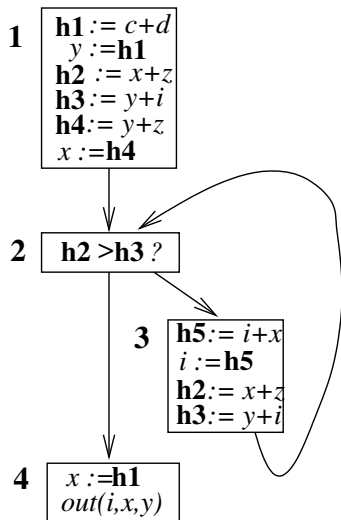
# Die EAM-Transformation im Detail (2)

Effekt des Präprozesses:



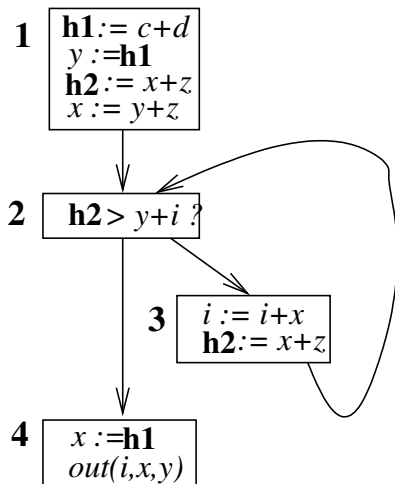
# Die EAM-Transformation im Detail (3)

Effekt des Hauptprozesses:



# Die EAM-Transformation im Detail (4)

Effekt des Postprozesses und damit des EAM-Gesamteffekts:





# EAM-Hauptresultate

Analog zu den PRAE/PDCE/PFCE-Transformationen gelten auch für die EAM-Transformation

- ▶ Korrektheits- und
- ▶ Optimalitätsresultate.

Anders als für die PRAE/PDCE/PFCE-Transformationen gilt:

- ▶ EAM-Optimalität zerfällt in Aussagen über
  - ▶ Ausdrücke, Anweisungen und Hilfsvariablen(anzahl)
  - ▶ lokale und globale Optimalität.

Bezeichnungen für die folgenden Korrektheits- und Optimalitätstheoreme:

- ▶ Sei  $G$  ein Programm,  $G_{eam}$  das durch Anwendung von  $eam$  auf  $G$  entstehende Programm und  $\mathcal{G}_{EAM}$  das durch EAM-Elementartransformationen aufgespannte Universum.

## Theorem (12.2.1, Korrektheit)

$$G_{eam} \in \mathcal{G}_{EAM}$$

# EAM-Optimalität: Ausdrucksoptimalität

## Theorem (12.2.2, Ausdrucksoptimalität)

$G_{eam}$  is *ausdrucksoptimal* in  $\mathcal{G}_{EAM}$ , d.h., während seiner Ausführung werden *höchstens so viele Ausdrücke ausgewertet* wie in jedem anderen Programm, das durch Anwendung von *PREE-* und *PRAE-Transformationen* entstehen kann.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

12.1

12.2

12.3

Kap. 13

Kap. 14

Kap. 15

Kap. 16

627/863

# EAM-Optimalität: Anweisungsoptimalität

## Theorem (12.2.3, Relative Anweisungsoptimalität)

$G_{eam}$  ist *relativ anweisungsoptimal* in  $\mathcal{G}_{EAM}$ , d.h. es ist nicht möglich, die Zahl der von  $G_{eam}$  zur Laufzeit ausgeführten Anweisungen durch *PREE*- und *PRAE*-Transformationen zu verringern.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

12.1

12.2

**12.3**

Kap. 13

Kap. 14

Kap. 15

Kap. 16

628/863

# EAM-Optimalität: Hilfsvariablenoptimalität

## Theorem (12.2.4, Relative Hilfsvariablenoptimalität)

$G_{eam}$  ist *relative hilfsvariablenoptimal* in  $\mathcal{G}_{EAM}$ , d.h. es ist nicht möglich, die Zahl der Zuweisungen an Hilfsvariablen oder die Länge der Lebenszeiten der Hilfsvariablen in  $G_{eam}$  durch Anweisungssenkungen (assignment sinkings) zu verringern.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

12.1

12.2

12.3

Kap. 13

Kap. 14

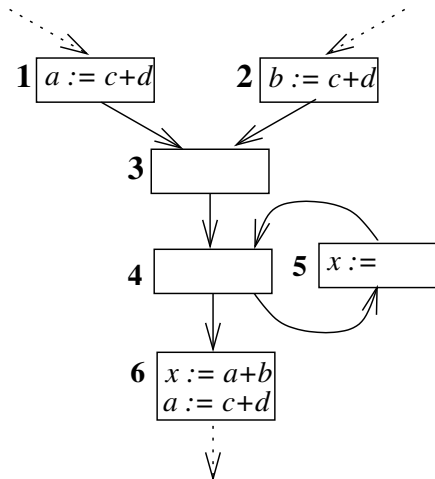
Kap. 15

Kap. 16

629/863

# Warum nur relative A/HV-Optimalität?

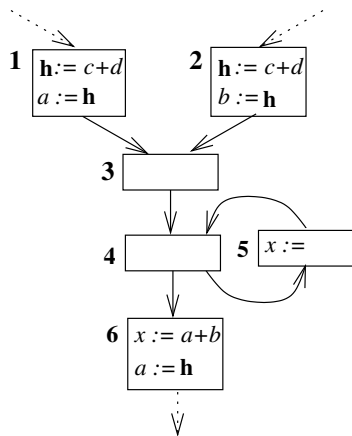
Betrachte dazu folgendes Programm:



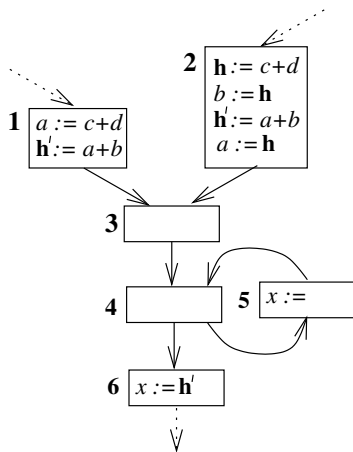
# Zur relativen A/HV-Optimalität

...und folgende zwei unvergleichbare Transformationsresultate:

a)



b)

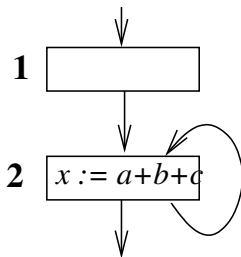


⇒ Relative A/HV-Optimalität ist das Beste, was möglich ist!

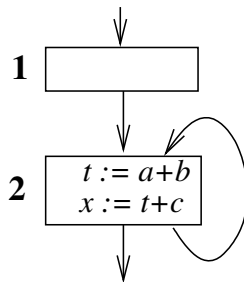
# Weitere PREE/PRAE-Phänomene (1)

PREE/PRAE-Phänomene im Zusammenhang mit 3-Adresscode und Nicht-3-Adresscode:

a)



b)

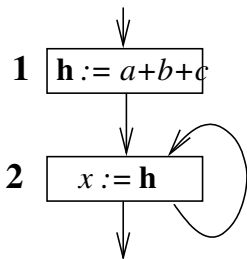




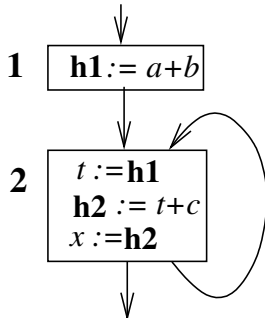
# Weitere PREE/PRAE-Phänomene (2)

Der Effekt von PREE auf die Programme aus Abb. (a) und (b):

a)



b)

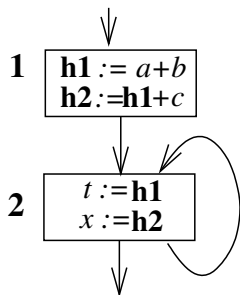


# Weitere PREE/PRAE-Phänomene (3)

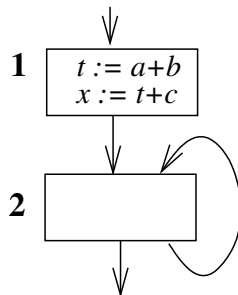
Die **Effekte** von

- ▶ PREE gefolgt von **Konstantenpropagierung (constant propagation (CP))**: Abb. (a)
- ▶ **EAM**: Abb. (b)




a)






b)





# Ergänzende, weiterführende und vertiefende Leseempfehlungen für Kapitel 12 (1)

-  D. M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.
-  D. M. Dhamdhere. *A usually linear Algorithm for Register Assignment using Edge Placement of Load and Store Instructions*. Journal of Computer Languages 15(2):83-94, 1990.
-  D. M. Dhamdhere. *Practical Adaptation of the Global Optimization Algorithm of Morel and Renvoise*. ACM Transactions on Programming Languages and Systems 13(2):291-294, 1991. Technical Correspondence.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (2)

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Lazy Code Motion*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):224-234, 1992.
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *The Power of Assignment Motion*. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95), ACM SIGPLAN Notices 30(6):233-245, 1995.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 12 (3)

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Retrospective: Lazy Code Motion*. In "20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation (1979 - 1999): A Selection", ACM SIGPLAN Notices 39(4):460-461&462-472, 2004.
-  Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.

# Kapitel 13

## Kombination von PRAE und PDCE

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

**Kap. 13**

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Motivation

## Erinnerung:

Konzeptuell können wir **PRAE** und **PDCE** wie folgt verstehen:

- ▶  $PRAE = (AH + RAE)^*$
- ▶  $PDCE = (AS + DCE)^*$

Das legt nahe auch die “**Summe**” aus **PRAE** und **PDCE** zu betrachten:

- ▶  $AP = (AH + RAE + AS + DCE)^*$

**AP** steht dabei für **Assignment Placement**.

# Individuelle PRAE/PDCE-Opt.-Ergebnisse

PRAE- bzw. PDCE-Ableitungsrelation  $\vdash$ :

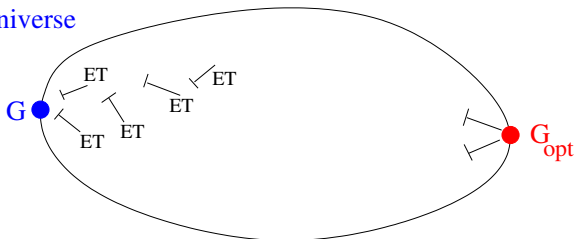
- ▶ PRAE:  $G \vdash_{AH,RAE} G'$  (d.h.  $ET =_{df} \{AH,RAE\}$ )
- ▶ PDCE:  $G \vdash_{AS,DCE} G'$  (d.h.  $ET =_{df} \{AS,DCE\}$ )

Wir haben gezeigt (s. Kap. 11 und 12):

## Theorem (13.1, PRAE-/PDCE-Optimalität)

Die *PRAE*- und *PDCE*-spezifischen Ableitungsrelationen  $\vdash_{ET}$  sind *konfluent* und *terminierend* (und somit *optimal*).

Universe





# Kombination von PRAE und PDCE: AP

Betrachte nun:

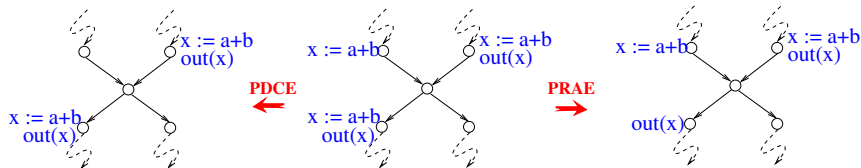
- ▶ Assignment Placement AP

$$AP = (AH + RAE + AS + DCE)^*$$

Erwartung:

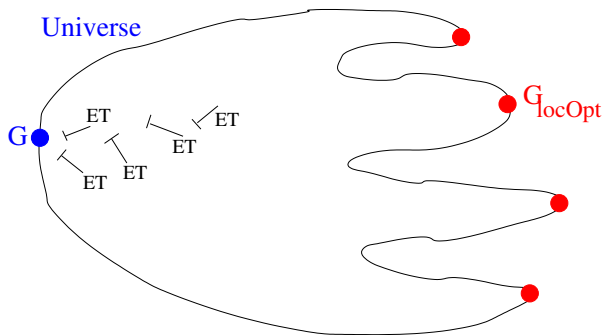
- ▶ AP sollte mächtiger sein als PRAE und PDCE individuell!

Das gilt in der Tat. Aber:



# AP: Verlust globaler Optimalität

Konfluenz und damit **globale Optimalität** sind verloren!



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

**Kap. 13**

Kap. 14

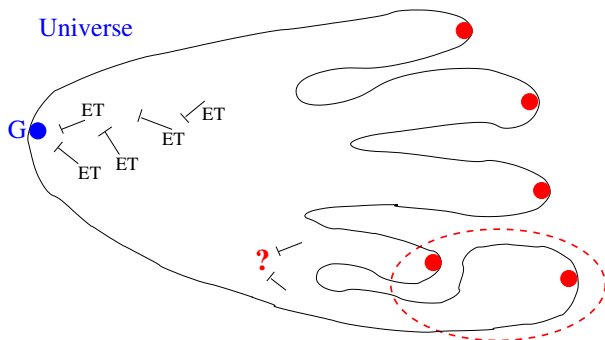
Kap. 15

Kap. 16

Kap. 17

# AP: Verlust lokaler Optimalität

In speziellen Szenarien geht sogar lokale Optimalität verloren:



- ▶ Knoop, J., and Mehofer, E. **Distribution Assignment Placement: Effective Optimization of Redistribution Costs.** *IEEE Transactions on Parallel and Distributed Systems* 13(6):628-647, 2002.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 13

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.
-  Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs.* IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

**Kap. 13**

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Kapitel 14

## Konstantenfaltung auf dem Wertegraphen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

**Kap. 14**

14.1

14.2

14.3

Kap. 15

Kap. 16

## Konstantenfaltung und -ausbreitung

- ▶ Hintergrund und Motivation
- ▶ Der VG-Konstantenfaltungsalgorithmus
- ▶ Der PVG-Konstantenfaltungsalgorithmus

# Kapitel 14.1

## Motivation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

**14.1**

14.2

14.3

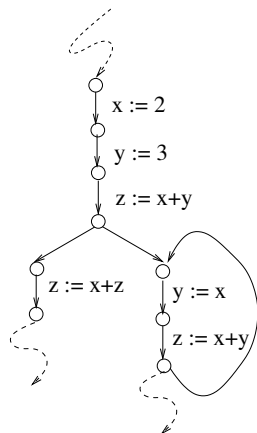
Kap. 15

Kap. 16

# Konstantenfaltung und -ausbreitung

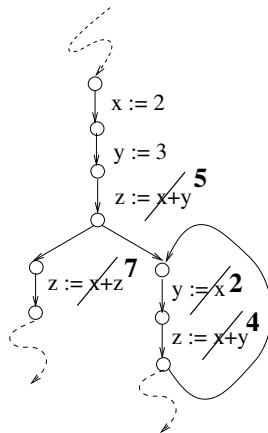
## Veranschaulichendes Beispiel

a)



Original program

b)



After simple constant propagation



# Ursprung und Entwicklung

...von Algorithmen zu Konstantenfaltung und -ausbreitung.

Prägend:

- ▶ Gary A. Kildalls Algorithmus zur Berechnung einfacher Konstanten (engl. *simple constants (SC)*) (POPL'73).

Beachte:

- ▶ Der Algorithmus zur Berechnung einfacher Konstanten aus Kapitel 6.6 stimmt im Ergebnis, nicht jedoch in den verwendeten Datenstrukturen mit Kildalls Algorithmus überein.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

649/863

# Erweiterungen von Kildalls SC-Algorithmus (1)

Verbesserungen von Kildalls Algorithmus zielen auf:

- ▶ **Anwendungsreichweite**
  - ▶ **Interprozedurale Erweiterungen**
    - ▶ Callahan, Cooper, Kennedy, Torczon (SCC'86)
    - ▶ Grove, Torczon (PLDI'93)
    - ▶ Metzger, Stroud (LOPLAS, 1993)
    - ▶ Sagiv, Reps, Horwitz (TAPSOFT'95)
    - ▶ Duesterwald, Gupta, Soffa (TOPLAS, 1997)
    - ▶ ...
  - ▶ **Explizit parallele Erweiterungen**
    - ▶ Lee, Midkiff, Padua (J. of Parallel Prog., 1998)
    - ▶ Knoop (Euro-Par'98)
    - ▶ ...

wobei **Anwendungsreichweite** zulasten von **Ausdruckskraft** gewonnen wird: **Kopierkonstanzen** (engl. **copy constants**), **lineare Konstanten** (engl. **linear constants**) anstelle von **einfachen Konstanten** (engl. **simple constants**).

# Erweiterungen von Kildalls SC-Algorithmus (2)

## ▶ Performanz

- ▶ SSA-Form: Wegman, Zadeck (POPL'85)
- ▶ ...

## ▶ Ausdruckskraft

- ▶ "SC+": Kam, Ullman (Acta Informatica, 1977)
- ▶ **Konditionale Konstanten** (engl. **conditional constants**): Wegman, Zadeck (POPL'85)
- ▶ **Endliche Konstanten** (engl. **finite constants**): Steffen, Knoop (MFCS'89)
- ▶ ...

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

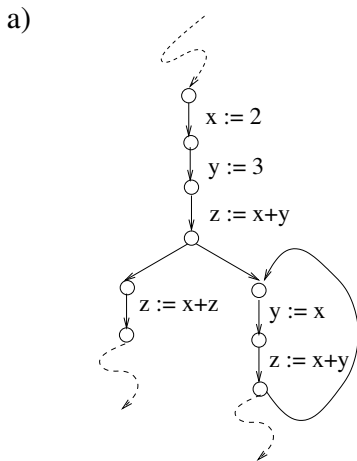
14.3

Kap. 15

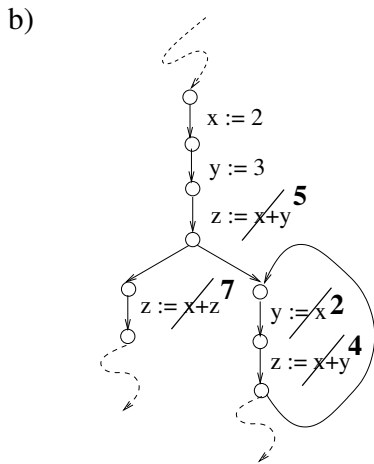
Kap. 16

651/863

# Warum nach größerer Ausdruckskraft streben?



Original program



After simple constant propagation

Das SC-Transformationsergebnis ist doch überzeugend, nicht?

# Tatsächlich ist es nicht überzeugend

Der Begriff **einfacher Konstanten** ist **schwach**:

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

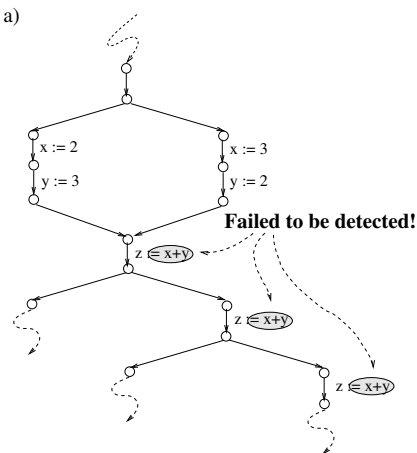
14.2

14.3

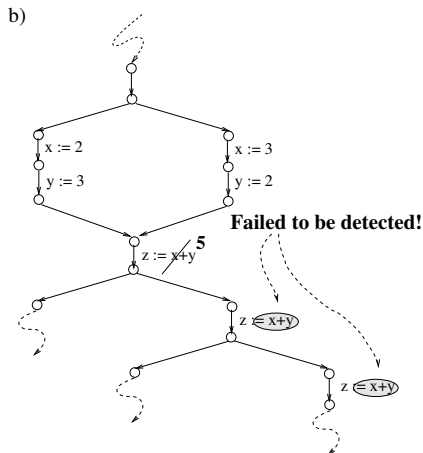
Kap. 15

Kap. 16

653/863



After simple constant propagation  
(Note: No effect at all!)



