

Verlässliche Echtzeitsysteme

Abstrakte Interpretation

Peter Ulbrich

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
www4.informatik.uni-erlangen.de

30. Juni 2014



Fragestellungen

- Warum ist es so schwierig **Korrektheitsaussagen** zu formulieren?
 - auch wenn nur eine **bestimmten Programmeigenschaft** relevant ist

↪ Wie hilft uns „**Abstrakte Interpretation**“ bei diesem Problem?
- Was sind die **mathematischen Grundlagen** abstrakter Interpretation?
 - eine „informelle“ Sichtweise auf die Zusammenhänge
- **Ziel:** **grobes Verständnis** abstrakter Interpretation entwickeln!



Gliederung

- 1 Überblick
- 2 Problemstellung
- 3 Sammelsemantiken
- 4 Präfixsemantiken
- 5 Mathematische Grundlagen
- 6 Zusammenfassung



Wiederholung: Was kann hier alles schief gehen?

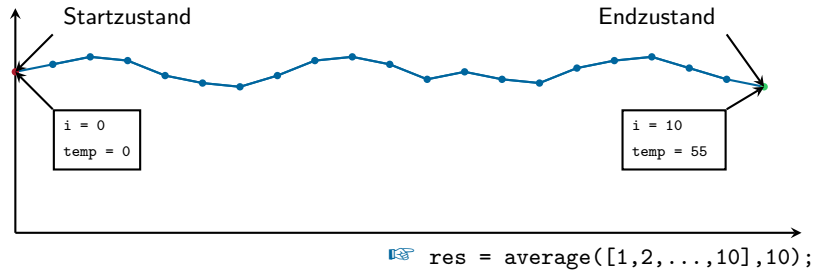
Die Gretchenfrage der Softwareentwicklung ...

```
1 unsigned int average(unsigned int *array,  
2 unsigned int size)  
3 {  
4     unsigned int temp = 0;  
5  
6     for(unsigned int i = 0; i < size; i++) {  
7         temp += array[i];  
8     }  
9  
10    return temp/size;  
11 }
```

- Wo könnte es hier klemmen?
 - Ist der Zugriff auf Feld array in Zeile 7 korrekt?
 - Kann die Addition in Zeile 7 überlaufen?
 - Kann in Zeile 10 eine Division durch 0 auftreten?
- Wie findet man das heraus?
 - ↪ Schauen wir mal, wie sich das Programm verhält.



Das Verhalten zur Laufzeit ist entscheidend!



```

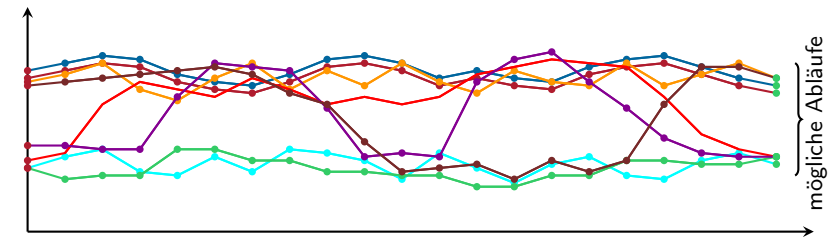
1 unsigned int average(uint *array,
2                     uint size)
3 {
4     uint temp = 0;
5     for(uint i = 0; i < size; i++) {
6         temp += array[i];
7     }
8     return temp/size;
9 }
10
11

```

i	temp
0	0
1	1
2	3
3	6
...	...
10	55

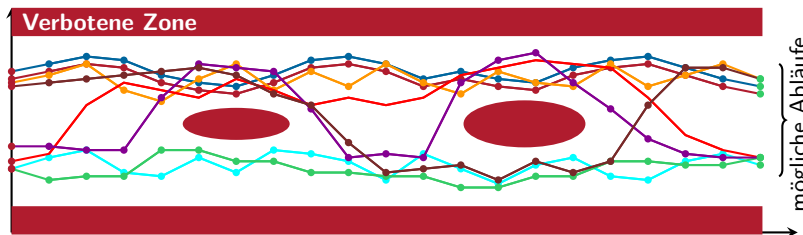
Konkrete Programmsemantik

Eine informelle Einführung in die Prinzipien abstrakter Interpretation [1]