After simple constant propagation  
enriched by the "look-ahead-of-one" heuristics  
of Kam and Ullman

# Entscheidbarkeitsfragen für Konstantenfaltung

Es gilt:

- ▶ **Konstantenfaltung** ist **unentscheidbar**: Reif, Lewis (POPL 1977)

Andererseits:

- ▶ **Konstantenfaltung** ist **entscheidbar** auf schleifenfreien Programmen (azyklische Programme (engl. directed acyclic graphs (DAGs))).

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

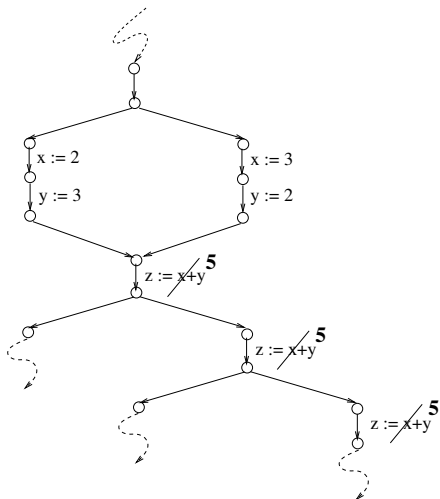
Kap. 15

Kap. 16

654/863

# Endliche Konstanten (1)

...sind **optimal** (**vollständig**) auf **schleifenfreien Programmen**,  
d.h. jede Konstante in einem schleifenfreien Programm ist eine  
**endliche Konstante!**



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

655/863

# Endliche Konstanten (2)

## Intuitiv:

- ▶ Endliche Konstanten sind eine systematisch, erschöpfend und endlich zu berechnende Erweiterung der von Kam&Ullmanns heuristischer “1-Anweisungsvorschau” erfassten Konstanten.

## Schlüsselfakten über endliche Konstanten:

- ▶ Für schleifenfreie Programme sind endliche Konstanten optimal.
- ▶ Für Programme mit beliebigem Kontrollfluss sind endliche Konstanten eine echte Obermenge einfacher Konstanten.
- ▶ Die Berechnungskomplexität endlicher Konstanten ist exponentiell im schlechtesten Fall (auch für schleifenfreie Programme).



# Hinsichtlich der Berechnungskomplexität

...endlicher Konstanten sollte man bedenken:

## Theorem (14.1.1)

*Konstantenfaltung und -ausbreitung ist für schleifenfreie Programme **co-NP-vollständig**.*

Knoop, Rüthing (CC'00)

Müller-Olm, Rüthing (ESOP'01)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

657/863

# Zurück zum laufenden Beispiel

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

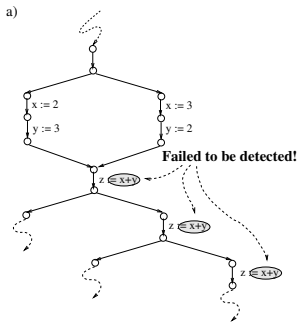
14.2

14.3

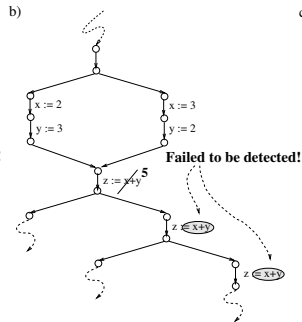
Kap. 15

Kap. 16

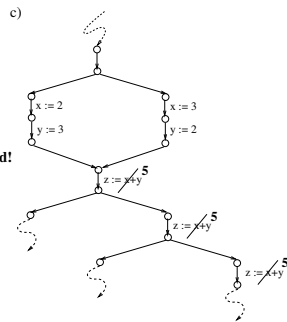
658/863



After simple constant propagation  
(Note: No effect at all!)



After simple constant propagation  
enriched by the "look-ahead-of-one" heuristics  
of Kam and Ullman



# Ein neuer Konstantenfaltungsalgorithmus

...der eine sorgfältige Balance hält zwischen

- ▶ **Ausdruckskraft** und **Performanz**.

Dieser neue Konstantenfaltungsalgorithmus stützt sich

- ▶ auf den **Wertegraphen** (engl. **value graph**) von Alpern, Wegman, and Zadeck (POPL'88)
- ▶ der sich seinerseits auf eine **SSA**-Repräsentation von Programmen stützt (**SSA = Static Single Assignment Form**, Cytron et al. (POPL'89)).

Insgesamt erhalten wir den

- ▶ **VG-Konstantenfaltungsalgorithmus**, einen Konstantenfaltungsalgorithmus **mit SSA-Form**, nicht auf SSA-Form.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

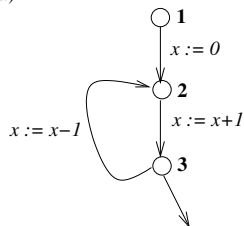
659/863

# Kapitel 14.2

## Der VG-Konstantenfaltungsalgorithmus

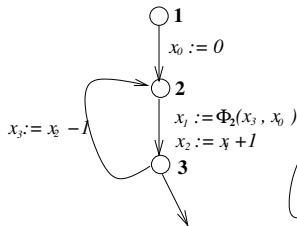
# The Wertegraph von Alpern, Wegman, Zadeck

a)



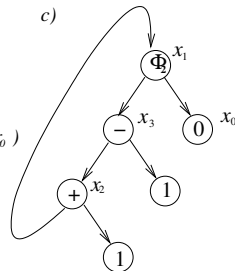
Ausgangsprogramm

b)



SSA-Form

c)



Wertegraph

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

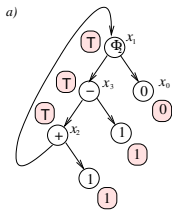
14.3

Kap. 15

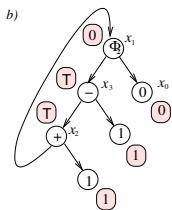
Kap. 16

661/863

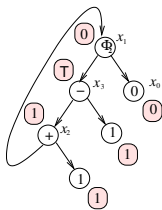
# Konstantenfaltung auf dem Wertegraphen



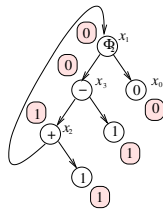
After the initialization step



After the 1st iteration step



After the 2nd iteration step



After the 3rd iteration step: Stable!

Analyseresultat:  $x_2$  und  $x_3$  sind von konstantem Wert!

# Konstantenfaltung auf dem Wertegraphen

...mit zwei Ausprägungen:

- ▶ **Der VG-Grundalgorithmus**  
...berechnet einfache Konstanten.
- ▶ **Der volle VG-Algorithmus**  
...geht über einfache Konstanten und die 1-Vorschauheuristik hinaus.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

**14.2**

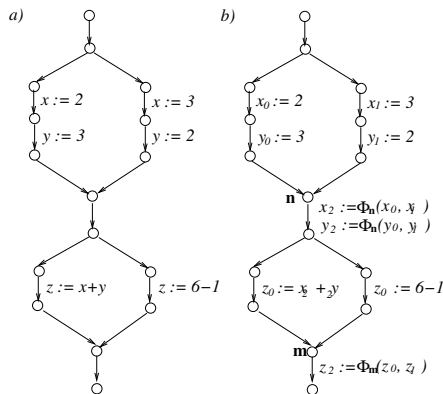
14.3

Kap. 15

Kap. 16

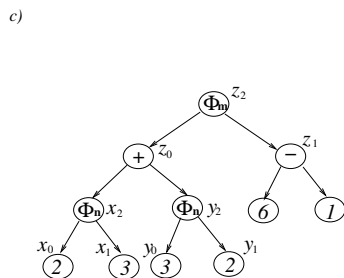
663/863

# Beispiel zur Veranschaulichung des vollen VG-Algorithmus



Ausgangsprogramm

SSA-Form



Wertegraph

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

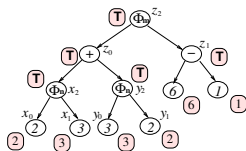
Kap. 15

Kap. 16

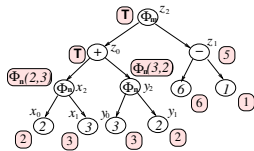
664/863



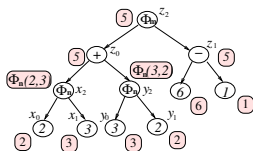
# Der volle VG-Algorithmus auf dem Wertegraphen



The start annotation



After the first iteration



After the second iteration. Stable!

## Der Clou:

- ▶ Einführung von  $\Phi$ -Konstanten und
- ▶ Anpassung der Evaluationsfunktion auf Wertegraphen!

# Hauptergebnisse

Für beliebigen, nicht eingeschränkten Kontrollfluss gilt:

- ▶ Der volle Algorithmus entdeckt eine **Obermenge einfacher Konstanten**.

Für azyklischen, schleifenfreien Kontrollfluss gilt:

- ▶ Der volle Algorithmus entdeckt die Konstanz jeden Terms, der ausschließlich aus in seinen relevanten Argumenten **injektiven** Operatoren aufgebaut ist.

Insgesamt gilt:

- ▶ Der volle Algorithmus erreicht eine ausgewogene Balance zwischen Ausdruckskraft und Performanz.
- ▶ Die Abstützung auf **SSA-Form** und **Wertegraph** sind dafür **essentiell**.

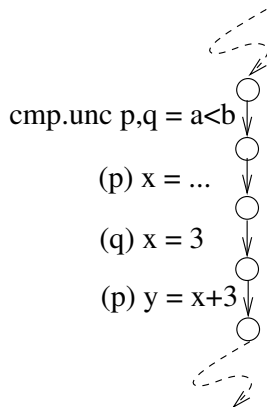
...zum VG-Konstantenfaltungsalgorithmus:

- ▶ Jens Knoop, Oliver Rüthing. [Constant Propagation on the Value Graph: Simple Constants and Beyond](#). In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.

# Kapitel 14.3

## Der PVG-Konstantenfaltungsalgorithmus

# Prädikatierter Code



...als Resultat sog. **if-Konversion**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

**14.3**

Kap. 15

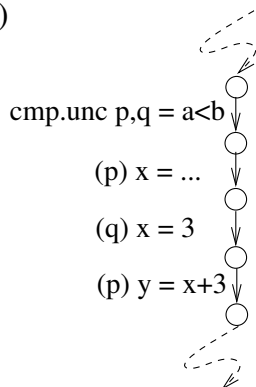
Kap. 16

669/863

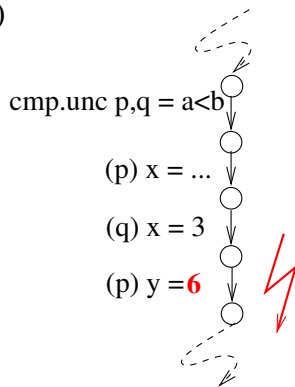
# Naive Konstantenfaltung

... auf prädikatiertem Code schlägt fehl:

a)



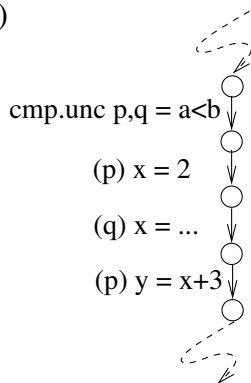
b)



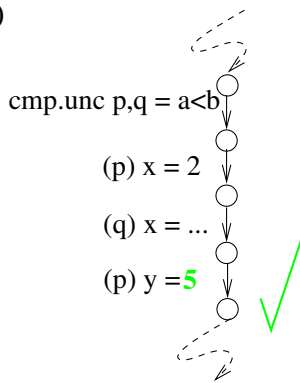
# Naive korrekte Konstantenfaltung

...scheint andererseits zu konservativ und zu viele Transformationsmöglichkeiten auszulassen:

a)



b)



# Ziel

Ein intelligenterer Umgang mit prädikatiertem Code.

Hyperblöcke sind

- ▶ wichtige Komponenten prädikatierten Codes.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

**14.3**

Kap. 15

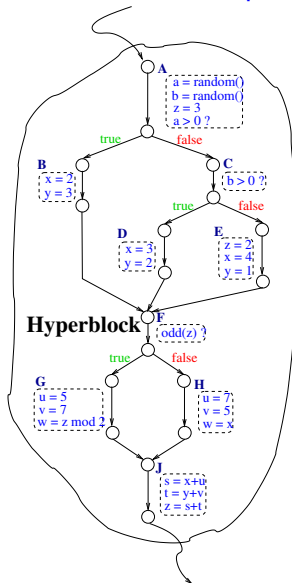
Kap. 16

672/863



# Hyperblöcke

...ein Eintrittspunkt, mehrere Austrittspunkte:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

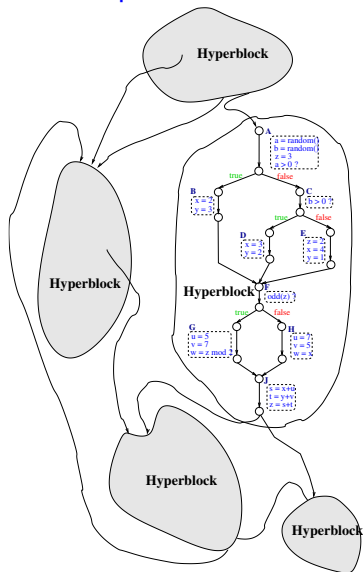
Kap. 15

Kap. 16

673/863

# Hyperblockzerlegung eines Programms

Unser durchgehendes Beispiel:



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

674/863

# Der PVG-Konstantenausbreitungsalgorithmus

...mit zwei<sup>+</sup> Ausprägungen:

- ▶ Der PVG-Grundalgorithmus
- ▶ Der volle PVG-Algorithmus

zuzüglich einiger

- ▶ performanz-gesteigerter Varianten.

Alle diese Algorithmenvarianten bestehen jeweils aus einer

- ▶ lokalen und
- ▶ globalen

Analysestufe.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

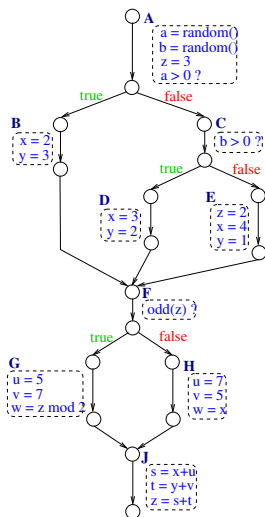
Kap. 15

Kap. 16

675/863

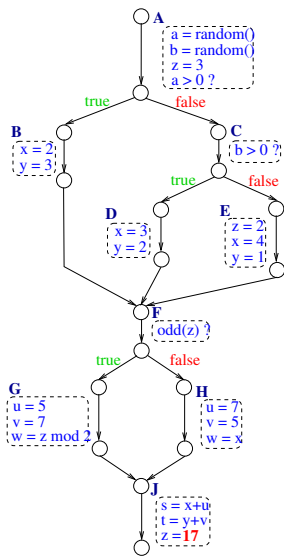
# Diskussion der lokalen Analysestufe

Dazu betrachten wir folgenden **Hyperblock**:



Original Hyperblock

# Die PVG-Grundalgorithmustransformation



The Non-Deterministic Path-Precise  
Basic Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

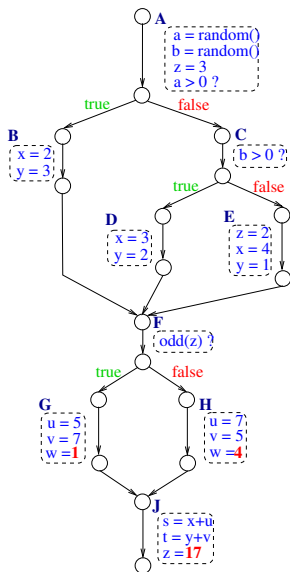
14.3

Kap. 15

Kap. 16

677/863

# Die volle PVG-Algorithmustransformation



The Deterministic Path-Precise  
Full Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

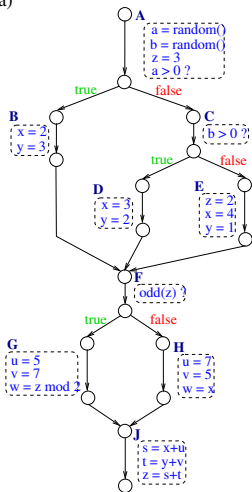
Kap. 15

Kap. 16

678/863

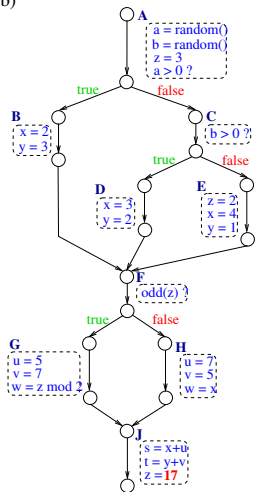
# Ausgangsprogramm & beide Transformationen

a)



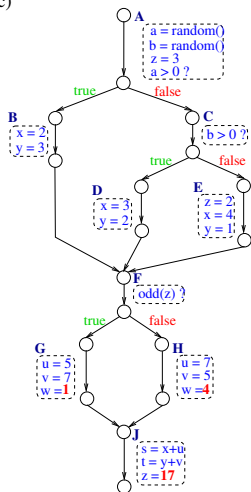
Original Hyperblock

b)



The Non-Deterministic Path-Precise  
Basic Optimization

c)



The Deterministic Path-Precise  
Full Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

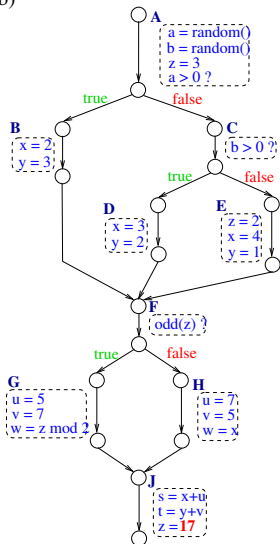
Kap. 15

Kap. 16

679/863

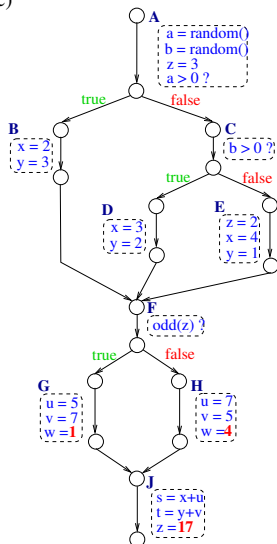
# Beide Transformationen auf einen Blick

b)



The Non-Deterministic Path-Precise  
Basic Optimization

c)



The Deterministic Path-Precise  
Full Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

680/863



# Ursprünglicher und prädikatierter Code

## Ursprünglicher Hyperblock | Nach if-Konversion

===== | =====

```
begin \\ Original Hyperblock | begin \\ After if-Conversion
(a,b) = (random(),random()); | (p0) (a,b) = (random(),random());
z = 3; | (p0) z = 3;
if a>0 then | (p0) cmp.unc B,C (a>0);
  x = 2; | (B) x = 2;
  y = 3 | (B) y = 3;
elseif b>0 then | (C) cmp.unc D,E (b>0);
  x = 3; | (D) x = 3;
  y = 2 | (D) y = 2;
else |
  z = 2; | (E) z = 2;
  x = 4; | (E) x = 4;
  y = 1 fi; | (E) y = 1;
if odd(z) then | (p0) cmp.unc G,H (odd(z));
  u = 5; | (G) u = 5;
  v = 7; | (G) v = 7;
  w = z mod 2 | (G) w = z mod 2;
else |
  u = 7; | (H) u = 7;
  v = 5; | (H) v = 5;
  w = x fi; | (H) w = x;
s = x+u; | (p0) s = x+u;
t = y+v; | (p0) t = y+v;
z = s+t end. | (p0) z = s+t end.
```

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

681/863

# Prädikierte SSA-Form (PSSA-Form)

...von Carter, Simon, Calder, Ferrante (PACT'99):

```
begin (p0)      A = OR(TRUE);           | [*] (HFBA)   w2 = x1;
(A)            (a1,b1) = (random(),random()); | [*] (HFDCA) w2 = x2;
(A)            z1 = 3;                   | (HFECA)    w2 = x3;
(A)            cmp.unc BA,CA (a1>0);      | (H)        u2 = 7;
(p0)           B = OR(BA);               | (H)        v2 = 5;
(p0)           C = OR(CA);               | (GFBA)     JGFBA = OR(TRUE);
(B)            x1 = 2;                   | (GFDCA)   JGFDCA = OR(TRUE);
(B)            y1 = 3;                   | [*] (GFECA) JGFECA = OR(TRUE);
(C)            cmp.unc DCA,ECA (b1>0);    | [*] (HFBA)  JHFBA = OR(TRUE);
(p0)           D = OR(DCA);              | [*] (HFDCA) JHFDCA = OR(TRUE);
(p0)           E = OR(ECA);              | (HFECA)   JHFECA = OR(TRUE);
(D)            x2 = 3;                   | [-] (p0)   J = OR(JGFBA,JGFDCA,
(E)            y2 = 2;                   |           JGFECA,JHFBA,
(E)            z2 = 2;                   |           JHFDCA,JHFECA);
(E)            x3 = 4;                   | (JGFBA)   s1 = x1+u1;
(E)            y3 = 1;                   | (JGFBA)   t1 = y1+v1;
(BA)           FBA = OR(TRUE);           | [*] (JGFDCA) s1 = x2+u1;
(DCA)          FDCA = OR(TRUE);          | [*] (JGFDCA) t1 = y2+v1;
(ECA)          FECA = OR(TRUE);          | (JGFECA)  s1 = x3+u1;
(p0)           F = OR(FBA,FDCA,FECA);    | (JGFECA)  t1 = y3+v1;
(FBA)          cmp.unc GFBA,HFBA (odd(z1)); | [*] (JHFBA) s1 = x1+u2;
(FDCA)         cmp.unc GFDCA,HFDCA (odd(z1)); | [*] (JHFBA) t1 = y1+v2;
(FECA)         cmp.unc GFECA,HFECA (odd(z2)); | [*] (JHFDCA) s1 = x2+u2;
[-] (p0)       G = OR(GFBA,GFDCA,GFECA); | [*] (JHFDCA) t1 = y2+v2;
[-] (p0)       H = OR(HFBA,HFDCA,HFECA); | (JHFECA)  s1 = x3+u2;
(GFBA)         w1 = z1 mod 2;             | (JHFECA)  t1 = y3+v2;
(GFDCA)        w1 = z1 mod 2;             | (J)        z3 = s1+t1;
[*] (GFECA)    w1 = z2 mod 2;             | end.
(G)            u1 = 5;
(G)            v1 = 7;
```

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

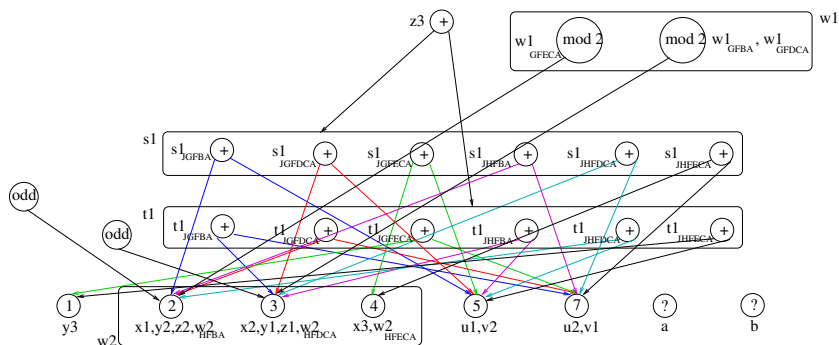
Kap. 15

Kap. 16

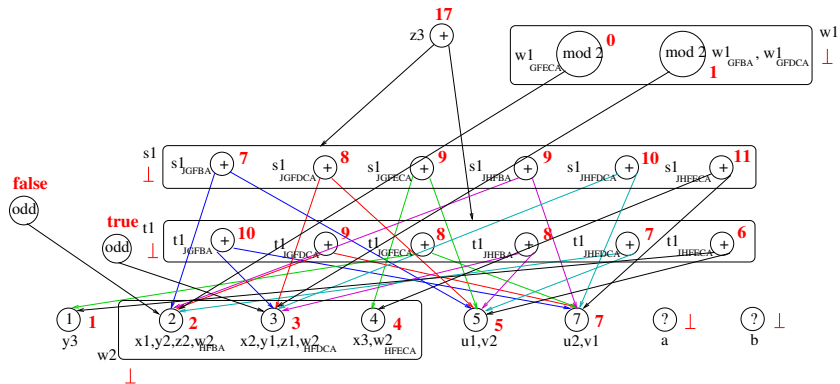
682/863

# Die Grundform des prädikatierten Wertegraphen auf PSSA-Grundlage

...ohne Ausnutzung der Wächterprädikate (engl. guarding predicates):



# Nach Konstantenfaltung durch PVG-Grundalg.



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

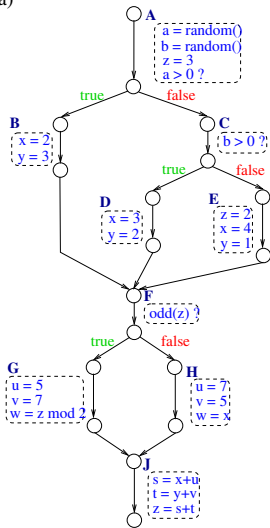
Kap. 15

Kap. 16

684/863

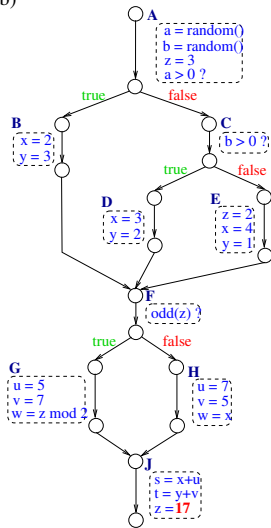
# Die PVG-Grundalgorithmustransformation

a)



Original Hyperblock

b)



The Non-Deterministic Path-Precise  
Basic Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

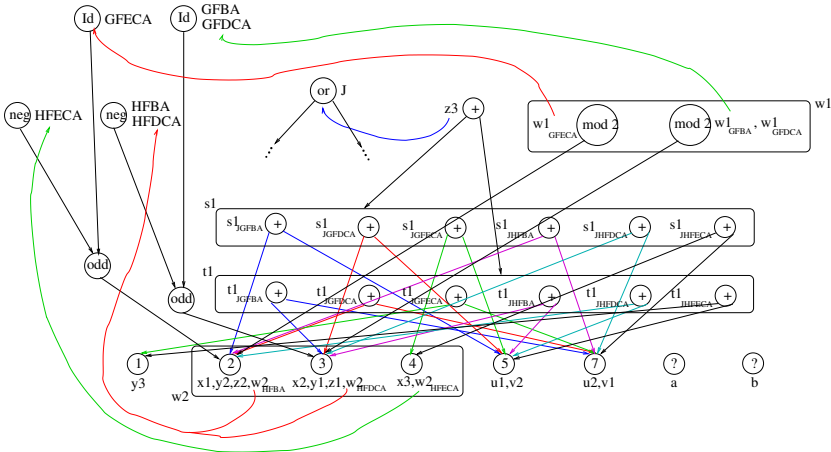
Kap. 15

Kap. 16

685/863

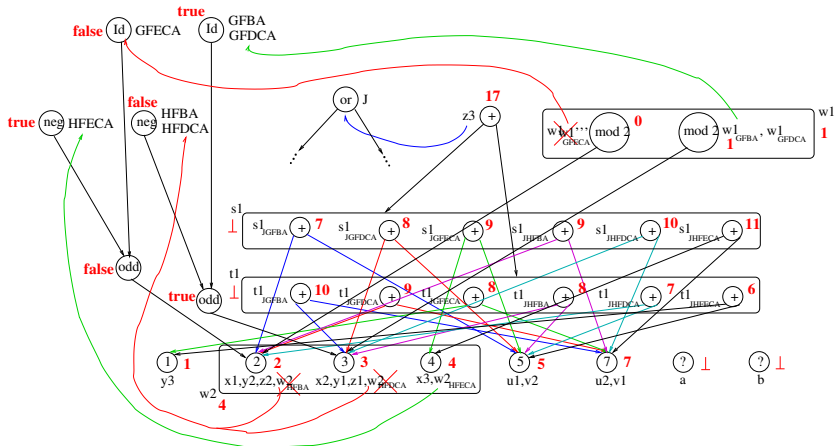
# Der prädikierte Wertegraph

...unter Ausnutzung der Wächterprädikate (engl. guarding predicates):



- Inhalt
- Kap. 1
- Kap. 2
- Kap. 3
- Kap. 4
- Kap. 5
- Kap. 6
- Kap. 7
- Kap. 8
- Kap. 9
- Kap. 10
- Kap. 11
- Kap. 12
- Kap. 13
- Kap. 14
  - 14.1
  - 14.2
  - 14.3
- Kap. 15
- Kap. 16

# Nach Konstantenfaltung durch vollen PVG-A.



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

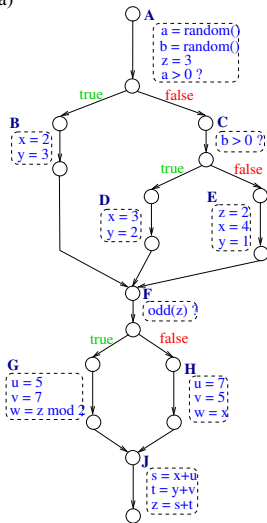
Kap. 15

Kap. 16

687/863

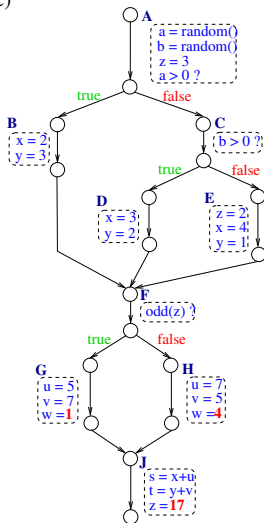
# Die volle PVG-Algorithmustransformation

a)



Original Hyperblock

c)



The Deterministic Path-Precise Full Optimization

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

688/863



# Der transformierte Hyperblock in PSSA-Form

```
begin (p0)      A = OR(TRUE);          |
(A)            a1 = random();         | [-] (p0)      G = OR(GFBA,GFDC);
(A)            b1 = random();         | [-] (p0)      H = OR(HFECA);
(A)            z1 = 3;                | (G)          w1 = 1;
(A)            cmp.unc BA,CA (a1>0);  | (G)          u1 = 5;
(p0)           B = OR(BA);            | (G)          v1 = 7;
(p0)           C = OR(CA);            | (HFECA)      w2 = 4;
(B)            x1 = 2;                | (H)          u2 = 7;
(B)            y1 = 3;                | (H)          v2 = 5;
(C)            cmp.unc DCA,ECA (b1>0);| (GFBA)      JGFBA = OR(TRUE);
(p0)           D = OR(DCA);           | (GFDCA)     JGFDCA = OR(TRUE);
(p0)           E = OR(ECA);           | (HFECA)     JHFECA = OR(TRUE);
(D)            x2 = 3;                | [-] (p0)    J = OR(JGFBA,JGFECA,
(D)            y2 = 2;                |              JHFECA);
(E)            z2 = 2;                | (JGFBA)     s1 = 7;
(E)            x3 = 4;                | (JGFBA)     t1 = 10;
(E)            y3 = 1;                | (JGFECA)    s1 = 9;
(BA)           FBA = OR(TRUE);        | (JGFECA)    t1 = 8;
(DCA)          FDCA = OR(TRUE);       | (JHFECA)    s1 = 11;
(ECA)          FECA = OR(TRUE);       | (JHFECA)    t1 = 6;
(p0)           F = OR(FBA,FDCA,FECA); | (J)         z3 = 17;
(FBA)          cmp.unc GFBA,HFBA (TRUE)); | end.
(FDCA)         cmp.unc GFDCA,HFDCA (TRUE); |
(FECA)         cmp.unc GFECA,HFECA (FALSE); |
```

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

689/863

# Hauptergebnisse

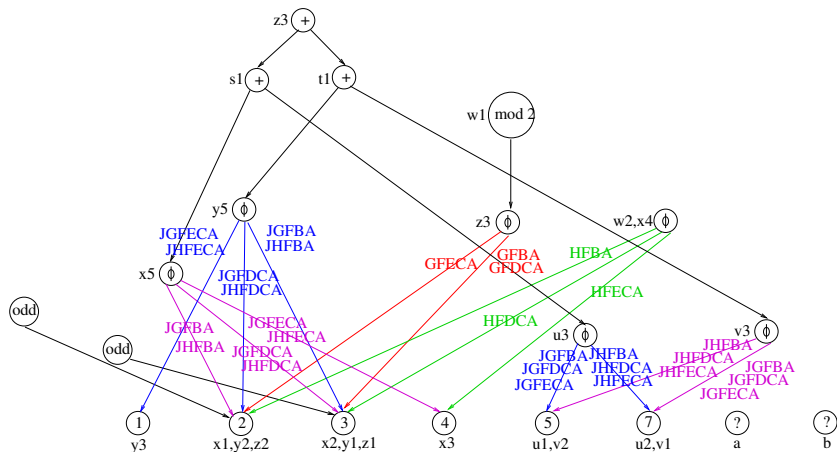
## Korrektheit

- ▶ Der globale Konstantenfaltungsalgorithmus ist **korrekt** (sowohl für die Grundform wie für die volle Variante der lokalen Analysestufe).

## Vollständigkeit/Optimalität

- ▶ Der Grundalgorithmus der lokalen Analysestufe ist **pfad-präzise** bezüglich nicht-deterministischer Interpretation von Verzweigungsbedingungen.
- ▶ Der volle Algorithmus der lokalen Analysestufe ist **prädikatssensitiv pfad-präzise**.

# Performanzsteigerung: Grundalgorithmus (1)



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

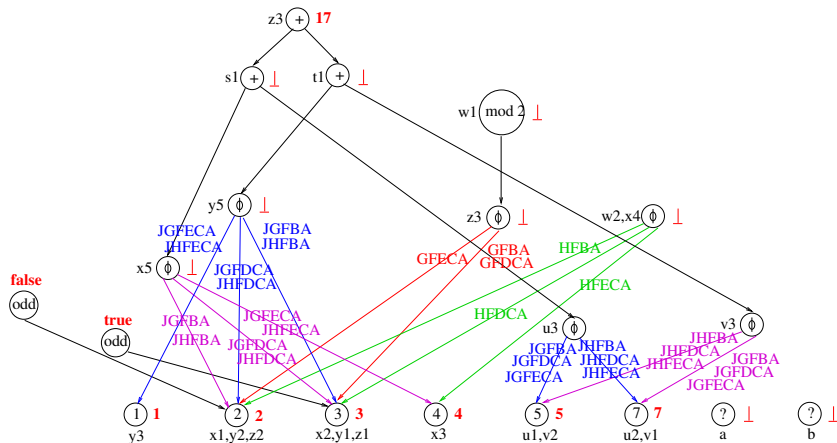
14.3

Kap. 15

Kap. 16

691/863

# Performanzsteigerung: Grundalgorithmus (2)



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

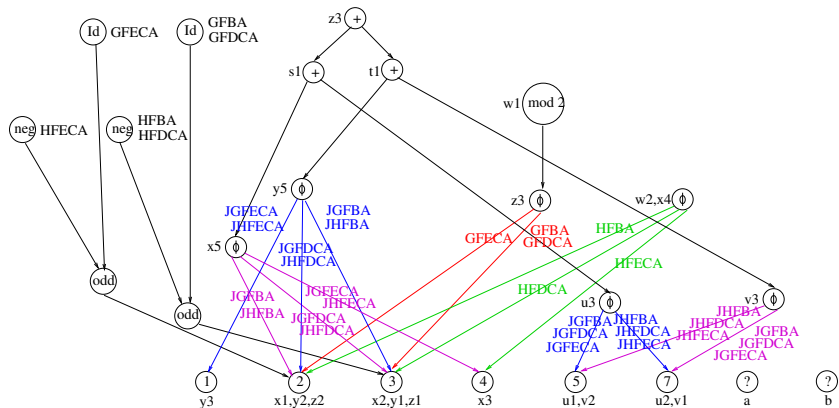
14.3

Kap. 15

Kap. 16

692/863

# Performanzsteigerung: Voller Algorithmus (1)



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16

693/863



# Zusammenfassung und Schlussfolgerungen

## Konstantenfaltung und SSA/PSSA-Form:

- ▶ sind perfekt aufeinander abgestimmt – SSA/PSSA sind wirklich von Hilfe für Analyse und Transformation!
- ▶ Der Schlüssel: Wertegraph und prädikatierter Wertegraph.

## Offen für Erweiterungen, z.B.:

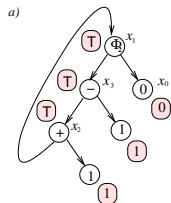
- ▶ Wertegraph: Bedingte Konstanten (engl. conditional constants).

## Zusammenfassend:

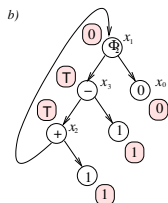
- ▶ Konstantenfaltung ist besonders geeignet, Vorteile aus der Nutzung von (P)SSA zur Programmrepräsentation aufzuzeigen.

# Konstantenfaltung auf SSA-basiertem Wertegraph

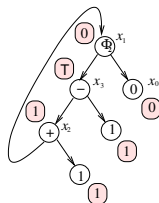
...erzielt **Triple E** Rating: **E**xpressive, **E**fficient, **E**asy!



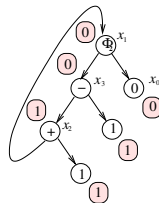
After the initialization step



After the 1st iteration step



After the 2nd iteration step



After the 3rd iteration step: Stable!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2

14.3

Kap. 15

Kap. 16



696/863



...zum PVG-Konstantenfaltungsalgorithmus:

- ▶ Jens Knoop, Oliver Rüthing. [Constant Propagation on Predicated Code](#). Journal of Universal Computer Science 9(8):829-850, 2003. (Sonderausgabe zur SBLP'03).

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (1)

-  Bowen Alpern, Mark N. Wegman, F. Ken Zadeck. *Detecting Equality of Variables in Programs*. In Conference Record of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'88), 1-11, 1988.
-  Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Ken Zadeck. *An Efficient Method of Computing Static Single Assignment Form*. In Conference Record of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'89), 25-35, 1989.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

14.1

14.2




14.3

Kap. 15

Kap. 16

698/863




# Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (2)

-  Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Ken Zadeck. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*. ACM Transactions on Programming Languages and Systems (TOPLAS) 13(4):451-490, 1991.
-  John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. Acta Informatica 7:305-317, 1977.
-  Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.




# Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (3)

-  Jens Knoop, Oliver Rüthing. *Constant Propagation on the Value Graph: Simple Constants and Beyond*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.
-  Jens Knoop, Oliver Rüthing. *Constant Propagation on Predicated Code*. Journal of Universal Computer Science 9(8):829-850, 2003. (Sonderausgabe zur SBLP'03).
-  John H. Reif, Harry R. Lewis. *Symbolic Evaluation and the Global Value Graph*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (4)

-  Oliver Rüthing, Jens Knoop, Bernhard Steffen. *Detecting Equalities of Variables: Combining Efficiency with Precision*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 232-247, 1999.
-  Oliver Rüthing, Markus Müller-Olm. *On the Complexity of Constant Propagation*. In Proceedings of the 10th European Symposium on Programming (ESOP 2001), Springer-V., LNCS 2028, 190-205, 2001.
-  Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. Theoretical Computer Science 80(2):303-318, 1991.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 14 (5)

-  Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. Information Processing Society of Japan 38(11):2237-2250, 1990.
-  Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.
-  Mark N. Wegman, F. Ken Zadeck. *Constant Propagation with Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.

# Teil IV

## Abstrakte Interpretation und Modellprüfung

# Kapitel 15

## Abstrakte Interpretation und DFA

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

**Kap. 15**

15.1

15.2

15.3

15.4



# Motivation

Die **Theorie abstrakter Interpretation** — ein “mondäner” Programmmanalyseansatz.

Intuitiv:

- ▶ Der **DFA-Ansatz** beinhaltet die Spezifikation einer Programmmanalyse, deren Korrektheit separat und unabhängig *a posteriori* validiert wird.
- ▶ Der **abstrakte Interpretationsansatz** beinhaltet den Korrektheitsnachweis von Anfang an als integralen Bestandteil der Programmmanalyse.