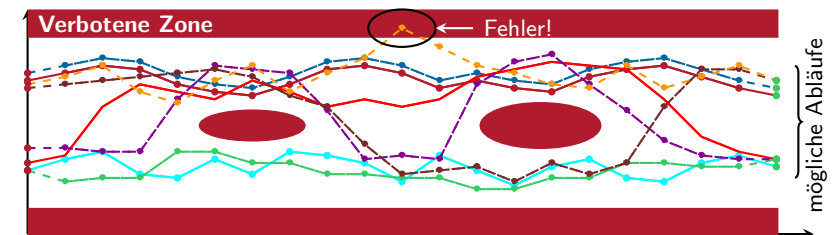
- die **konkrete Semantik** (engl. *concrete semantics*) beschreibt
 - alle möglichen Ausführungen eines Programms
 - unter allen möglichen Ausführungsbedingungen
 - Für unser Beispiel bedeutet dies:
 - 2^{32} verschieden große Felder, 2^{32} verschiedene Werte für jedes Element
- sie beschreibt ein „unendliches“ mathematisches Objekt
 - im Allgemeinen **nicht berechenbar** durch einen Algorithmus
 - alle nicht-trivialen Fragestellungen sind **nicht entscheidbar**

Sicherheitseigenschaft



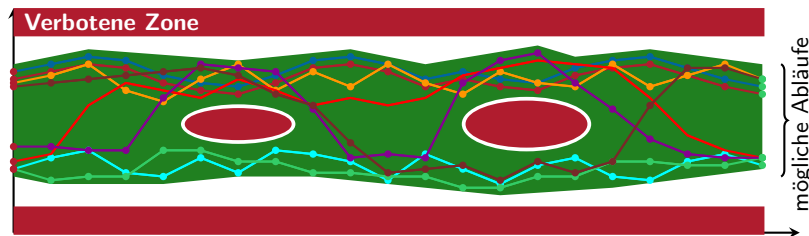
- Sicherheitseigenschaften** (engl. *safety properties*) stellen sicher, dass keine **fehlerhaften Zustände** eingenommen werden
- ein **Sicherheitsnachweis** (engl. *safety proof*) garantiert, dass die konkrete Semantik nie eine **verbotene Zone** durchläuft
- das ist ein **unentscheidbares Problem**
 - die konkrete Programmsemantik ist nicht berechenbar

Testen: Das Problem der Möglichkeiten



- Testen betrachtet **nur eine Teilmenge** aller möglichen Ausführungen
 - gut geeignet, um die **Existenz** von Defekten zu zeigen
 - ungeeignet, um ihre **Abwesenheit** zu zeigen
 - evtl. hat man die fehlerhafte Ausführung einfach nicht getestet
- Problem: **unzureichende Abdeckung** der konkreten Semantik

Abstrakte Interpretation



- **Abstrakte Interpretation** (engl. *abstract interpretation*)
 - betrachtet eine **abstrakte Semantik** (engl. *abstract semantics*)
 - sie umfasst **alle Fälle der konkreten Programmsemantik**
 - ist die abstrakte Semantik sicher \Rightarrow konkrete Semantik ist sicher



Formale Methoden sind abstrakte Interpretationen

Die abstrakte Semantik wird aber auf unterschiedliche Weise bestimmt

Model Checking

- abstrakte Semantik wird explizit vom Nutzer angegeben
 - \leadsto endliche Beschreibung der konkreten Programmsemantik
 - z.B. endliche Automaten, Aussagen- oder Prädikatenlogik
- automatische Ableitung durch **statische Analyse**

Deduktive Methoden

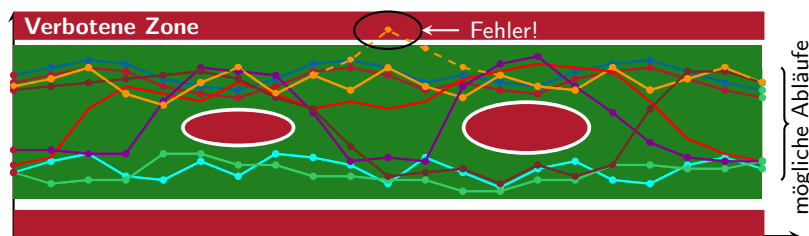
- abstrakte Semantik wird durch Nachbedingungen beschrieben
- Nutzer gibt sie durch induktive Argumente an
 - z.B. Vorbedingungen und Invarianten
- automatische Ableitung durch **statische Analyse**