# Kapitel 15.1

## Theorie abstrakter Interpretation

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**15.1**

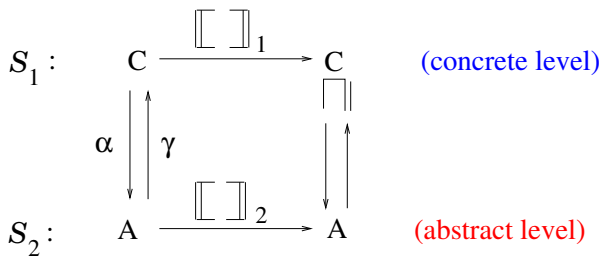
15.2

15.3

15.4

# Theorie abstrakter Interpretation

...ein Ansatz mit 2 Beobachtungsniveaus: Konkret&abstrakt



Bezeichnungen:

- ▶ Abstraktionsfunktion  $\alpha : C \rightarrow A$
- ▶ Konkretisierungsfunktion  $\gamma : A \rightarrow C$

Wohlzusammenhangsforderung:

- ▶ Abstraktions- und Konkretisierungsfunktion müssen eine Galois-Verbindung bilden.

# Galois-Verbindungen

## Definition (15.1.1, Galois-Verbindung)

Seien  $\mathcal{C} = (C, \sqcup_C, \sqsubseteq_C, \perp_C, \top_C)$  und  $\mathcal{A} = (A, \sqcup_A, \sqsubseteq_A, \perp_A, \top_A)$  zwei vollständige Halbverbände und  $\alpha : \mathcal{C} \rightarrow \mathcal{A}$  und  $\gamma : \mathcal{A} \rightarrow \mathcal{C}$  eine Abstraktions- und Konkretisierungsfunktion. Dann definieren wir:

Das Viertupel  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  heißt **Galois-Verbindung** zwischen  $\mathcal{C}$  und  $\mathcal{A}$  gdw  $\alpha$  und  $\gamma$  sind **monoton** und erfüllen:

1.  $\gamma \circ \alpha \sqsupseteq_C Id_C$
2.  $\alpha \circ \gamma \sqsubseteq_A Id_A$

mit  $Id_C$  und  $Id_A$  Identität auf  $C$  und  $A$ , d.h.  $Id_C(c) = \lambda c. c$  und  $Id_A(a) = \lambda a. a$  für alle  $c \in C$  und  $a \in A$ .

# Eigenschaften von Galois-Verbindungen (1)

## Lemma (15.1.2, Eindeutigkeit von Galois-Verbindungsbeziehungen)

Sei  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung. Dann gilt:

1.  $\alpha$  bestimmt  $\gamma$  *eindeutig* durch  $\gamma(a) = \bigsqcup \{c \mid \alpha(c) \sqsubseteq_A a\}$ .
2.  $\gamma$  bestimmt  $\alpha$  *eindeutig* durch  $\alpha(c) = \bigsqcap \{a \mid c \sqsubseteq_C \gamma(a)\}$ .
3.  $\alpha$  ist *additiv* und  $\gamma$  *distributiv*.

*Insbesondere gilt:  $\alpha(\perp_C) = \perp_A$  und  $\gamma(\top_A) = \top_C$ .*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Eigenschaften von Galois-Verbindungen (2)

## Lemma (15.1.3, Existenz und Eindeutigkeit von Galois-Verbindungsergänzungen)

1. Ist  $\alpha : \mathcal{C} \rightarrow \mathcal{A}$  additiv, dann gibt es ein  $\gamma : \mathcal{A} \rightarrow \mathcal{C}$ , so dass  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung ist.
2. Ist  $\gamma : \mathcal{A} \rightarrow \mathcal{C}$  distributiv, dann gibt es ein  $\alpha : \mathcal{C} \rightarrow \mathcal{A}$ , so dass  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung ist.

## Lemma (15.1.4, Keine Informationsverluste und -gewinne)

Sei  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung. Dann gilt:

1.  $\alpha \circ \gamma \circ \alpha = \alpha$
2.  $\gamma \circ \alpha \circ \gamma = \gamma$

# Galois-Einpassungen

## Definition (15.1.5, Galois-Einpassung)

Seien  $\mathcal{C} = (C, \sqcup_C, \sqsubseteq_C, \perp_C, \top_C)$  und  $\mathcal{A} = (A, \sqcup_A, \sqsubseteq_A, \perp_A, \top_A)$  zwei vollständige Halbverbände und  $\alpha : \mathcal{C} \rightarrow \mathcal{A}$  und  $\gamma : \mathcal{A} \rightarrow \mathcal{C}$  eine Abstraktions- und Konkretisierungsfunktion. Dann definieren wir:

Das Viertupel  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  heißt **Galois-Einpassung** zwischen  $\mathcal{C}$  und  $\mathcal{A}$  gdw  $\alpha$  und  $\gamma$  sind **monoton** und erfüllen:

1.  $\gamma \circ \alpha \sqsupseteq_C Id_C$
2.  $\alpha \circ \gamma = Id_A$

mit  $Id_C$  und  $Id_A$  Identität auf  $C$  und  $A$ , d.h.  $Id_C(c) = \lambda c. c$  und  $Id_A(a) = \lambda a. a$  für alle  $c \in C$  und  $a \in A$ .

# Interpretation

Ist  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine **Galois-Einpassung** zwischen  $\mathcal{C}$  und  $\mathcal{A}$ , so führt eine vorangehende Konkretisierung vor einer Abstraktion

- ▶ nicht zu einem Informationsverlust:  $\alpha \circ \gamma = Id_{\mathcal{A}}$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

712/863



# Eigenschaften von Galois-Einpassungen

## Lemma (15.1.6, Äquivalenzaussagen)

Sei  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung. Dann sind die folgenden Aussagen äquivalent:

1.  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Einpassung.
2.  $\alpha$  ist surjektiv, d.h.  $\forall a \in A \exists c \in C. \alpha(c) = a$ .
3.  $\gamma$  ist injektiv, d.h.  $\forall a_1, a_2 \in A. \gamma(a_1) = \gamma(a_2) \Rightarrow a_1 = a_2$ .
4.  $\gamma$  ist ordnungs-ähnlich, d.h.  
 $\forall a_1, a_2 \in A. \gamma(a_1) \sqsubseteq_C \gamma(a_2) \iff a_1 \sqsubseteq_A a_2$ .

# Konstruktion von Galois-Einpassungen

## Lemma (15.1.7, Galois-Einpassungskonstruktion)

Sei  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung und  $\rho : \mathcal{A} \rightarrow \mathcal{A}$  der wie folgt definierte Reduktionsoperator:

$$\forall a \in \mathcal{A}. \rho(a) = \sqcap \{a' \mid \gamma(a) = \gamma(a')\}$$

Dann ist  $\rho(\mathcal{A}) = (\{\rho(a) \mid a \in \mathcal{A}\}, \sqcup_{\mathcal{A}}, \sqsubseteq_{\mathcal{A}}, \perp_{\mathcal{A}}, \top_{\mathcal{A}})$  ein vollständiger Halbverband und das 4-Tupel  $(\mathcal{C}, \alpha, \gamma, \rho(\mathcal{A}))$  eine Galois-Einpassung.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

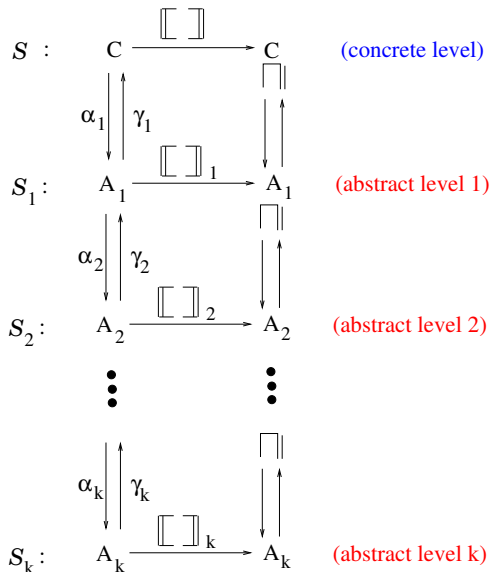
15.1

15.2

15.3

15.4

# Hierarchie von Galois-Verb./-Einpassungen



Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

715/863

# Kapitel 15.2

## Systematische Konstruktion abstrakter Interpretationen

# Übersicht

Im einzelnen betrachten wir 6 Methoden:

- ▶ Unabhängige Attributemethode
- ▶ Relationale Methode
- ▶ Totale Funktionenraummethode
- ▶ Monotone Funktionenraummethode
- ▶ Direkte Produktmethode
- ▶ Direkte Tensorproduktmethode

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

**15.2**

15.3

15.4

# Die unabhängige Attributemethode

## Lemma (15.2.1, Unabhängige Attributemethode)

Seien  $(\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1)$  und  $(\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2)$  zwei Galois-Verbindungen.

Dann ist auch  $(\mathcal{C}_1 \times \mathcal{C}_2, \alpha, \gamma, \mathcal{A}_1 \times \mathcal{A}_2)$  mit

$$\alpha(c_1, c_2) = (\alpha_1(c_1), \alpha_2(c_2))$$

$$\gamma(a_1, a_2) = (\gamma_1(a_1), \gamma_2(a_2))$$

eine Galois-Verbindung.

# Die relationale Methode

## Lemma (15.2.2, Relationale Methode)

Seien  $(\mathcal{P}(C_1), \alpha_1, \gamma_1, \mathcal{P}(A_1))$  und  $(\mathcal{P}(C_2), \alpha_2, \gamma_2, \mathcal{P}(A_2))$  zwei Galois-Verbindungen, wobei  $\mathcal{P}$  den Potenzmengenoperator bezeichnet.

Dann ist auch  $(\mathcal{P}(C_1 \times C_2), \alpha, \gamma, \mathcal{P}(A_1 \times A_2))$  mit

$$\alpha(CC) = \bigcup \{ \alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \mid (c_1, c_2) \in CC \}$$

$$\gamma(AA) = \{ (c_1, c_2) \mid \alpha_1(\{c_1\}) \times \alpha_2(\{c_2\}) \subseteq AA \}$$

für  $CC \subseteq C_1 \times C_2$  und  $AA \subseteq A_1 \times A_2$  eine Galois-Verbindung.

# Die totale Funktionenraummethode

## Lemma (15.2.3, Totale Funktionenraummethode)

Sei  $(\mathcal{C}, \alpha, \gamma, \mathcal{A})$  eine Galois-Verbindung und  $M$  eine Menge.

Dann ist auch  $([M \rightarrow \mathcal{C}], \alpha', \gamma', [M \rightarrow \mathcal{A}])$  mit

$$\alpha'(f) = \alpha \circ f$$

$$\gamma'(g) = \gamma \circ g$$

eine Galois-Verbindung.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

**15.2**

15.3

15.4

720/863



# Die monotone Funktionenraummethode

## Lemma (15.2.4, Monotone Funktionenraummethode)

Seien  $(\mathcal{C}_1, \alpha_1, \gamma_1, \mathcal{A}_1)$  und  $(\mathcal{C}_2, \alpha_2, \gamma_2, \mathcal{A}_2)$  zwei Galois-Verbindungen.

Dann ist auch  $([\mathcal{C}_1 \rightarrow \mathcal{C}_2], \alpha, \gamma, [\mathcal{A}_1 \rightarrow \mathcal{A}_2])$  mit

$$\alpha(f) = \alpha_2 \circ f \circ \gamma_1$$

$$\gamma(g) = \gamma_2 \circ g \circ \alpha_1$$

eine Galois-Verbindung.

# Die direkte Produktmethode

## Lemma (15.2.5, Direkte Produktmethode)

Seien  $(\mathcal{C}, \alpha_1, \gamma_1, \mathcal{A}_1)$  und  $(\mathcal{C}, \alpha_2, \gamma_2, \mathcal{A}_2)$  zwei Galois-Verbindungen.

Dann ist auch  $(\mathcal{C}, \alpha, \gamma, \mathcal{A}_1 \times \mathcal{A}_2)$  mit

$$\begin{aligned}\alpha(c) &= (\alpha_1(c), \alpha_2(c)) \\ \gamma(a_1, a_2) &= \gamma_1(a_1) \sqcap \gamma_2(a_2)\end{aligned}$$

eine Galois-Verbindung.

# Die direkte Tensorproduktmethode

## Lemma (15.2.6, Direkte Tensorproduktmethode)

Seien  $(\mathcal{P}(C), \alpha_1, \gamma_1, \mathcal{P}(A_1))$  und  $(\mathcal{P}(C), \alpha_2, \gamma_2, \mathcal{P}(A_2))$  zwei Galois-Verbindungen, wobei  $\mathcal{P}$  den Potenzmengenoperator bezeichnet.

Dann ist auch  $(\mathcal{P}(C), \alpha, \gamma, \mathcal{P}(A_1 \times A_2))$  mit

$$\begin{aligned}\alpha(C') &= \bigcup \{ \alpha_1(\{c\}) \times \alpha_2(\{c\}) \mid c \in C' \} \\ \gamma(AA) &= \{ c \mid \alpha_1(\{c\}) \times \alpha_2(\{c\}) \subseteq AA \}\end{aligned}$$

für  $C' \subseteq C$  und  $AA \subseteq A_1 \times A_2$  eine Galois-Verbindung.

# Kapitel 15.3

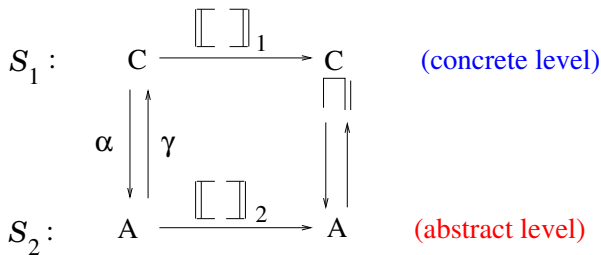
## Korrektheit abstrakter Interpretationen

# Korrektheit

Der in seiner Grund- bzw. erweiterten Form auf

- ▶ 2 bzw. einer höheren Zahl von **Beobachtungsniveaus** beruhende **abstrakte Interpretationsansatz** erlaubt die
- ▶ **Korrektheit** einer abstrakten Interpretation auf einem **niedrigeren** Beobachtungsniveau relativ zu einer anderen abstrakten Interpretation auf einem **höheren** Beobachtungsniveau

formal zu fassen und nachzuweisen.



# Vollständigkeit und Optimalität (1)

In diesem Sinne ist der auf Patrick und Radhia Cousot (POPL'77) zurückgehende Ansatz abstrakter Interpretation in seiner “klassischen” Ausprägung

- ▶ **semantik-orientiert** (unter Bezug auf eine **Standardsemantik** (besonders wichtig hierbei: Die sog. **Aufsammlungsemantik** (engl. **collecting semantics**))).

Der semantik-orientierte Ansatz unterstützt nicht *per se* einen

- ▶ **Vollständigkeits- und Optimalitätsbegriff**

und dessen **formalen Nachweis**.

# Vollständigkeit und Optimalität (2)

Zur Betrachtung von **Vollständigkeit und Optimalität** gehen wir in der Folge von der semantik-orientierten zu einer auf Bernhard Steffen (TAPSOFT'87, MFCS'89) zurückgehenden

- ▶ **beobachtungs-orientierten Perspektive** (ohne Bezug auf eine **Standardsemantik**)

über.

# Kapitel 15.4

## Vollständigkeit und Optimalität

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

**15.4**



# Vorbereitungen

In der Folge bezeichnen

- ▶ **N** eine Menge von Knoten, die für die Menge der Vorkommen elementarer Anweisungen steht,
- ▶  $G =_{df} (N, E, s, e)$  einen Flussgraphen mit Knotenmenge  $N \subseteq \mathbf{N}$ , Kantenmenge  $E \subseteq N \times N$ , Startknoten  $s \in N$  und Endknoten  $e \in N$ , wobei **s** keine Vorgänger, **e** keine Nachfolger hat;  $\mathbf{P}(G)$  die Menge aller Pfade von **s** nach **e** in  $G$ ,
- ▶ **FG** die Menge aller Flussgraphen über **N** und **LFG** die Menge aller linearen Flussgraphen über **N**, d.h. die Menge aller Flussgraphen mit genau einem Pfad von **s** und **e**,
- ▶  $\mathcal{C} =_{df} (C, \sqcup, \sqsubseteq, \perp, \top)$  einen vollständigen Halbverband mit kleinstem Element  $\perp$  und größtem Element  $\top$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Lokale abstrakte Semantik

## Definition (15.3.1, Lokale abstrakte Semantik)

Sei  $\llbracket \_ \rrbracket_l : N \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$  eine Funktion, die jedem Knoten  $n \in N$  eine **additive** Funktion auf  $\mathcal{C}$  zuordnet, d.h.

$$\forall n \in \mathbf{N} \forall C' \subseteq \mathcal{C}. \llbracket n \rrbracket_l(\bigsqcup C') = \bigsqcup \{ \llbracket n \rrbracket_l(c) \mid c \in C' \}.$$

Dann heißt das Paar  $(\llbracket \_ \rrbracket_l, \mathcal{C})$  **lokale abstrakte Semantik** bzw. **lokale abstrakte Interpretation**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Globale abstrakte Semantik

## Definition (15.3.2, Globale abstrakte Semantik)

Sei  $(\llbracket \_ \rrbracket_I, \mathcal{C})$  abstrakte Interpretation und  $\llbracket \_ \rrbracket : \mathbf{FG} \rightarrow (\mathcal{C} \rightarrow \mathcal{C})$  die Globalisierung von  $\llbracket \_ \rrbracket_I$ , d.h.  $\forall G \in \mathbf{FG} \forall c \in \mathcal{C}$  gilt:

$$\llbracket G \rrbracket(c) =_{df}$$

$$\begin{cases} \llbracket n_k \rrbracket_I \circ \dots \circ \llbracket n_1 \rrbracket_I(c) & \text{falls } G = (n_1, \dots, n_k) \in \mathbf{LFG} \\ \sqcup \{ \llbracket P \rrbracket(c) \mid P \in \mathbf{P}(G) \} & \text{sonst} \end{cases}$$

Dann heißt das Paar  $(\llbracket \_ \rrbracket, \mathcal{C})$  die von  $(\llbracket \_ \rrbracket_I, \mathcal{C})$  induzierte (globale) abstrakte Semantik.

# Abstraktion und Konkretisierung

## Definition (15.3.3, Abstraktion und Konkretisierung)

Seien  $\mathcal{S}_1 = (\llbracket \_ \rrbracket_1, \mathcal{C}_1)$  und  $\mathcal{S}_2 = (\llbracket \_ \rrbracket_2, \mathcal{C}_2)$  zwei abstrakte Semantiken.

- ▶ Eine Funktion  $A : \mathcal{C}_1 \rightarrow \mathcal{C}_2$  heißt **Abstraktionsfunktion**, in Zeichen  $\mathcal{S}_2 \leq_A \mathcal{S}_1$ , falls  $A$  additiv und surjektiv ist und die Korrektheitsbedingung

$$\forall n \in \mathbf{N}. A \circ \llbracket n \rrbracket_1 \sqsubseteq \llbracket n \rrbracket_2 \circ A$$

erfüllt.

- ▶ Die Funktion  $A^a : \mathcal{C}_2 \rightarrow \mathcal{C}_1$  definiert durch

$$\forall c \in \mathcal{C}_2. A^a(c) =_{df} \bigsqcup \{c' \mid A(c') = c\}$$

heißt **adjungierte** oder **Konkretisierungsfunktion** zu  $A$ .

# Anmerkungen zu Abstraktion&Konkretisierung

- ▶ **Additivität** ist eine wesentliche Anforderung an eine abstrakte Interpretation. Die meisten der folgenden Ergebnisse gelten nur unter dieser Voraussetzung.
- ▶ **Surjektivität** ist keine wesentliche Voraussetzung, erleichtert aber die formale Argumentation.
- ▶ **Paare aus Abstraktions- und Konkretisierungsfunktion**  $(A, A^a)$  sind **Paare adjungierter Funktionen** im Sinne von Cousot und Cousot (POPL'77) (ebenso in Kapitel 8 die Paare aus Datenfluss- und reversem Datenflussanalysefunktional).
- ▶ Die Konkretisierungsfunktion  $A^a$  ist monoton, i.a. aber nicht additiv.
- ▶ Mit den Bezeichnungen aus Kap. 15.2 entsprechen sich  $\alpha$  und  $A$  und  $\gamma$  und  $A^a$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

733/863

# Isomorphie abstrakter Semantiken

## Definition (15.3.4, Isomorphie)

Seien  $\mathcal{S}_1 = (\llbracket \cdot \rrbracket_1, \mathcal{C}_1)$  und  $\mathcal{S}_2 = (\llbracket \cdot \rrbracket_2, \mathcal{C}_2)$  zwei abstrakte Semantiken.

$\mathcal{S}_1$  und  $\mathcal{S}_2$  heißen **isomorph**, in Zeichen  $\mathcal{S}_1 \approx_A \mathcal{S}_2$  oder  $\mathcal{S}_1 \approx \mathcal{S}_2$ , wenn es eine additive und bijektive Abstraktionsfunktion  $A : \mathcal{C}_1 \rightarrow \mathcal{C}_2$  gibt, so dass für alle  $G \in \mathbf{FG}$  gilt:

$$A \circ \llbracket G \rrbracket_1 = \llbracket G \rrbracket_2 \circ A$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Beobachtungsniveau und Beobachtung

## Definition (15.3.5, Beobachtung(sniveau))

Sei  $\mathcal{S}$  eine abstrakte Semantik,  $\Omega$  (“ $\Omega$ ” für Beobachtung) ein vollständiger Halbverband und  $A : \mathcal{C} \rightarrow \Omega$  eine additive und surjektive Funktion.

Dann induziert  $\mathcal{S}$  ein **Semantikfunktional** oder **Verhalten**  $\llbracket \_ \rrbracket_A : \mathbf{FG} \rightarrow (\Omega \rightarrow \Omega)$  auf  $\Omega$  durch

$$\forall G \in \mathbf{FG}. \llbracket G \rrbracket_A =_{df} A \circ \llbracket G \rrbracket \circ A^a$$

Wir bezeichnen diese Situation mit  $\mathcal{S} \rightarrow_A \Omega$  und nennen  $\Omega$  ein **Beobachtungsniveau**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

735/863

## Definition (15.3.6, Modell)

Sei  $\Omega$  ein Beobachtungsniveau und  $\mathcal{S}$  eine abstrakte Semantik mit  $\mathcal{S} \rightarrow_A \Omega$ .

Dann heißt das Paar  $(\mathcal{S}, A)$  ein **Modell von  $\Omega$** .



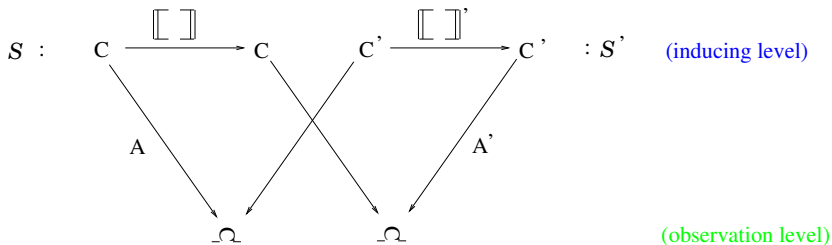
# Beobachtungsäquivalenz

## Definition (15.3.7, Beobachtungsäquivalenz)

Seien  $(\mathcal{S}, A)$  und  $(\mathcal{S}', A')$  zwei Modelle von  $\Omega$ .

Dann heißen  $(\mathcal{S}, A)$  und  $(\mathcal{S}', A')$   $\Omega$ -äquivalent oder **beobachtungsäquivalent** für  $\Omega$ , in Zeichen  $(\mathcal{S}, A) \approx_{\Omega} (\mathcal{S}', A')$  gdw sie dasselbe Verhalten auf  $\Omega$  induzieren, d.h. gdw  $\llbracket \cdot \rrbracket_A = \llbracket \cdot \rrbracket_{A'}$ .

Veranschaulichung:



# Eigenschaften der Relation $\approx_{\Omega}$

## Lemma (15.3.8, Äquivalenzrelation)

*Die Relation  $\approx_{\Omega}$  ist eine Äquivalenzrelation auf der Menge aller Modelle von  $\Omega$ .*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Lokale Optimalität (1)

## Definition (15.3.9, Lokale Optimalität)

Sei  $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$  und  $\mathcal{S}_2 \rightarrow_{A_2} \Omega$ . Dann definieren wir:

1.  $RI(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) =_{df} \{c \in \mathcal{C}_2 \mid \exists G \in \mathbf{FG} \exists c' \in \Omega. c = A_1 \circ \llbracket G \rrbracket_1 \circ A_1^a \circ A_2^a\}(c')\}$
2.  $RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$  bezeichnet die vollständige Halbverbandshülle von  $RI(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$  in  $\mathcal{C}_2$ .
3.  $RS(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) =_{df} (\llbracket \cdot \rrbracket, RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega))$ , wobei  $\llbracket \cdot \rrbracket$  folgendermaßen definiert ist:

$$\forall G \in \mathbf{FG}. \llbracket G \rrbracket =_{df}$$

$$\begin{cases} \llbracket G \rrbracket_2 \mid_{RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)} & \text{falls } RL(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega) \\ \perp & \text{abgeschlossen ist unter } \llbracket \cdot \rrbracket_2 \\ & \text{sonst} \end{cases}$$

# Lokale Optimalität (2)

## Definition (15.3.9, Lokale Optimalität (fgs.))

4.  $\mathcal{S}_2$  heißt **lokal optimal** für  $\mathcal{S}_1$  und  $A$  gdw für alle  $n \in \mathbf{N}$  gilt:

$$A_1 \circ \llbracket n \rrbracket_1 \circ A_1^a = \llbracket n \rrbracket_2$$

# Voll abstrakte Modelle

## Definition (15.3.10, Voll abstrakte Modelle)

Seien  $\mathcal{S}_1$ ,  $\Omega$  und  $A$  mit  $\mathcal{S}_1 \rightarrow_A \Omega$ .

Ein Paar  $(\mathcal{S}_2, A_2)$  mit  $\mathcal{S}_2 \rightarrow_A \Omega$  heißt **voll abstraktes Modell** für  $\mathcal{S}_1$  bezüglich  $\Omega$  gdw eine Abstraktionsfunktion  $A_1$  mit  $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$  existiert, die folgende 4 Eigenschaften erfüllt:

1.  $A = A_2 \circ A_1$
2.  $\mathcal{S}_2 = RS(\mathcal{S}_1, A_1, \mathcal{S}_2, A_2, \Omega)$
3.  $\mathcal{S}_2$  ist lokal optimal für  $\mathcal{S}_1$  und  $A_1$
4.  $\forall c, c' \in RI(\mathcal{S}_1, ID, \mathcal{S}_1, A, \Omega). A_1(c) = A_1(c') \iff \forall G \in \mathbf{LFG}. A \circ \llbracket G \rrbracket_1(c) = A \circ \llbracket G \rrbracket_1(c')$ , wobei  $ID$  die Identität auf dem semantischen Bereich von  $\mathcal{S}_1$  ist.

Wir bezeichnen die Menge aller voll abstrakten Modelle für  $\mathcal{S}_1$ ,  $\Omega$  und  $A$ , die in der Beziehung  $\mathcal{S}_1 \rightarrow_A \Omega$  stehen, mit  $\Phi(\mathcal{S}_1, A, \Omega)$ .

# Existenz und Eindeutigkeit

## Theorem (15.3.11, Existenz und Eindeutigkeit)

Seien  $\mathcal{S}_1$ ,  $\Omega$  und  $A$  mit  $\mathcal{S}_1 \rightarrow_A \Omega$ .

Dann gibt es ein voll abstraktes Modell  $(\mathcal{S}_2, A_2)$  für  $\mathcal{S}_1$  bezüglich  $\Omega$  mit folgender Eindeutigkeitseigenschaft:

$$\Phi(\mathcal{S}_1, A, \Omega) = \{(\mathcal{S}'_2, A'_2) \mid \exists A'. \mathcal{S}_2 \approx_{A'} \mathcal{S}'_2 \wedge A_2 = A'_2 \circ A'\}$$

# Voll abstrakt ist “gut genug”

Theorem (15.3.12, Abschneidetheorem)

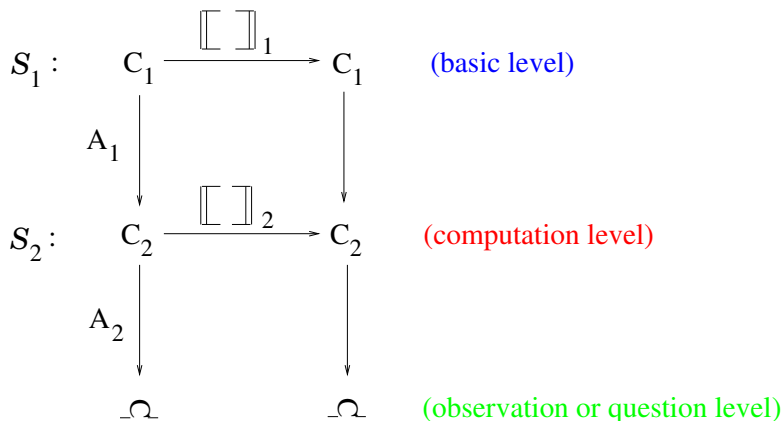
Sei  $(\mathcal{S}_2, A_2) \in \Phi(\mathcal{S}_1, A_2 \circ A_1, \Omega)$  mit  $\mathcal{S}_2 \leq_{A_1} \mathcal{S}_1$ . Dann gilt:

$$\forall G \in \mathbf{FG}. A_1 \circ \llbracket G \rrbracket_1 \circ A_1^a = \llbracket G \rrbracket_2$$

*Insbesondere gilt weiters:*

$$(\mathcal{S}_1, A_2 \circ A_1) \approx_{\Omega} (\mathcal{S}_2, A_2)$$

# Veranschaulichung





# Äquivalenz

## Theorem (15.2.13, Äquivalenz)

Seien  $(\mathcal{S}, A)$  und  $(\mathcal{S}', A')$  zwei Modelle von  $\Omega$ . Dann gilt:

$$(\mathcal{S}, A) \approx_{\Omega} (\mathcal{S}', A') \iff \Phi(\mathcal{S}, A, \Omega) = \Phi(\mathcal{S}', A', \Omega)$$

Interpretation:

- ▶ Zusammen mit dem Existenz- und Eindeigkeitstheorem 15.3.11 und dem Abschneidetheorem 15.3.12 liefert das Äquivalenztheorem 15.3.13, dass voll abstrakte Modelle (bis auf Isomorphie) die “abstraktesten” Repräsentanten ihrer Beobachtungsäquivalenzklasse sind.

# Interpretation und Quintessenz (1)

Zu vorgegebenem Beobachtungsniveau  $\Omega$  und Modell  $(\mathcal{S}, A)$  von  $\Omega$  gibt es ein

- ▶ beobachtungsäquivalentes “abstraktestes” Berechnungsniveau.

Dieses “abstrakteste” Berechnungsniveau ist das bis auf Isomorphie

- ▶ eindeutig bestimmte voll abstrakte Modell.

Das voll abstrakte Modell ist korrekt und zugleich das Gesuchte

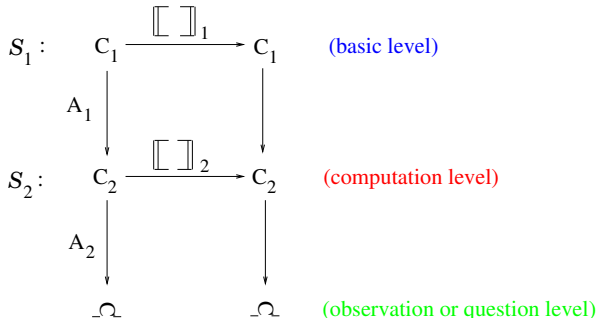
- ▶ vollständige bzw. optimale Modell.

# Interpretation und Quintessenz (2)





Dieses voll abstrakte Modell liegt hierarchisch eingebettet innerhalb des

- ▶ 3-Niveaumodells.

In diesem Sinn ist das 3-Niveaumodell hinreichend allgemein für den nicht auf Hierarchien abstrakter Interpretationen beschränkten Begriff der Beobachtungsäquivalenz.



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (1)

-  Samson Abramsky, Chris Hankin. *An Introduction to Abstract Interpretation*. In *Abstract Interpretation of Declarative Languages*, Samson Abramsky, Chris Hankin (Eds). Prentice Hall, 63-102, 1987.
-  Patrick Cousot. *Methods and Logics for Proving Programs*. In *Handbook of Theoretical Computer Science*, Jan van Leeuwen (Ed.), Elsevier Science Publishers B. V., chapter 15, 841-993, 1990.
-  Patrick Cousot. *Abstract Interpretation*. *ACM Computing Surveys* 28(2):324-328, 1996.
-  Patrick Cousot. *Refining Model-Checking by Abstract Interpretation*. *Autom. Softw. Eng.* 6(1):69-95, 1999.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15




15.1

15.2



15.3

15.4

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (2)

-  Patrick Cousot. *Design of Syntactic Program Transformations by Abstract Interpretation of Semantic Transformations*. In Proceedings of the 17th International Conference on Logic Programming (ICLP 2001), Springer-V., LNCS 2237, 4-5, 2001.
-  Patrick Cousot. *The Verification Grand Challenge and Abstract Interpretation*. In Proceedings of Verified Software: Theories, Tools, Experiments (VSTTE 2005), Springer-V, LNCS 4171, 189-201, 2005.
-  Patrick Cousot. *Verification by Abstract Interpretation*. In Proceedings of the 4th International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI 2003), Springer-V., LNCS 2575, 20-24, 2003.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (3)

-  Patrick Cousot. *Verification by Abstract Interpretation*. In *Verification: Theory and Practice, Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday*. Springer-V., LNCS 2772, 243-268, 2003.
-  Patrick Cousot, Radhia Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In *Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77)*, 238-252, 1977.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1


15.2


15.3


15.4

750/863


# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (4)


 Patrick Cousot, Radhia Cousot. *Systematic Design of Program Analysis Frameworks*. In Conference Record of the 6th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79), 269-282, 1979.

 Patrick Cousot, Radhia Cousot. *Abstract Interpretation Frameworks*. Journal of Logic and Computation 2(4):511-547, 1992.

 Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (5)

 Patrick Cousot, Radhia Cousot. *A Gentle Introduction to Formal Verification of Computer Systems by Abstract Interpretation*. In Logics and Languages for Reliability and Security. NATO Science for Peace and Security - D; Information and Communication Security, Vol. 25, IOS Press, 2010. ISBN 978-1-60750-099-5.

 Patrick Cousot, Radhia Cousot, Laurent Mauborgne. *Theories, Solvers and Static Analysis by Abstract Interpretation*. Journal of the ACM 59(6), Article 31, 56 Seiten, 2012.

 Patrick Cousot, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2





15.3

15.4


752/863




# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (6)

-  Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.
-  Kim Marriot. *Frameworks for Abstract Interpretation*. Acta Informatica 30:103-129, 1993.
-  Flemming Nielson. *A Bibliography on Abstract Interpretations*. ACM SIGPLAN Notices 21:31-38, 1986.
-  Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2nd edition, Springer-V., 2005. (Chapter 1.5, Abstract Interpretation; Chapter 4, Abstract Interpretation)

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 15 (7)

 Bernhard Steffen. *Optimal Run Time Optimization – Proved by a New Look at Abstract Interpretation*. In Proceedings of the 2nd Joint International Conference on the Theory and Practice of Software Development (TAPSOFT'87), Springer-V., LNCS 249, 52-68, 1987.

 Bernhard Steffen. *Optimal Data Flow Analysis via Observational Equivalence*. In Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Springer-V., LNCS 379, 492-502, 1989.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

15.1

15.2

15.3

15.4

# Kapitel 16

## Modellprüfung und DFA

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**Kap. 16**

Kap. 17

L755/863

# Motivation

Das Grundproblem der Modellprüfung.

Gegeben:

- ▶ Ein Modell  $\mathcal{M}$
- ▶ Eine Eigenschaft  $\mathcal{E}$  ausgedrückt als eine Formel  $\phi$

Modellprüfungsfrage:

- ▶ Besitzt Modell  $\mathcal{M}$  Eigenschaft  $\mathcal{E}$ , erfüllt  $\mathcal{M}$  Formel  $\phi$ ?

$$\mathcal{M} \models \phi$$

# Modellprüfer und Modellprüfung

## Modellprüfer:

- ▶ Eine **Methode**, ein **Werkzeug**, das die Modellprüfungsfrage beantwortet.

## Modellprüfung:

- ▶ Ansetzen eines **Modellprüfers MP** auf ein Paar aus **Modell  $\mathcal{M}$**  und **Formel  $\phi$** :
  - ▶  $\mathcal{M} \models_{MP} \phi$  wird **nachgewiesen**:  $\mathcal{M}$  ist bezüglich  $\phi$  **verifiziert** oder  $\phi$  ist für  $\mathcal{M}$  **verifiziert**.
  - ▶  $\mathcal{M} \models_{MP} \phi$  wird **widerlegt**:  $\mathcal{M}$  ist bezüglich  $\phi$  **falsifiziert** oder  $\phi$  ist für  $\mathcal{M}$  **falsifiziert**.

**Zweckmäßig**: Ausgabe eines (minimalen) Gegenbeispiels, das die Verletzung der Formel zeigt (**CEGAR (counter-example-guided abstraction refinement)**-Ansatz).

- ▶  $\mathcal{M} \models_{MP} \phi$  wird **weder nachgewiesen noch widerlegt**:  
Modellprüfer ist **unvollständig** für Modell- und Formelsprache.

# Modellsprachen

## Modellsprachen:

- ▶ Transitionssysteme, Kripke-Strukturen

## und spezielle Ausprägungen:

- ▶ Automaten
- ▶ Flussgraphen
- ▶ (Programm-) Zustandsgraphen
- ▶ ...

## Modelle können sein

- ▶ endlich: Endliche Modellprüfung
- ▶ unendlich: Unendliche Modellprüfung

## Notorisches (und schwierig handhabbares) Problem:

- ▶ Explosion des Zustandsraums: Zustandsraumexplosion

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L758/863

# Formelsprachen

## Formelsprachen:

- ▶ Temporale, modale Logiken (linear time logics, branching time logics)
  - ▶ LTL, CTL, CTL\*
  - ▶  $\mu$ -Kalkül
  - ▶ ...

## Notorisches (und schwierig handhabbares) Problem:

- ▶ Balancierung von **Ausdruckskraft** und **Entscheidbarkeit**, insbesondere **effizienter Entscheidbarkeit**.