Statische Analyse

- abstrakte Semantik wird ausgehend vom Quelltext bestimmt
 - Abbildung auf **vorab bestimmte, wohldefinierte Abstraktionen**
- Anpassungen (automatisch/durch den Nutzer) sind möglich



Eigenschaften abstrakter Semantiken

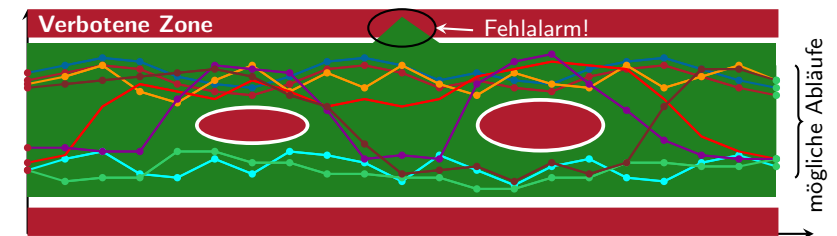


Vollständigkeit und Korrektheit

- keine potentieller Defekt darf übersehen werden
 - \leadsto nur so kann die Abwesenheit von Defekten gezeigt werden
 - ansonsten wäre gegenüber reinem Testen nichts gewonnen



Eigenschaften abstrakter Semantiken

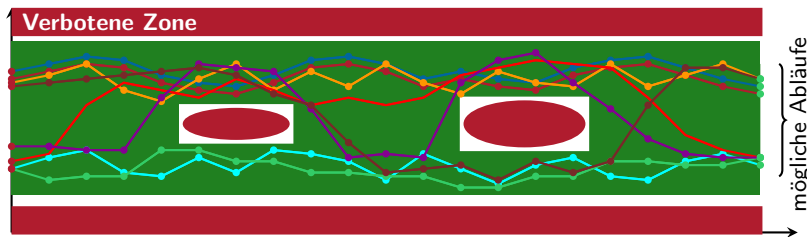


Präzision

- weitgehende Vermeidung von **Fehlalarmen** (engl. *false alarms*)
 - synonyme englische Bezeichnung: **false positives**
- ermöglicht erst eine vollkommen automatisierte Anwendung



Eigenschaften abstrakter Semantiken



geringe Komplexität

- Berechnung der abstrakten Semantik in akzeptabler Laufzeit
 - Vermeidung der kombinatorischen Explosion des Zustandsraums



Problemstellung

Reduktion des Zustandsraums ist unumgänglich!

- ☞ Fasse verschiedene Zustände geeignet zusammen
 - ↪ Sammelsemantiken (s. Folie X/14 ff.)
- ☞ Betrachte nur den Anfang der Zustandshistorie
 - ↪ Präfixsemantiken (s. Folie X/22 ff.)

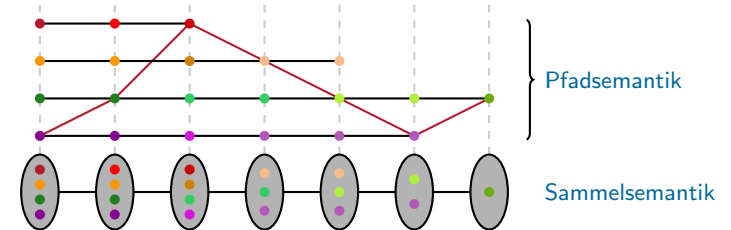


Gliederung

- 1 Überblick
- 2 Problemstellung
- 3 Sammelsemantiken
- 4 Präfixsemantiken
- 5 Mathematische Grundlagen
- 6 Zusammenfassung



Sammelsemantik (engl. *collecting semantics*)

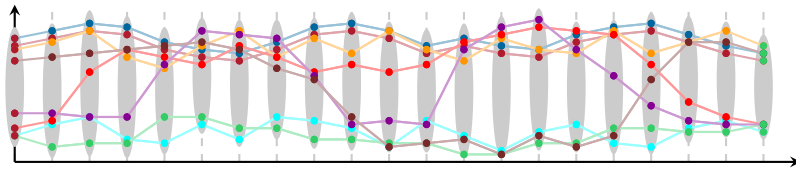


- sammelt die Zustände aller Pfade an einem bestimmten Punkt
 - d. h. an einer bestimmten Programmanweisung
 - aufgrund der Größe, wird sie i. d. R. approximiert
- das ist eine **verlustbehaftete Abstraktion**
 - Beispiel: Existiert der rote Pfad?
 - konkrete Semantik \mapsto Nein, Sammelsemantik \mapsto ???
- ☞ Der **Laufzeitgewinn** wird durch **Unschärfe** erkaufte!
 - das Ergebnis „Weiß nicht ...“ ist typisch für solche Methoden
 - und die Ursache vieler Vorbehalte ...