# Modaler $\mu$ -Kalkül (1)

...um Rückwärtsmodalitäten erweiterter  $\mu$ -Kalkül:

Syntax:

$$\Phi ::= tt \mid X \mid \Phi \wedge \Phi \mid \neg\Phi \mid \beta \mid [\alpha]\Phi \mid \overline{[\alpha]}\Phi \mid \nu X. \Phi$$

Semantik:

$$\llbracket tt \rrbracket e = \mathcal{S}$$

$$\llbracket X \rrbracket e = e(X)$$

$$\llbracket \Phi_1 \wedge \Phi_2 \rrbracket e = \llbracket \Phi_1 \rrbracket e \wedge \llbracket \Phi_2 \rrbracket e$$

$$\llbracket \neg\Phi \rrbracket e = \mathcal{S} \setminus \llbracket \Phi \rrbracket e$$

$$\llbracket \beta \rrbracket e = \{p \in \mathcal{S} \mid \beta \in \lambda(p)\}$$

$$\llbracket [\alpha]\Phi \rrbracket e = \{p \in \mathcal{S} \mid \forall q \in \text{Succ}_\alpha. q \in \llbracket \Phi \rrbracket e\}$$

$$\llbracket \overline{[\alpha]}\Phi \rrbracket e = \{p \in \mathcal{S} \mid \forall p \in \text{Pred}_\alpha. p \in \llbracket \Phi \rrbracket e\}$$

$$\llbracket \nu X. \Phi \rrbracket e = \bigcup \{S' \subseteq \mathcal{S} \mid S' \subseteq \llbracket \Phi \rrbracket e[S'/X]\}$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L760/863



# Modaler $\mu$ -Kalkül (2)

Abgeleitete Operatoren:

$$ff = \neg tt$$

$$\Phi_1 \vee \Phi_2 = \neg(\neg\Phi_1 \wedge \neg\Phi_2)$$

$$\langle \alpha \rangle \Phi = \neg[\alpha](\neg\Phi)$$

$$\overline{\langle \alpha \rangle} \Phi = \neg\overline{[\alpha]}(\neg\Phi)$$

$$\mu X. \Phi = \nu X. \neg(\Phi[\neg X/X])$$

$$\Phi \succ \Psi = \neg\Phi \vee \Psi$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**Kap. 16**

Kap. 17

761/863

# Modaler $\mu$ -Kalkül (3)

Höher-abstrakte abgeleitete Operatoren:

$$\mathbf{AG} \phi = \nu X. (\phi \wedge [.]X)$$

$$\phi \mathbf{U} \psi = \nu X. (\psi \vee (\phi \wedge [.]X))$$

$$\overline{\mathbf{AG}} \phi = \nu X. (\phi \wedge \overline{[.]X})$$

$$\phi \overline{\mathbf{U}} \psi = \nu X. (\psi \vee (\phi \wedge \overline{[.]X}))$$

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**Kap. 16**

Kap. 17

L762/863

# Analogie DFA – Modellprüfung

## Datenflussanalyse:

DFA-Algorithmus für Eigenschaft  $E$  :

Programme  $\rightarrow$  Menge der  $E$  erfüllenden Programmpunkte

## Modellprüfung:

Modellprüfer : (Modallogische) Formel  $\times$  Modell

$\rightarrow$  Menge der die Formel erfüllenden Zustände

## Intuitiv:

- ▶ Ein DFA-Algorithmus für  $E$  ist ein bezüglich  $E$  partiell ausgewerteter oder bezüglich  $E$  spezialisierter Modellprüfer!

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L763/863

# Anwendung: PREE

Sicherheit (Notwendigkeit der Berechnung):

$$NEC =_{df} (\neg(\text{Mod} \vee \text{end})) \mathbf{U} \text{Used}$$

Frühestheit (Wert kann nicht früher bereitgestellt werden):

$$EAR =_{df} \text{start} \vee \neg([\bar{\cdot}]((\neg(\text{Mod} \vee \text{start})) \bar{\mathbf{U}} (\text{NEC} \wedge \neg \text{Mod})))$$

Berechnungspunkte:

$$OCP =_{df} EAR \wedge NEC$$

## Theorem (16.1, Korrektheit und Optimalität)

*Das Ersetzen der originalen Berechnungen durch neue an den durch **OCP** gegebenen Programmpunkten ist **korrekt** (d.h. semantikerhaltend) und **optimal** (d.h. mindestens so gut wie jede andere korrekte Platzierung der Berechnungen).*

# Zusammenfassung (1)

Diese Charakterisierung des Zusammenhangs von **DFA** und **Modellprüfung** und die **PREE-Anwendung** geht zurück auf:

- ▶ Bernhard Steffen. **Data Flow Analysis as Model Checking**. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.
- ▶ Bernhard Steffen. **Generating Data Flow Analysis Algorithms from Modal Specifications**. International Journal on Science of Computer Programming 21:115-139, 1993.

## Zusammenfassung (2)

...ist aufgegriffen worden von:

- ▶ David A. Schmidt. **Data Flow is Model Checking of Abstract Interpretations**. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.

...und hat in der Folge geführt zu:

- ▶ David A. Schmidt, Bernhard Steffen. **Program Analysis as Model Checking of Abstract Interpretations**. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16




Kap. 17

L766/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (1)





-  Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Peit, Laure Petrucci, Philippe Schnoebelen with Pierre McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.
-  Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*. MIT Press, 2001.
-  Edmund M. Clarke, H. Schlingloff. *Model Checking*. In *Handbook of Automated Deduction*, A. Robinson, A. Voronkov (Hrsg.), Vol. II, Elsevier, 1635-1790, 2000.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (2)





-  Patrick Cousot, Radhia Cousot. *Temporal Abstract Interpretation*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 12-25, 2000.
-  E. Allen Emerson. *Temporal and Modal Logic*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.
-  Orna Grumberg, Helmut Veith. *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.
-  George E. Hughes, Max J. Cresswell. *An Introduction to Modal Logic*. Methuan, 1968.






# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (3)

-  George E. Hughes, Max J. Cresswell. *A Companion to Modal Logic*. Methuan, 1986.
-  George E. Hughes, Max J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.
-  Fred Kröger, Stephan Merz. *Temporal Logic and State Systems*. Springer-V., 2008. (Chapter 3, Extensions of Linear Time Logic; Chapter 5, First-Order Linear Time Logic; Chapter 10, Other Temporal Logics; Chapter 11, System Verification by Model Checking)
-  Janusz Laski, William Stanley. *Software Verification and Analysis: An Integrated, Hands-On Approach*. Springer-V., 2009.



# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (4)

-  Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008. (Chapter 20.2.2, Temporal, Modal, and Dynamic Logics)
-  Markus Müller-Olm, David A. Schmidt, Bernhard Steffen. *Model-Checking: A Tutorial Introduction*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 330-354, 1999.
-  Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.
-  Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, 2012.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (5)

-  David A. Schmidt. *Data Flow Analysis is Model Checking of Abstract Interpretations*. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.
-  David A. Schmidt, Bernhard Steffen. *Program Analysis as Model Checking of Abstract Interpretations*. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.
-  Bernhard Steffen. *Data Flow Analysis as Model Checking*. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 16 (6)

-  Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. International Journal on Science of Computer Programming 21:115-139, 1993.
-  Bernhard Steffen. *Property-Oriented Expansion*. In Proceedings of the 3rd Static Analysis Symposium (SAS'96), Springer-V., LNCS 1145, 22-41, 1996.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**Kap. 16**

Kap. 17

L772/863

# Teil V

## Abschluss und Ausblick

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

**Kap. 16**

Kap. 17

# Kapitel 17

## Resümee und Perspektiven

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

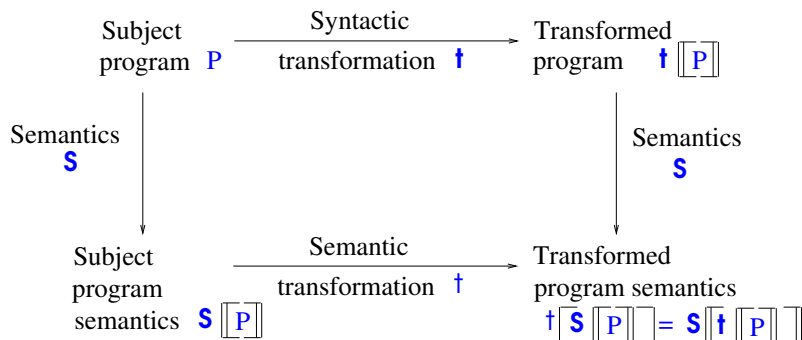
Kap. 15

Kap. 16

**Kap. 17**

# Analyse, Verifikation und Transformation

...bewiesen (beweisbar) korrekt und vollständig/optimal in einem einheitlichen Rahmen:



(aus Cousot&Cousot, POPL 2002)

# Wichtig für verschiedenste Gebiete, darunter:

- ▶ Optimierende Übersetzer (Performanz, Energie,...)
- ▶ Software-Verifikation
- ▶ Übersetzer-Verifikation, Betriebssystem-Verifikation,...
- ▶ Software
  - ▶ -Spezifikation, -Analyse, -Validierung, -Generierung (insbesondere auch modellgetriebene Spezifikation, Analyse, Verifikation, Generierung, Testung,...)
  - ▶ -Verstehen
    - ▶ Refaktorisierung
    - ▶ (Re-)engineering, Reverse Engineering
    - ▶ Dokumentation
  - ▶ -(Kunden)anpassung, -spezialisierung (customization)
- ▶ Safety and Security (security policy enforcement,...)
- ▶ Datenschürfung und -ausbeutung (data mining)
- ▶ Hardware-Verifikation
- ▶ ...
- ▶ Grüne Informationstechnologie



# All dies

- ▶ Statisch und dynamisch.
- ▶ Auf Programm-Ebene im Kleinen.
- ▶ Auf System-Ebene im Großen
  - ▶ Systeme von Systemen
  - ▶ Hard- und Software-Systeme von Systemen
    - ▶ Eingebettete Systeme
    - ▶ Cyberphysikalische Systeme
    - ▶ Verteilte Systeme: Service-orientierte Systeme, Wolken-Systeme, Mehrkern-(HW)-Systeme,...
    - ▶ Echtzeit-Systeme
    - ▶ ...
- ▶ auf Spezifikations-, Modellierungs-, Programmier-, Zwischensprach- und Binärcode-Ebene.

# Unverzichtbar

## Rigorese Fundierung

- ▶ Formale Methoden

## Wirksame Werkzeug-Unterstützung

- ▶ Hochskalierende “Denk”-Werkzeuge
  - ▶ Vollautomatisch
    - ▶ Knopfdruckanalyse, -verifikation und -transformation
  - ▶ Halbautomatisch
    - ▶ Interaktive, benutzergeleitete Analyse, Verifikation und Transformation
- ▶ Orchestrierung und geordnetes Zusammenspiel und -wirkung über Methodengrenzen hinweg (z.B. Abstrakte Interpretation, Modellprüfung, Theorembeweisen,...)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L778/863

# Wichtige Konferenzen und Zeitschriften (1)

- ▶ Annual **International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI) Series**, Springer-V., LNCS series, since 2000.
- ▶ Annual **International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)**, Springer-V., LNCS series, since 1995.
- ▶ Annual **International Conference on Computer-Aided Verification (CAV) Series**, Springer-V., LNCS series, since 1989.
- ▶ Annual **International Conference on Software Testing, Verification and Validation (ICST) Series**, IEEE, since 2008.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L779/863

## Wichtige Konferenzen und Zeitschriften (2)

- ▶ Annual **International Symposium on Formal Methods (FM)** Series, Springer-V., LNCS series, since 1995.
- ▶ Biennial **International Symposium on Leveraging Applications of Formal Methods, Verification, and Validation (ISoLA)** Series, Springer-V., LNCS series, since 2004.
- ▶ **International Journal on Software Tools for Technology Transfer (STTT)**, Springer-V, since 1999.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14


Kap. 15


Kap. 16

Kap. 17

L780/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (1)

 Julien Bertrane, Patrick Cousot, Radhia Cousot, J r me Feret, Laurent Mauborgne, Antoine Min , Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.

 Julien Bertrane, Patrick Cousot, Radhia Cousot, J r me Feret, Laurent Mauborgne, Antoine Min , Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14


Kap. 15

Kap. 16



Kap. 17

L781/863

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (2)

-  Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. Communications of the ACM 56(2):82-90, 2013.
-  Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.
-  Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.

# Vertiefende und weiterführende Leseempfehlungen für Kapitel 17 (3)

-  Ashish Tiwari, Sumit Gulwani. *Static Program Analysis Using Theorem Proving*. In Proceedings of the 21st Conference on Automated Deduction (CADE-21), LNCS 4603, Springer-V., 147-166, 2007.
-  Daniel Weise. *Static Analysis of Mega-Programs (Invited Paper)*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 300-302, 1999.

# Literatur

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Literatur



# Literaturhinweise und Leseempfehlungen

...zum vertiefenden und weiterführenden Selbststudium.

- ▶ I Lehrbücher
- ▶ II Artikel, Dissertationen, Sammelbände

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14






Kap. 15

Kap. 16

Kap. 17

Literatur

# I Lehrbücher (1)

-  Alfred V. Aho, Monica S. Lam, Ravi Sethi, Jeffrey D. Ullman. *Compilers: Principles, Techniques, & Tools*. 2. Auflage, Addison-Wesley, 2007.
-  Randy Allen, Ken Kennedy. *Optimizing Compilers for Modern Architectures*. Morgan Kaufman Publishers, 2002.
-  Krzysztof R. Apt, Ernst-Rüdiger Olderog. *Programmverifikation – Sequentielle, parallele und verteilte Programme*. Springer-V., 1994.
-  Krzysztof R. Apt, Frank S. de Boer, Ernst-Rüdiger Olderog. *Verification of Sequential and Concurrent Programs*. 3. Auflage, Springer-V., 2009.
-  Mordechai Ben-Ari. *Mathematical Logic for Computer Science*. 2. Auflage, Springer-V., 2001.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

L7867862

# I Lehrbücher (2)

-  Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Peit, Laure Petrucci, Philippe Schnoebelen with Pierre McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-V., 2001.
-  Edmund M. Clarke, Orna Grumberg, Doron Peled. *Model Checking*. MIT Press, 2001.
-  Keith D. Cooper, Linda Torczon. *Engineering a Compiler*. Morgan Kaufman Publishers, 2004.
-  Jaco W. De Backer. *Mathematical Theory of Program Correctness*. Prentice-Hall, 1980.
-  Gilles Dowek. *Principles of Programming Languages*. Springer-V, 2009.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14







Kap. 15

Kap. 16

Kap. 17

L787863

# I Lehrbücher (3)

-  Michael J.C. Gordon. *The Denotational Description of Programming Languages*. Springer-V., 1979.
-  Matthew S. Hecht. *Flow Analysis of Computer Programs*. Elsevier, North-Holland, 1977.
-  George E. Hughes, Max J. Cresswell. *An Introduction to Modal Logic*. Methuan, 1968.
-  George E. Hughes, Max J. Cresswell. *A Companion to Modal Logic*. Methuan, 1986.
-  George E. Hughes, Max J. Cresswell. *A New Introduction to Modal Logic*. Routledge, 1996.
-  Fred Kröger, Stephan Merz. *Temporal Logic and State Systems*. Springer-V., 2008.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

L7887862

# I Lehrbücher (4)

-  Janusz Laski, William Stanley. *Software Verification and Analysis*. Springer-V., 2009.
-  Jacques Loeckx, Kurt Sieber. *The Foundations of Program Verification*. Wiley, 1984.
-  Robert Lover. *Elementary Logic for Software Development*. Springer-V., 2008.
-  Robert Morgan. *Building an Optimizing Compiler*. Digital Press, 1998.
-  Stephen S. Muchnick. *Advanced Compiler Design Implementation*. Morgan Kaufman Publishers, 1997.
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14






Kap. 15

Kap. 16

Kap. 17

L7097862

# I Lehrbücher (5)

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007.
-  Flemming Nielson, Hanne Riis Nielson, Chris Hankin. *Principles of Program Analysis*. 2. Auflage, Springer-V., 2005.
-  Doron A. Peled. *Software Reliability Methods*. Springer-V., 2001.
-  Joseph E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. MIT Press, 1981.
-  Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, 1993.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

L707862

## II Artikel, Dissertationen, Sammelbände (1)

-  Samson Abramsky, Chris Hankin. *An Introduction to Abstract Interpretation*. In *Abstract Interpretation of Declarative Languages*, Samson Abramsky, Chris Hankin (Eds). Prentice Hall, 63-102, 1987.
-  Gagan Agrawal. *Demand-driven Construction of Call Graphs*. In *Proceedings of the 9th International Conference on Compiler Construction (CC 2000)*, Springer-V., LNCS 1781, 125-140, 2000.
-  Bowen Alpern, Mark N. Wegman, F. Ken Zadeck. *Detecting Equality of Variables in Programs*. In *Conference Record of the 15th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'88)*, 1-11, 1988.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

L7017862

## II Artikel, Dissertationen, Sammelbände (2)

-  Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part 1*. ACM Transactions on Programming Languages and Systems 3(4):431-483, 1981.
-  Krzysztof R. Apt. *Ten Years of Hoare's Logic: A Survey – Part II: Nondeterminism*. Theoretical Computer Science 28(1-2):83-109, 1984.
-  Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part I - Exhaustive Analysis*. Acta Informatica 10(3):245-264, 1978.
-  Wayne A. Babich, Mehdi Jazayeri. *The Method of Attributes for Data Flow Analysis: Part II - Demand Analysis*. Acta Informatica 10(3):265-272, 1978.



## II Artikel, Dissertationen, Sammelbände (3)

-  Bernhard Beckert, Reiner Hähnle, Peter H. Schmitt (Hrsg.). *Verification of Object-Oriented Software: The KeY Approach*. LNCS 4334, Springer-V., 2007.
-  Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis and Verification of Aerospace Software by Abstract Interpretation*. In Proceedings AIAA Infotech@Aerospace (AIAA I@A 2010), AIAA-2010-3385, American Institute of Aeronautics and Astronautics, 1-38, April 2010.
-  Julien Bertrane, Patrick Cousot, Radhia Cousot, Jèrôme Feret, Laurent Mauborgne, Antoine Minè, Xavier Rival. *Static Analysis by Abstract Interpretation of Embedded Critical Software*. ACM Software Engineering Notes 36(1):1-8, 2011.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14



Kap. 15

Kap. 16

Kap. 17

L7937862

## II Artikel, Dissertationen, Sammelbände (4)

-  Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, Dawson Engler. *A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World*. Communications of the ACM 53(2):66-75, 2010.
-  Ras Bodik, Rajiv Gupta. *Partial Dead Code Elimination using Slicing Transformations*. In Proceedings of the ACM SIGPLAN'97 Conference on Programming Language Design and Implementation (PLDI'97), ACM SIGPLAN Notices 32(6):159-170, 1997.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

794/7862

## II Artikel, Dissertationen, Sammelbände (5)

-  Ras Bodík, Rajiv Gupta, Vivek Sarkar. *ABCD: Eliminating Array Bounds Check on Demand*. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'00), ACM SIGPLAN Notices 35(5):321-333, 2000.
-  Cristian Cadar, Koushik Sen. *Symbolic Execution for Software Testing: Three Decades Later*. *Communications of the ACM* 56(2):82-90, 2013.
-  Edmund M. Clarke. *Programming Language Constructs for which it is Impossible to Obtain Good Hoare Axiom Systems*. *Journal of the ACM* 26(1):129-147, 1979.
-  Edmund M. Clarke, Stephen M. German, Joseph Y. Halpern. *Effective Axiomatizations of Hoare Logics*. *Journal of the ACM* 30(1):612-636, 1983.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

795/862

## II Artikel, Dissertationen, Sammelbände (6)

-  Edmund M. Clarke, Orna Grumberg, David E. Long. *Model Checking and Abstraction*. ACM Transactions on Programming Languages and Systems 16(5):1512-1542, 1994.
-  Edmund M. Clarke, H. Schlingloff. *Model Checking*. In Handbook of Automated Deduction, A. Robinson, A. Voronkov (Hrsg.), Vol. II, Elsevier, 1635-1790, 2000.
-  Stephen A. Cook. *Soundness and Completeness of an Axiom System for Program Verification*. SIAM Journal on Computing 7(1):70-90, 1978.
-  Patrick Cousot. *Methods and Logics for Proving Programs*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Ed.), Elsevier Science Publishers B. V., Chapter 15, 841-993, 1990.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

L7967862

## II Artikel, Dissertationen, Sammelbände (7)

-  Patrick Cousot. *Abstract Interpretation*. ACM Computing Surveys 28(2):324-328, 1996.
-  Patrick Cousot. *Refining Model-Checking by Abstract Interpretation*. Autom. Softw. Eng. 6(1):69-95, 1999.
-  Patrick Cousot. *Design of Syntactic Program Transformations by Abstract Interpretation of Semantic Transformations*. In Proceedings of the 17th International Conference on Logic Programming (ICLP 2001), Springer-V., LNCS 2237, 4-5, 2001.
-  Patrick Cousot. *The Verification Grand Challenge and Abstract Interpretation*. In Proceedings of Verified Software: Theories, Tools, Experiments (VSTTE 2005), Springer-V, LNCS 4171, 189-201, 2005.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

797/863

## II Artikel, Dissertationen, Sammelbände (8)

-  Patrick Cousot. *Verification by Abstract Interpretation*. In Proceedings of the 4th International Conference on Verification, Model-Checking, Abstract Interpretation (VMCAI 2003), Springer-V., LNCS 2575, 20-24, 2003.
-  Patrick Cousot. *Verification by Abstract Interpretation*. In Verification: Theory and Practice, Essays dedicated to Zohar Manna on the Occasion of His 64th Birthday. Springer-V., LNCS 2772, 243-268, 2003.
-  Patrick Cousot, Radhia Cousot. *Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 238-252, 1977.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

798/862

## II Artikel, Dissertationen, Sammelbände (9)

-  Patrick Cousot, Radhia Cousot. *Constructive Versions of Tarski's Fixed Point Theorems*. Pacific Journal of Mathematics 82(1):43-57, 1979.
-  Patrick Cousot, Radhia Cousot. *Systematic Design of Program Analysis Frameworks*. In Conference Record of the 6th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79), 269-282, 1979.
-  Patrick Cousot, Radhia Cousot. *Abstract Interpretation Frameworks*. Journal of Logic and Computation 2(4):511-547, 1992.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

799/862

## II Artikel, Dissertationen, Sammelbände (10)

-  Patrick Cousot, Radhia Cousot. *Temporal Abstract Interpretation*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 12-25, 2000.
-  Patrick Cousot, Radhia Cousot. *Systematic Design of Program Transformation Frameworks by Abstract Interpretation*. In Conference Record of the 29th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2002), 178-190, 2002.
-  Patrick Cousot, Radhia Cousot. *A Gentle Introduction to Formal Verification of Computer Systems by Abstract Interpretation*. In Logics and Languages for Reliability and Security. NATO Science for Peace and Security - D; Information and Communication Security, Vol. 25, IOS Press, 2010. ISBN 978-1-60750-099-5.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Seite 7/862



## II Artikel, Dissertationen, Sammelbände (11)

-  Patrick Cousot, Radhia Cousot, Laurent Mauborgne. *Theories, Solvers and Static Analysis by Abstract Interpretation*. Journal of the ACM 59(6), Article 31, 56 Seiten, 2012.
-  Patrick Cousot, Michael Monerau. *Probabilistic Abstract Interpretation*. In Proceedings 21st Symposium on Programming (ESOP 2012), Springer-V., LNCS 7211, 169-193, 2012.
-  Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Ken Zadeck. *An Efficient Method of Computing Static Single Assignment Form*. In Conference Record of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'89), 25-35, 1989.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16




Kap. 17

Seite 78/82

## II Artikel, Dissertationen, Sammelbände (12)

-  Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, F. Ken Zadeck. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*. ACM Transactions on Programming Languages and Systems 13(4):451-490, 1991.
-  D. M. Dhamdhere. *Register Assignment using Code Placement Techniques*. Journal of Computer Languages 13(2):75-93, 1988.
-  D. M. Dhamdhere. *A usually linear Algorithm for Register Assignment using Edge Placement of Load and Store Instructions*. Journal of Computer Languages 15(2):83-94, 1990.