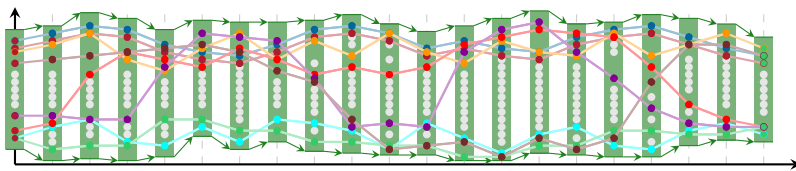


Intervallabstraktion

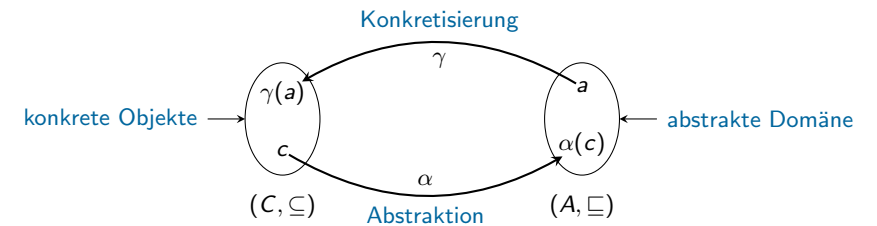
Als Approximation der Sammelsemantik



- die Sammelsemantik verwaltet Zustandsmengen
 - ☞ die Intervallabstraktion nur ihre oberen und unteren Schranken
 - die zu verwaltenden Daten werden dadurch beträchtliche reduziert
 - allerdings wird auch die Präzision reduziert
- ~ bestimmte Zustände im approximierten Zustandsraum werden nicht erreicht



Theoretisches Fundament ~ Galoisverbindungen



- wähle eine **abstrakte Domäne** (engl. *abstract domain*)
 - ersetzt die Menge konkreter Objekte S durch ihre Abstraktion $\alpha(S)$
 - verschiedene Domänen unterscheiden sich hinsichtlich ihrer Präzision
 - Vorzeichen, Intervalle, Oktagon, Polyhedra, ...
- **Abstraktionsfunktion** α (engl. *abstraction function*)
 - bildet die Menge konkrete Objekte auf ihre abstrakte Interpretation ab
- **Konkretisierungsfunktion** γ (engl. *concretization function*)
 - bildet die Menge abstrakter Objekte auf konkrete Objekte ab



Theoretisches Fundament ~ Abstrakte Interpretation

☞ Approximation von f durch die abstrakte Funktion f'

- häufig verwendet man **Galoiseinbettungen**
 - diese sind Galoisverbindungen $(C, \subseteq) \xleftrightarrow[\alpha]{\gamma} (A, \sqsubseteq)$
 - mit der Eigenschaft $\alpha(\gamma(a)) = a$
 - Konkretisierung gefolgt von Abstraktion impliziert keinen Präzisionsverlust
- **Abstrakte Interpretation** nutzt diese Eigenschaften
 - statt die konkrete Funktion $f(c)$ zu berechnen
 - kann man sie annähern, indem
 - man die abstrakte Funktion f' auf die Abstraktion $\alpha(c)$ anwendet
 - und das Ergebnis $f'(\alpha(c))$ wieder konkretisiert



Beispiel: Intervallabstraktion für ein C-Programm

```

1 unsigned short x = 1;
2
3 while(x < 10000) {
4     x = x + 1;
5 }
6
7 return x;
    
```

Die Intervallabstraktion liefert:

Zeile 1 $x_1 = [1, 1]$
 Zeile 3 $x_3 = (x_1 \cup x_4) \cap [-\infty, 9999]$
 Zeile 4 $x_4 = x_3 \oplus [1, 1]$
 Zeile 7 $x_7 = (x_1 \cup x_5) \cap [10000, \infty]$

- die Intervallabstraktion ist eine **manuell vorgegebene, abstrakte Interpretation** der Semantik der Programmiersprache C
 - C-Programme werden dann **automatisiert darauf abgebildet**
 - z. B. durch einen Übersetzer oder ein statisches Analysewerkzeug
 - nur Elemente, die den Wertebereich von x betreffen, sind relevant
 - dies ist bereits eine **starke Vereinfachung**
 - angenommen x wäre eingangs nicht bekannt
- ~ es gäbe 10000 verschiedene Pfade durch den Zustandsraum
- nehme eine Schleifenobergrenze `unsigned short y` statt 10000 an
- ~ es gäbe $\leq (2^{16})^2$ verschiedene Pfade durch den Zustandsraum



Beispiel: Intervallabstraktion für ein C-Programm (Forts.)

```
1 unsigned short x = 1;
2
3 while(x < 10000) {
4     x = x + 1;
5 }
6
7 return x;
```

Die Intervallabstraktion liefert:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = (x_1 \cup x_4) \cap [-\infty, 9999]$

Zeile 4 $x_4 = x_3 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_4) \cap [10000, \infty]$

- Approximation durch **chaotische Iteration** (engl. *chaotic iteration*)

Iteration 1:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_4 = [1, 1]$

Zeile 4 $x_4 = [2, 2]$

Zeile 7 $x_7 = \emptyset$

Iteration 2:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_4 = [1, 2]$

Zeile 4 $x_4 = [2, 3]$

Zeile 7 $x_7 = \emptyset$



Beispiel: Intervallabstraktion (Forts.)

```
1 unsigned short x = 1;
2
3 while(x < 10000) {
4     x = x + 1;
5 }
6
7 return x;
```

Die Intervallabstraktion liefert:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = (x_1 \cup x_4) \cap [-\infty, 9999]$

Zeile 4 $x_4 = x_3 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_4) \cap [10000, \infty]$

- Approximation durch **chaotische Iteration** (engl. *chaotic iteration*)

Iteration 3:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = [1, 3]$

Zeile 4 $x_4 = [2, 4]$

Zeile 7 $x_7 = \emptyset$

viele, viele Iterationen später:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = [1, 9999]$

Zeile 4 $x_4 = [2, 10000]$

Zeile 7 $x_7 = [10000, 10000]$

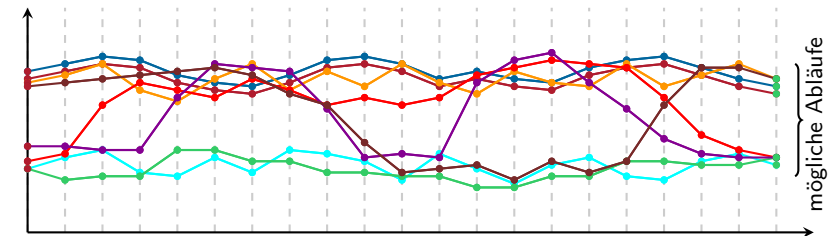


Gliederung

- 1 Überblick
- 2 Problemstellung
- 3 Sammelsemantiken
- 4 Präfixsemantiken
- 5 Mathematische Grundlagen
- 6 Zusammenfassung



Pfadsemantik



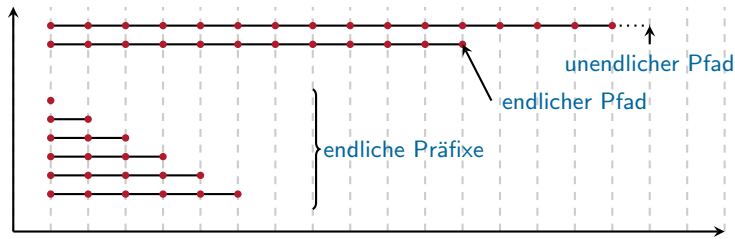
- betrachte durch ein **Transitionssystem** beschriebene **Programmpfade**
 - Ausgehend von ausgezeichneten Startzuständen,
 - beschreiben sie eine (unendliche) Abfolge von **Programmzuständen**,
 - deren Reihenfolge durch die Übergangsrelation bestimmt wird.