## II Artikel, Dissertationen, Sammelbände (13)

-  D. M. Dhamdhere. *Practical Adaptation of the Global Optimization Algorithm of Morel and Renvoise*. ACM Transactions on Programming Languages and Systems 13(2):291-294, 1991. Technical Correspondence.
-  Evelyn Duesterwald. *A Demand-driven Approach for Efficient Interprocedural Data-Flow Analysis*. PhD thesis, University of Pittsburgh, 1996.
-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *Demand-driven Computation of Interprocedural Data Flow*. In Conference Record of the 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'95), 37-48, 1995.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Los 7862

## II Artikel, Dissertationen, Sammelbände (14)

-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Demand-driven Analyzer for Data Flow Testing at the Integration Level*. In Proceedings of the IEEE Conference on Software Engineering (CoSE'96), 575-586, 1996.
-  Evelyn Duesterwald, Rajiv Gupta, Mary Lou Soffa. *A Practical Framework for Demand-driven Interprocedural Data Flow Analysis*. ACM Transactions on Programming Languages and Systems 19(6):992-1030, 1997.
-  E. Allen Emerson. *Temporal and Modal Logic*. In Handbook of Theoretical Computer Science, Jan van Leeuwen (Hrsg.), Elsevier, 995-1072, 1990.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Letzte

## II Artikel, Dissertationen, Sammelbände (15)

-  L. Feigen, D. Klappholz, R. Casazza, X. Xue. *The Revival Transformation*. In Conference Record of the 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'94), 1994.
-  Robert W. Floyd. *Assigning Meaning to Programs*. In Proceedings of Symposium on Applied Mathematics, Mathematical Aspects of Computer Science, American Mathematical Society, New York, 19:19-32, 1967.
-  Alfons Geser, Jens Knoop, Gerald Lüttgen, Oliver Rüthing, Bernhard Steffen. *Non-monotone Fixpoint Iterations to Resolve Second Order Effects*. In Proceedings of the 6th International Conference on Compiler Construction (CC'96), Springer-V., LNCS 1060, 106-120, 1996.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Los 7862

## II Artikel, Dissertationen, Sammelbände (16)

-  Orna Grumberg, Helmut Veith. *25 Years of Model Checking: History, Achievements, Perspectives*. Springer-V., LNCS 5000, 2008.
-  Reiner Hähnle, Richard Bubel. *A Hoare-Style Calculus with Explicit State Updates*. Handout in the course Program Verification at the Department of Computer Science at the Chalmers University of Technology, 19 pages. [www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf](http://www.iti.uni-karlsruhe.de/~key/download/hoare/students.pdf)
-  Nevin Heintze, Joxan Jaffar, Răzvan Voicu. *A Framework for Combining Analysis and Verification*. In Conference Record of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL 2000), 26-39, 2000.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

Lösung

## II Artikel, Dissertationen, Sammelbände (17)

-  Charles A.R. Hoare. *An Axiomatic Basis for Computer Programming*. Communications of the ACM 12(19):576-580, 583, 1969.
-  Susan Horwitz, Thomas Reps, Mooly Sagiv. *Demand Interprocedural Data Flow Analysis*. In Proceedings of the 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-3), 104-115, 1995.
-  John Hughes, John Launchbury. *Reversing Abstract Interpretations*. In Proceedings of the 4th European Symposium on Programming (ESOP'92), Springer-V., LNCS 582, 269-286, 1992.
-  John Hughes, John Launchbury. *Reversing Abstract Interpretations*. Science of Computer Programming 22:307-326, 1994.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17




Lösung

## II Artikel, Dissertationen, Sammelbände (18)

-  Tudor Jebelean, Laura Kovács, Nikolaj Popov. *Experimental Program Verification in the Theorema System*. In Proceedings of the International Symposium on Leveraging Applications of Formal Methods (ISoLA 2004), 92-99, 2004. [www.risc.jku.at/publications/download/risc\\_2243/KoPoJeb.pdf](http://www.risc.jku.at/publications/download/risc_2243/KoPoJeb.pdf)
-  John B. Kam, Jeffrey D. Ullman. *Monotone Data Flow Analysis Frameworks*. *Acta Informatica* 7:305-317, 1977.
-  Gary A. Kildall. *A Unified Approach to Global Program Optimization*. In Conference Record of the 1st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'73), 194-206, 1973.



## II Artikel, Dissertationen, Sammelbände (19)

-  Raimund Kirner, Jens Knoop, Adrian Prantl, Markus Schordan, Albrecht Kadlec. *Beyond Loop Bounds: Comparing Annotation Languages for Worst-Case Execution Time Analysis*. *Journal of Software and Systems Modeling* 10(3):411-437, Springer-V., 2011.
-  Jens Knoop. *From DFA-frameworks to DFA-generators: A Unifying Multiparadigm Approach*. In *Proceedings of the 5th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, Springer-V., LNCS 1579, 360-374, 1999.
-  Jens Knoop. *Demand-driven Analysis of Explicitly Parallel Programs: An Approach based on Reverse Data-Flow Analysis*. In *Proceedings of the 9th International Workshop on Compilers for Parallel Computers (CPC 2001)*, 151-162, 2001.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Seite 7/862

## II Artikel, Dissertationen, Sammelbände (20)

-  Jens Knoop. *Data-Flow Analysis for Hot-Spot Program Optimization*. In Proceedings of the 14th Biennial Workshop on “Programmiersprachen und Grundlagen der Programmierung” (KPS 2007). Bericht A-07-07 der Institute für Mathematik und Informatik, Universität Lübeck, Germany, 124-131, 2007.
-  Jens Knoop, Dirk Koschützki, Bernhard Steffen. *Basic-block Graphs: Living Dinosaurs?* In Proceedings of the 7th International Conference on Compiler Construction (CC'98), Springer-V., LNCS 1383, 65 - 79, 1998.
-  Jens Knoop, Eduard Mehofer. *Distribution Assignment Placement: Effective Optimization of Redistribution Costs*. IEEE Transactions on Parallel and Distributed Systems 13(6):628-647, 2002.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Länge

## II Artikel, Dissertationen, Sammelbände (21)

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Lazy Code Motion*. In Proceedings of the ACM SIGPLAN'92 Conference on Programming Language Design and Implementation (PLDI'92), ACM SIGPLAN Notices 27(7):224-234, 1992.
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Partial Dead Code Elimination*. In Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation (PLDI'94), ACM SIGPLAN Notices 29(6):147-158, 1994.
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Optimal Code Motion: Theory and Practice*. ACM Transactions on Programming Languages and Systems 16(4):1117-1155, 1994.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Lösung

## II Artikel, Dissertationen, Sammelbände (22)

-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *The Power of Assignment Motion*. In Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation (PLDI'95), ACM SIGPLAN Notices 30(6):233-245, 1995.
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Code Motion and Code Placement: Just Synonyms?* In Proceedings of the 7th European Symposium on Programming (ESOP'98), Springer-V., LNCS 1381, 154-169, 1998.
-  Jens Knoop, Oliver Rüthing. *Constant Propagation on the Value Graph: Simple Constants and Beyond*. In Proceedings of the 9th International Conference on Compiler Construction (CC 2000), Springer-V., LNCS 1781, 94-109, 2000.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Seite 78/82

## II Artikel, Dissertationen, Sammelbände (23)

-  Jens Knoop, Oliver Rüthing. *Constant Propagation on Predicated Code*. Journal of Universal Computer Science 9(8):829-850, 2003. (special issue devoted to SBLP'03).
-  Jens Knoop, Oliver Rüthing, Bernhard Steffen. *Retrospective: Lazy Code Motion*. In "20 Years of the ACM SIGPLAN Conference on Programming Language Design and Implementation (1979 - 1999): A Selection", ACM SIGPLAN Notices 39(4):460-461&462-472, 2004.
-  Jens Knoop, Bernhard Steffen. *The Interprocedural Coincidence Theorem*. In Proceedings of the 4th International Conference on Compiler Construction (CC'92), Springer-V., LNCS 641, 125-140, 1992.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14



Kap. 15

Kap. 16

Kap. 17

8137862

## II Artikel, Dissertationen, Sammelbände (24)

-  Jens Knoop, Bernhard Steffen, Jürgen Vollmer. *Parallelism for Free: Efficient and Optimal Bitvector Analyses for Parallel Programs*. *ACM Transactions on Programming Languages and Systems* 18(3):268-299, 1996.
-  Laura Kovács, Tudor Jebelean. *Practical Aspects of Imperative Program Verification using Theorema*. In *Proceedings of the 5th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2003)*, 317-320, 2003. [www.risc.jku.at/publications/download/risc\\_464/synasc03.pdf](http://www.risc.jku.at/publications/download/risc_464/synasc03.pdf)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16





Kap. 17

Länge: 7862

## II Artikel, Dissertationen, Sammelbände (25)

-  Laura Kovács, Tudor Jebelean. *Generation of Invariants in Theorema*. In Proceedings of the 10th International Symposium of Mathematics and its Applications, 407-415, 2003. [www.risc.jku.at/publications/download/risc\\_2053/2003-11-06-A.pdf](http://www.risc.jku.at/publications/download/risc_2053/2003-11-06-A.pdf)
-  Jean-Louis Lassez, V.L. Nguyen, Liz Sonenberg. *Fixed Point Theorems and Semantics: A Folk Tale*. Information Processing Letters 14(3):112-116, 1982.
-  Yuan Lin, David A. Padua. *Demand-driven Interprocedural Array Property Analysis*. In Proceedings of the 12th International Conference on Languages and Compilers for Parallel Computing (LCPC'99), Springer-V., LNCS, 1999.

## II Artikel, Dissertationen, Sammelbände (26)

-  Björn Lisper, Andreas Ermedahl, Dietmar Schreiner, Jens Knoop, Peter Gliwa. *Practical Experiences of Applying Source-level WCET Flow Analysis to Industrial Code*. Journal of Software Tools for Technology Transfer (STTT) 15(1):53-63, Springer-V., 2013.
-  Thomas J. Marlowe, Barbara G. Ryder. *Properties of Data Flow Frameworks*. Acta Informatica 20:121-163, 1990.
-  Kim Marriot. *Frameworks for Abstract Interpretation*. Acta Informatica 30:103-129, 1993.
-  Steve P. Miller, Michael W. Whalen, Darren D. Cofer. *Software Model Checking Takes Off*. Communications of the ACM 53(2):58-64, 2010.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Länge: 7862



## II Artikel, Dissertationen, Sammelbände (27)

-  Ronald J. Mintz, Gerald A. Fisher, Micha Sharir. *The Design of a Global Optimizer*. In Proceedings of the ACM SIGPLAN'79 Symposium on Compiler Construction (SoCC'79), ACM SIGPLAN Notices 14(8):226-234, 1979.
-  Markus Müller-Olm, David A. Schmidt, Bernhard Steffen. *Model-Checking: A Tutorial Introduction*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 330-354, 1999.
-  Flemming Nielson. *A Bibliography on Abstract Interpretations*. ACM SIGPLAN Notices 21:31-38, 1986.
-  David von Oheimb. *Hoare Logic for Java in Isabelle/HOL*. Concurrency and Computation: Practice and Experience 13(13):1173-1214, 2001.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14





Kap. 15

Kap. 16

Kap. 17

Lösung

## II Artikel, Dissertationen, Sammelbände (28)

-  Ernst-Rüdiger Olderog. *Correctness of Programs with Pascal-like Procedures without Global Variables*. Theoretical Computer Science 30(1):49-90, 1984.
-  Ernst-Rüdiger Olderog, Bernhard Steffen. *Formale Semantik und Programmverifikation*. In Informatik-Handbuch, Peter Rechenberg, Gustav Pomberger (Hrsg.), Carl Hanser Verlag, 4. Auflage, 145-166, 2006.
-  Ernst-Rüdiger Olderog, Reinhard Wilhelm. *Turing und die Verifikation*. Informatik Spektrum 35(4):271-279, 2012.
-  Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981, reprinted 1991.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Lösung 7/8/9

## II Artikel, Dissertationen, Sammelbände (29)

-  Gordon D. Plotkin. *An Operational Semantics for CSP*. In Proceedings of TC-2 Working Conference on Formal Description of Programming Concepts II, Dines Bjørner (Ed.), North-Holland, Amsterdam, 1982.
-  Gordon D. Plotkin. *A Structural Approach to Operational Semantics*. Journal of Logic and Algebraic Programming 60-61:17-139, 2004.
-  Peter Puschner, Daniel Prokesch, Benedikt Huber, Jens Knoop, Stefan Hepp, Gernot Gebhard. *The T-CREST Approach of Compiler and WCET-Analysis Integration*. In Proceedings of the 9th International Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2013), June 2013, to appear.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Seite 78/82

## II Artikel, Dissertationen, Sammelbände (30)

-  John H. Reif, Harry R. Lewis. *Symbolic Evaluation and the Global Value Graph*. In Conference Record of the 4th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77), 104-118, 1977.
-  Thomas Reps. *Solving Demand Versions of Interprocedural Analysis Problems*. In Proceedings of the 5th International Conference on Compiler Construction (CC'95), Springer-V., LNCS 786, 389-403, 1994.
-  Thomas Reps. *Demand Interprocedural Program Analysis using Logic Databases*. In Applications of Logic Databases, R. Ramakrishnan (Ed.), Kluwer Academic Publishers, 1994.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

LNCS 786

## II Artikel, Dissertationen, Sammelbände (31)

-  Dirk Richter. *Programmanalysen zur Verbesserung der Softwaremodellprüfung*. Dissertation, Universität Halle-Wittenberg, 2012.
-  F. Robert. *Convergence locale d'itérations chaotiques non linéaires*. Technical Report 58, Laboratoire d'Informatique, U.S.M.G., Grenoble, France, Dec. 1976.
-  Oliver Rüthing, Jens Knoop, Bernhard Steffen. *Detecting Equalities of Variables: Combining Efficiency with Precision*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 232-247, 1999.
-  Oliver Rüthing, Markus Müller-Olm. *On the Complexity of Constant Propagation*. In Proceedings of the 10th European Symposium on Programming (ESOP 2001), Springer-V., LNCS 2028, 190-205, 2001.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Lösung

## II Artikel, Dissertationen, Sammelbände (32)

-  David A. Schmidt. *Data Flow Analysis is Model Checking of Abstract Interpretations*. In Conference Record of the 25th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'98), 38-48, 1998.
-  David A. Schmidt, Bernhard Steffen. *Program Analysis as Model Checking of Abstract Interpretations*. In Proceedings of the 5th Static Analysis Symposium (SAS'98), Springer-V., LNCS 1503, 351-380, 1998.
-  Mary Lou Soffa. *Tutorial: Techniques to improve the Scalability and Precision of Data Flow Analysis*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 355-356, 1999.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

8227862

## II Artikel, Dissertationen, Sammelbände (33)

-  Bernhard Steffen. *Optimal Run Time Optimization – Proved by a New Look at Abstract Interpretation*. In Proceedings of the 2nd Joint International Conference on the Theory and Practice of Software Development (TAPSOFT'87), Springer-V., LNCS 249, 52-68, 1987.
-  Bernhard Steffen. *Optimal Data Flow Analysis via Observational Equivalence*. In Proceedings of the 14th International Symposium on Mathematical Foundations of Computer Science (MFCS'89), Springer-V., LNCS 379, 492-502, 1989.
-  Bernhard Steffen. *Data Flow Analysis as Model Checking*. In Proceedings of the International Conference on Theoretical Aspects of Computer Software (TACS'91), Springer-V., LNCS 526, 346-365, 1991.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

8237862

## II Artikel, Dissertationen, Sammelbände (34)

-  Bernhard Steffen. *Generating Data Flow Analysis Algorithms from Modal Specifications*. *International Journal on Science of Computer Programming* 21:115-139, 1993.
-  Bernhard Steffen. *Property-Oriented Expansion*. In *Proceedings of the 3rd Static Analysis Symposium (SAS'96)*, Springer-V., LNCS 1145, 22-41, 1996.
-  Bernhard Steffen, Jens Knoop. *Finite Constants: Characterizations of a New Decidable Set of Constants*. *Theoretical Computer Science* 80(2):303-318, 1991.
-  Munehiro Takimoto, Kenichi Harada. *Effective Partial Redundancy Elimination based on Extended Value Graph*. *Information Processing Society of Japan* 38(11):2237-2250, 1990.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15




Kap. 16

Kap. 17

Lehrstuhl



## II Artikel, Dissertationen, Sammelbände (35)

-  Munehiro Takimoto, Kenichi Harada. *Partial Dead Code Elimination Using Extended Value Graph*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 179-193, 1999.
-  Alfred Tarski. *A Lattice-theoretical Fixpoint Theorem and its Applications*. Pacific Journal of Mathematics 5:285-309, 1955.
-  Ashish Tiwari, Sumit Gulwani. *Static Program Analysis Using Theorem Proving*. In Proceedings of the 21st Conference on Automated Deduction (CADE-21), LNCS 4603, Springer-V., 147-166, 2007.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14




Kap. 15

Kap. 16

Kap. 17

Lösung

## II Artikel, Dissertationen, Sammelbände (36)

-  Peng Tu, David A. Padua. *Gated SSA-based Demand-driven Symbolic Analysis for Parallelizing Computers*. In Proceedings of the International Conference on Supercomputing (SC'95), 414-423, 1995.
-  Mark N. Wegman, F. Ken Zadeck. *Constant Propagation With Conditional Branches*. ACM Transactions on Programming Languages and Systems 13(2):181-201, 1991.
-  Daniel Weise. *Static Analysis of Mega-Programs (Invited Paper)*. In Proceedings of the 6th Static Analysis Symposium (SAS'99), Springer-V., LNCS 1694, 300-302, 1999.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14



Kap. 15

Kap. 16

Kap. 17

Seite 78/82

## II Artikel, Dissertationen, Sammelbände (37)

-  Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter Puschner, Jan Staschulat, Per Stenström. *The Worst-case Execution Time Problem – Overview of Methods and Survey of Tools*. ACM Transactions on Embedded Computing Systems 7(3):36.1-36.53, 2008.
-  X. Yuan, Rajiv Gupta, R. Melham. *Demand-driven Data Flow Analysis for Communication Optimization*. Parallel Processing Letters 7(4):359-370, 1997.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

Lösung

# Anhang

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# A

## Mathematische Grundlagen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Mathematische Grundlagen

## ...im Zusammenhang mit der Festlegung

- ▶ der denotationellen Semantik von *WHILE*,
- ▶ der abstrakten Semantiken für Programmanalysen,
- ▶ formaler Optimalitätskriterien von Programmtransformationen.