↪ die Gesamtheit dieser Programmpfade heißt **Pfadsemantik**

 - Wie die konkrete Programmsemantik ist sie **nicht berechenbar**.
- Reduktion der Komplexität durch **Abstraktion**
 - unendliche Pfade ↪ (endliche) **Pfadpräfixe**



Pfadpräfixe



- Pfadsemantiken enthalten alle endlichen und unendlichen Pfade

- Pfadpräfixe enthalten nur die Anfänge dieser Pfade



das ist eine **verlustbehaftete Abstraktion**

- Beispiel: betrachte Worte der Sprache $a^n b$
 - Frage: Gibt es Worte mit unendlich vielen aufeinanderfolgenden 'a'?
 - Pfadsemantik: $\{a^n b | n \geq 0\} \mapsto$ **Nein**
 - Pfadpräfixe: $\{a^n | n \geq 0\} \cup \{a^n b | n \geq 0\} \mapsto$???



Pfadpräfixe und Fixpunkte

- Menge der Präfixe ist rekursiv:

$$\text{Präfixe} = \{x | x \text{ ist Startzustand}\} \cup \{x_1 \rightarrow^* x_2 \rightarrow x_3 | x_1 \rightarrow^* x_2 \in \text{Präfixe} \wedge x_2 \rightarrow x_3 \in \rightarrow\}$$

- zu lösen ist die Fixpunktiteration $\text{Präfixe} = F(\text{Präfixe})$

- üblicherweise besitzt diese Gleichung mehrere Lösungen

↪ ordne die Lösungen nach der **Teilmengenbeziehung** \subseteq

↪ wähle die kleinste Teilmenge als Lösung

↪ **least fixpoint prefix trace semantics**

- Vereinfachungen ermöglichen **effektive, iterative Analysealgorithmen**

- Vereinfachung im Sinne von Abstraktion bzw. Approximation

↪ man muss nur noch die Präfixe betrachten

- nicht mehr die vollständigen (evtl. unendlichen) Pfade



Gliederung

- 1 Überblick
- 2 Problemstellung
- 3 Sammelsemantiken
- 4 Präfixsemantiken
- 5 **Mathematische Grundlagen**
- 6 Zusammenfassung



Warum funktioniert das eigentlich ... ?

- Wann ist eine Abstraktion **korrekt**?

↪ Wenn sie durch eine **Galoisverbindung** beschrieben wird!

↪ **OK!**

- Fixpunkte ... wer sagt, dass die Iteration überhaupt **konvergiert**?

↪ **Aufsteigende Kettenbedingung!**

↪ **???**

- Das waren ziemlich viele Iterationen ... geht das auch **schneller**?

↪ **Widening-/Narrowing-Operatoren** helfen!

↪ **???**

- **Jetzt:** Grundlegende mathematische Zusammenhänge erfassen!

- Was ist das und was hat es mit abstrakter Interpretation zu tun?

- **Nicht:** Warum ist das korrekt?

- keine Beweisführung ...



Partiell geordnete Mengen

- Konkrete und abstrakte Domänen sind **partiell geordnete Mengen!**

Partiell geordnete Mengen (engl. *partially ordered sets*)

Eine **partiell geordnete Menge** ist ein Tupel (S, \sqsubseteq) :

- S ist eine Menge,
- $\sqsubseteq \subseteq S \times S$ ist eine **Ordnungsrelation** mit folgenden Eigenschaften:

reflexiv $\forall x \in S : x \sqsubseteq x$

antisymmetrisch $\forall x, y \in S : x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y$

transitiv $\forall x, y, z \in S : x \sqsubseteq y \wedge y \sqsubseteq z \Rightarrow x \sqsubseteq z$

- Beispiele:

- (\mathbb{N}, \leq) ist ein partiell geordnete Menge
- $(\mathcal{P}(S), \subseteq)$ ist ein partiell geordnete Menge



Obere und untere Schranken

- Sei (S, \sqsubseteq) eine partiell geordnete Menge

Obere Schranke (engl. *upper bound*)

$x \in S$ eine **obere Schranke** von $P \subseteq S \Leftrightarrow y \in P : y \sqsubseteq x$

☞ analog: **untere Schranke** (engl. *lower bound*)

Kleinste obere Schranke (engl. *least upper bound*)

$x \in S$ ist eine **kleinste obere Schranke** von $P \subseteq S \Leftrightarrow$

- x ist eine obere Schranke von P und
- x ist kleiner als alle oberen Schranken von P :

$$\forall y \in S : (\forall z \in P : z \sqsubseteq y) \Rightarrow x \sqsubseteq y$$