## Insbesondere

- ▶ Mengen, Relationen, Verbände
- ▶ Partielle und vollständige partielle Ordnungen
- ▶ Fixpunkte und Fixpunkttheoreme

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# A.1

## Mengen und Relationen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Mengen und Relationen

## Definition (A.1.1)

Sei  $M$  eine Menge,  $R$  eine Relation auf  $M$ , d.h.  $R \subseteq M \times M$ .

Dann heißt  $R$

- ▶ **reflexiv** gdw  $\forall m \in M. m R m$
- ▶ **transitiv** gdw  $\forall m, n, p \in M. m R n \wedge n R p \Rightarrow m R p$
- ▶ **antisymmetrisch** gdw  
 $\forall m, n \in M. m R n \wedge n R m \Rightarrow m = n$
- ▶ **symmetrisch** gdw  $\forall m, n \in M. m R n \iff n R m$
- ▶ **total** (oder **linear**) gdw  $\forall m, n \in M. m R n \vee n R m$



# A.2

## Ordnungen

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Ordnungen

## Definition (A.2.1, Quasiordnung, partielle Ordnung)

Eine Relation  $R$  auf  $M$  heißt

- ▶ **Quasiordnung** gdw  $R$  ist reflexiv und transitiv
- ▶ **partielle Ordnung** gdw  $R$  ist reflexiv, transitiv und antisymmetrisch

## Definition (A.2.2, Äquivalenzrelation)

Eine Relation  $R$  auf  $M$  heißt

- ▶ **Äquivalenzrelation** gdw  $R$  ist reflexiv, transitiv und symmetrisch

# Beachte

...eine **partielle Ordnung** ist also eine **antisymmetrische Quasiordnung**, eine **Äquivalenzrelation** eine **symmetrische Quasiordnung**.

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Schranken, kleinste und größte Elemente

## Definition (A.2.3, Schranken, kleinste und größte Elemente)

Sei  $(Q, \sqsubseteq)$  eine **Quasiordnung**, sei  $q \in Q$  und  $Q' \subseteq Q$ .

Dann heißt  $q$

- ▶ **obere (untere) Schranke** von  $Q'$ , in Zeichen:  
 $Q' \sqsubseteq q$  ( $q \sqsubseteq Q'$ ), wenn für alle  $q' \in Q'$  gilt:  
 $q' \sqsubseteq q$  ( $q \sqsubseteq q'$ ).
- ▶ **kleinste obere (größte untere) Schranke** von  $Q'$ , wenn  $q$   
obere (untere) Schranke von  $Q'$  ist und für jede andere  
obere (untere) Schranke  $\hat{q}$  von  $Q'$  gilt:  $q \sqsubseteq \hat{q}$  ( $\hat{q} \sqsubseteq q$ ).
- ▶ **größtes (kleinstes) Element** von  $Q$ , wenn gilt:  
 $Q \sqsubseteq q$  ( $q \sqsubseteq Q$ ).

# Existenz und Eindeutigkeit von Schranken

...in partiellen Ordnungen.

- ▶ In partiellen Ordnungen sind kleinste obere und größte untere Schranken **eindeutig** bestimmt, **wenn sie existieren**.
- ▶ **Existenz** (und damit **Eindeutigkeit**) vorausgesetzt, wird die **kleinste obere** (**größte untere**) **Schranke** einer Menge  $P' \subseteq P$  der Grundmenge einer partiellen Ordnung  $(P, \sqsubseteq)$  mit  $\bigsqcup P'$  ( $\bigsqcap P'$ ) bezeichnet. Man spricht dann auch vom **Supremum** und **Infimum** von  $P'$ .
- ▶ Analog für **kleinste** und **größte Elemente**. Existenz vorausgesetzt, sind sie eindeutig und werden üblicherweise mit  $\perp$  und  $\top$  bezeichnet.

# A.3

## Verbände

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

# Verbände und vollständige Verbände

## Definition (A.3.1, (Vollständiger) Verband)

Sei  $(P, \sqsubseteq)$  eine **partielle Ordnung**.

Dann heißt  $(P, \sqsubseteq)$

- ▶ **Verband**, wenn jede **endliche** Teilmenge  $P'$  von  $P$  eine kleinste obere und eine größte untere Schranke in  $P$  besitzt.
- ▶ **vollständiger Verband**, wenn **jede** Teilmenge  $P'$  von  $P$  eine kleinste obere und eine größte untere Schranke in  $P$  besitzt.

...(vollständige) Verbände sind also spezielle partielle Ordnungen.

# A.4

## Vollständige partielle Ordnungen



# Vollständige partielle Ordnungen

...ein etwas schwächerer als der Verbandsbegriff, aber für viele Problemstellungen in der Informatik ausreichend.

## Definition (A.4.1, Vollständige partielle Ordnung)

Sei  $(P, \sqsubseteq)$  eine **partielle Ordnung**.

Dann heißt  $(P, \sqsubseteq)$

- ▶ **vollständig**, kurz **CPO** (engl. **complete partial order**), wenn jede aufsteigende Kette  $K \subseteq P$  eine kleinste obere Schranke in  $P$  besitzt.

# Bemerkung

Es gilt:

- ▶ Eine CPO  $(C, \sqsubseteq)$  (genauer wäre: “kettenvollständige partielle Ordnung (engl. chain complete partial order (CCPO))”) besitzt stets ein kleinstes Element, das eindeutig bestimmt ist als Supremum der leeren Kette und üblicherweise mit  $\perp$  bezeichnet wird:  $\perp =_{df} \bigsqcup \emptyset$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

## Definition (A.4.2, Kette)

Sei  $(P, \sqsubseteq)$  eine **partielle Ordnung**.

Eine Teilmenge  $K \subseteq P$  heißt

- ▶ **Kette** in  $P$ , wenn die Elemente in  $K$  total geordnet sind.

Für  $K = \{k_0 \sqsubseteq k_1 \sqsubseteq k_2 \sqsubseteq \dots\}$  ( $\{k_0 \supseteq k_1 \supseteq k_2 \supseteq \dots\}$ )  
spricht man auch genauer von einer **aufsteigenden**  
(**absteigenden**) Kette in  $P$ .

Eine **Kette**  $K$  heißt

- ▶ **endlich**, wenn  $K$  endlich ist, sonst **unendlich**.

# Kettenendlichkeit, endliche Elemente

## Definition (A.4.3, Kettenendlich)

Eine **partielle Ordnung**  $(P, \sqsubseteq)$  heißt

- ▶ **kettenendlich** gdw  $P$  enthält keine unendlichen Ketten.

## Definition (A.4.4, Endliche Elemente)

Ein **Element**  $p \in P$  heißt

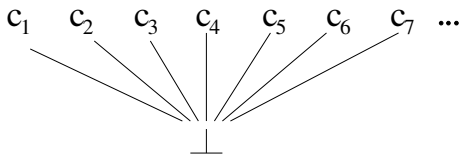
- ▶ **endlich** gdw die Menge  $Q =_{df} \{q \in P \mid q \sqsubseteq p\}$  keine unendliche Kette enthält.
- ▶ **endlich relativ zu**  $r \in P$  gdw die Menge  $Q =_{df} \{q \in P \mid r \sqsubseteq q \sqsubseteq p\}$  keine unendliche Kette enthält.

# (Standard-) CPO-Konstruktionen (1)

## Flache CPOs.

Sei  $(C, \sqsubseteq)$  eine CPO. Dann heißt  $(C, \sqsubseteq)$

- ▶ **flach**, wenn für alle  $c, d \in C$  gilt:  
 $c \sqsubseteq d \Leftrightarrow c = \perp \vee c = d$



## (Standard-) CPO-Konstruktionen (2)

### Produktkonstruktion.

Seien  $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \dots, (P_n, \sqsubseteq_n)$  CPOs.

Dann sind auch

- ▶ das **nichtstrikte (direkte) Produkt**  $(\times P_i, \sqsubseteq)$  mit
  - ▶  $(\times P_i, \sqsubseteq) = (P_1 \times P_2 \times \dots \times P_n, \sqsubseteq)$  mit  
 $\forall (p_1, p_2, \dots, p_n),$   
 $(q_1, q_2, \dots, q_n) \in \times P_i. (p_1, p_2, \dots, p_n) \sqsubseteq$   
 $(q_1, q_2, \dots, q_n) \Leftrightarrow \forall i \in \{1, \dots, n\}. p_i \sqsubseteq_i q_i$
- ▶ das **strikte (direkte) Produkt (smash Produkt)** mit
  - ▶  $(\otimes P_i, \sqsubseteq) = (P_1 \otimes P_2 \otimes \dots \otimes P_n, \sqsubseteq)$ , wobei  $\sqsubseteq$  wie oben definiert ist, jedoch zusätzlich gesetzt wird:

$$(p_1, p_2, \dots, p_n) = \perp \Leftrightarrow \exists i \in \{1, \dots, n\}. p_i = \perp_i$$

CPOs.

# (Standard-) CPO-Konstruktionen (3)

## Summenkonstruktion.

Seien  $(P_1, \sqsubseteq_1), (P_2, \sqsubseteq_2), \dots, (P_n, \sqsubseteq_n)$  CPOs.

Dann ist auch

- ▶ die **direkte Summe**  $(\bigoplus P_i, \sqsubseteq)$  mit
  - ▶  $(\bigoplus P_i, \sqsubseteq) = (P_1 \dot{\cup} P_2 \dot{\cup} \dots \dot{\cup} P_n, \sqsubseteq)$  disjunkte Vereinigung der  $P_i$ ,  $i \in \{1, \dots, n\}$  und  $\forall p, q \in \bigoplus P_i$ .  $p \sqsubseteq q \Rightarrow \exists i \in \{1, \dots, n\}$ .  $p, q \in P_i \wedge p \sqsubseteq_i q$  und der Identifikation der kleinsten Elemente der  $(P_i, \sqsubseteq_i)$ ,  $i \in \{1, \dots, n\}$ , d.h.  $\perp =_{df} \perp_i$ ,  $i \in \{1, \dots, n\}$

eine CPO.

# (Standard-) CPO-Konstruktionen (4)

## Funktionsraum.

Seien  $(C, \sqsubseteq_C)$  und  $(D, \sqsubseteq_D)$  zwei CPOs und sei  $[C \rightarrow D] =_{df} \{f : C \rightarrow D \mid f \text{ stetig}\}$  die Menge der stetigen Funktionen von  $C$  nach  $D$ .

Dann ist auch

- ▶ der **stetige Funktionsraum**  $([C \rightarrow D], \sqsubseteq)$  mit

$$\forall f, g \in [C \rightarrow D]. f \sqsubseteq g \iff \forall c \in C. f(c) \sqsubseteq_D g(c)$$

eine CPO.



# Monotonie und Stetigkeit

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

## Definition (A.4.5, Monotone, stetige Funktion)

Seien  $(C, \sqsubseteq_C)$  und  $(D, \sqsubseteq_D)$  zwei CPOs, sei  $f : C \rightarrow D$  eine Funktion von  $C$  nach  $D$ .

Dann heißt  $f$

- ▶ **monoton** gdw  $\forall c, c' \in C. c \sqsubseteq_C c' \Rightarrow f(c) \sqsubseteq_D f(c')$   
(Erhalt der Ordnung der Elemente)
- ▶ **stetig** gdw  $\forall C' \subseteq C. f(\bigsqcup_C C') =_D \bigsqcup_D f(C')$   
(Erhalt der kleinsten oberen Schranken)

# Eigenschaften

Mit den vorigen Bezeichnungen gilt:

## Lemma (A.4.6)

*f* ist monoton gdw.  $\forall C' \subseteq C. f(\sqcup_C C') \sqsupseteq_D \sqcup_D f(C')$

## Korollar (A.4.7)

*Eine stetige Funktion ist stets monoton, d.h. f stetig impliziert f monoton.*

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

## Definition (A.4.8, Inflationäre Funktion)

Sei  $(C, \sqsubseteq)$  eine CPO, sei  $f : C \rightarrow C$  eine Funktion auf  $C$ .

Dann heißt  $f$

- ▶ **inflationär (vergrößernd)** gdw  $\forall c \in C. c \sqsubseteq f(c)$

# A.5

## Fixpunkte und Fixpunkttheoreme

# Fixpunkte

## Definition (A.5.1, (Kleinster/größter) Fixpunkt)

Sei  $(C, \sqsubseteq)$  eine CPO,  $f : C \rightarrow C$  eine Funktion auf  $C$  und sei  $c$  ein Element von  $C$ , d.h.  $c \in C$ .

Dann heißt  $c$

- ▶ **Fixpunkt** von  $f$  gdw  $f(c) = c$

Ein **Fixpunkt**  $c$  von  $f$  heißt

- ▶ **kleinster Fixpunkt** von  $f$  gdw  $\forall d \in C. f(d) = d \Rightarrow c \sqsubseteq d$
- ▶ **größter Fixpunkt** von  $f$  gdw  $\forall d \in C. f(d) = d \Rightarrow d \sqsubseteq c$

Bezeichnungen:

- ▶ Der **kleinste** bzw. **größte Fixpunkt** einer Funktion  $f$  wird meist mit  $\mu f$  bzw.  $\nu f$  bezeichnet.

# Bedingte Fixpunkte

## Definition (A.5.2, Bedingter Fixpunkt)

Sei  $(C, \sqsubseteq)$  eine CPO,  $f : C \rightarrow C$  eine Funktion auf  $C$  und seien  $d, c_d \in C$ .

Dann heißt  $c_d$

- ▶ **bedingter kleinster Fixpunkt** von  $f$  bezüglich  $d$  gdw  $c_d$  ist der kleinste Fixpunkt von  $C$  mit  $d \sqsubseteq c_d$ , d.h. für alle anderen Fixpunkte  $x$  von  $f$  mit  $d \sqsubseteq x$  gilt:  $c_d \sqsubseteq x$ .

## Theorem (A.5.3, Knaster/Tarski, Kleene)

Sei  $(C, \sqsubseteq)$  eine CPO und  $f : C \rightarrow C$  eine stetige Funktion auf  $C$ .

Dann hat  $f$  einen kleinsten Fixpunkt  $\mu f$  und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette (sog. *Kleene-Kette*)  $\{\perp, f(\perp), f^2(\perp), \dots\}$ , d.h.

$$\mu f = \bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) = \bigsqcup \{\perp, f(\perp), f^2(\perp), \dots\}$$

# Beweis von Fixpunkttheorem A.5.3 (1)

Zu zeigen:

$\mu f$

1. existiert
2. ist Fixpunkt
3. ist kleinster Fixpunkt

von  $f$ .

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17



# Beweis von Fixpunkttheorem A.5.3 (2)

## 1. Existenz

- ▶ Es gilt  $f^0 \perp = \perp$  und  $\perp \sqsubseteq c$  für alle  $c \in C$ .
- ▶ Durch vollständige Induktion lässt sich damit zeigen:  
 $f^n \perp \sqsubseteq f^n c$  für alle  $c \in C$ .
- ▶ Somit gilt  $f^n \perp \sqsubseteq f^m \perp$  für alle  $n, m$  mit  $n \leq m$ . Somit ist  $\{f^n \perp \mid n \geq 0\}$  eine (nichtleere) Kette in  $C$ .
- ▶ Damit folgt die Existenz von  $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)$  aus der CPO-Eigenschaft von  $(C, \sqsubseteq)$ .

# Beweis von Fixpunkttheorem A.5.3 (3)

## 2. Fixpunkteigenschaft

$$\begin{aligned} & f(\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)) \\ (f \text{ stetig}) \quad &= \bigsqcup_{i \in \mathbb{N}_0} f(f^i \perp) \\ &= \bigsqcup_{i \in \mathbb{N}_1} f^i \perp \\ (K \text{ Kette} \Rightarrow \bigsqcup K = \perp \sqcup \bigsqcup K) \quad &= \bigsqcup_{i \in \mathbb{N}_1} f^i \perp \sqcup \perp \\ (f^0 = \perp) \quad &= \bigsqcup_{i \in \mathbb{N}_0} f^i \perp \\ &= \bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) \end{aligned}$$

# Beweis von Fixpunkttheorem A.5.3 (4)

## 3. Kleinster Fixpunkt

- ▶ Sei  $c$  beliebig gewählter Fixpunkt von  $f$ . Dann gilt  $\perp \sqsubseteq c$  und somit auch  $f^n \perp \sqsubseteq f^n c$  für alle  $n \geq 0$ .
- ▶ Folglich gilt  $f^n \perp \sqsubseteq c$  wg. der Wahl von  $c$  als Fixpunkt von  $f$ .
- ▶ Somit gilt auch, dass  $c$  eine obere Schranke von  $\{f^i(\perp) \mid i \in \mathbb{N}_0\}$  ist.
- ▶ Da  $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp)$  nach Definition die kleinste obere Schranke dieser Kette ist, gilt wie gewünscht  $\bigsqcup_{i \in \mathbb{N}_0} f^i(\perp) \sqsubseteq c$ .



# Bedingte Fixpunkte

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

## Theorem (A.5.4, Bedingte Fixpunkte)

Sei  $(C, \sqsubseteq)$  eine CPO und  $f : C \rightarrow C$  eine stetige, inflationäre Funktion auf  $C$  und sei  $d \in C$ .

Dann hat  $f$  einen kleinsten bedingten Fixpunkt  $\mu f_d$  bezüglich  $d$  und dieser Fixpunkt ergibt sich als kleinste obere Schranke der Kette  $\{d, f(d), f^2(d), \dots\}$ , d.h.

$$\mu f_d = \bigsqcup_{i \in \mathbb{N}_0} f^i(d) = \bigsqcup \{d, f(d), f^2(d), \dots\}$$

# Endliche Fixpunkte

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17

## Theorem (A.5.5, Endliche Fixpunkte)

Sei  $(C, \sqsubseteq)$  eine CPO und  $f : C \rightarrow C$  eine stetige Funktion auf  $C$ .


Dann gilt: Sind in der Kleene-Kette von  $f$  zwei aufeinanderfolgende Glieder gleich, etwa  $f^i(\perp) = f^{i+1}(\perp)$ , so gilt  $\mu f = f^i(\perp)$ .

# Existenz endlicher Fixpunkte

Hinreichende Bedingungen für die Existenz endlicher Fixpunkte sind z.B.:

- ▶ Endlichkeit von Definitionsbereich und Wertebereich von  $f$ .
- ▶  $f$  ist von der Form  $f(c) = c \sqcup g(c)$  für monotonen  $g$  über kettenendlichem Wertebereich.

# Leseempfehlungen für Anhang A

-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: A Formal Introduction*. Wiley, 1992. (Chapter 4, Denotational Semantics)
-  Hanne Riis Nielson, Flemming Nielson. *Semantics with Applications: An Appetizer*. Springer-V., 2007. (Chapter 5, Denotational Semantics)

Inhalt

Kap. 1

Kap. 2

Kap. 3

Kap. 4

Kap. 5

Kap. 6

Kap. 7

Kap. 8

Kap. 9

Kap. 10

Kap. 11

Kap. 12

Kap. 13

Kap. 14

Kap. 15

Kap. 16

Kap. 17