☞ analog: **größte untere Schranke** (engl. *greatest lower bound*)



Verbände

Vollständiger Verband (engl. *complete lattice*)

Ein **vollständiger Verband** ist eine partiell geordnete Menge $(S, \sqsubseteq, \perp, \top, \sqcup, \sqcap)$ mit folgenden Eigenschaften:

- (S, \sqsubseteq) ist eine partiell geordnete Menge
- für jede Teilmenge $P \subseteq S$ existiert eine
 - eine kleinste obere Schranke $\sqcup P$ und
 - eine größte untere Schranke $\sqcap P$
- $\perp = \sqcap S$ heißt **Infimum** von S
- $\top = \sqcup S$ heißt **Supremum** von S

- Beispiele:

- $(\mathcal{P}(S), \subseteq, \emptyset, S, \cup, \cap)$ ist ein vollständiger Verband
- $(\mathbb{Z} \cup \{-\infty, +\infty\}, \leq, -\infty, +\infty, \max, \min)$ ist ein vollständiger Verband
 - die Menge der ganzen Zahlen erweitert um $-\infty$ und $+\infty$



Terminierung der Fixpunktiteration

- **Möglichkeit 1:** aufsteigende Kettenbedingung **ist erfüllt**

↪ aufsteigende Ketten sind endlich

↪ Fixpunktiteration terminiert

- **Möglichkeit 2:** aufsteigende Kettenbedingung **ist nicht erfüllt**

↪ Terminierung kann durch einen **Widening-Operator** erzwungen werden

Widening-Operator

Sei V ein Verband, ein **Widening-Operator** $\nabla : V \times V \mapsto V$ ist eine Abbildung für die gilt:

$$\forall x, y \in V : x \sqsubseteq x \nabla y \wedge y \sqsubseteq x \nabla y$$

- sicher Abschätzung der Elemente x und y nach oben durch $x \nabla y$
- ermöglicht auch eine Beschleunigung der Fixpunktiteration
 - Widening-Operator $\nabla \approx$ Bestimmung der kleinsten oberen Schranke
 - in vollständigen Verbänden mit aufsteigender Kettenbedingung



Beispiel: Intervallabstraktion – nun mit Widening

```
1 unsigned short x = 1;
2
3 while(x < 10000) {
4   x = x + 1;
5 }
6
7 return x;
```

Die Intervallabstraktion liefert:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = (x_1 \nabla x_4) \cap [-\infty, 9999]$

Zeile 4 $x_4 = x_3 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \nabla x_4) \cap [10000, \infty]$

- Approximation mit Hilfe des **Widening-Operators**

Iteration 1:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = [1, 1]$

Zeile 4 $x_4 = [2, 2]$

Zeile 7 $x_7 = \emptyset$


Iteration 2:

Zeile 1 $x_1 = [1, 1]$

Zeile 3 $x_3 = [1, 9999]$

Zeile 4 $x_4 = [2, 10000]$

Zeile 7 $x_7 = [10000, 10000]$

-  Konvergenz in der 2. Iteration



Gliederung

- 1 Überblick
- 2 Problemstellung
- 3 Sammelsemantiken
- 4 Präfixsemantiken
- 5 Mathematische Grundlagen
- 6 Zusammenfassung



Zusammenfassung

Konkrete Programmsemantik ist **nicht berechenbar**

- Approximation durch eine **abstrakte Semantik**
 - Korrektheit der Approximation ist entscheidend
 - nur so kann man einen **Sicherheitsnachweis** führen
 - die Approximation muss präzise sein
 - nur so kann man **Fehlalarme** vermeiden
 - die Approximation darf nicht zu komplex sein
 - nur so kann sie **effizient berechnet** werden

Transitionssystem beschreiben Programme

- **Pfadsemantiken** beschreiben die konkrete Programmsemantik
- Approximation durch **Pfadpräfixe** und **Sammelsemantik**
 - ↪ abstrakte Interpretation approximiert die Sammelsemantik

Mathematische Grundlagen abstrakter Interpretation

- (vollständig) partiell geordnete Mengen, Verbände
- Galoiseinbettungen, lokale konsistente Funktionen, Widening
- Intervallabstraktion



Literaturverzeichnis

- [1] Cousot, P. :
Abstract Interpretation.
<http://web.mit.edu/16.399/www/>, 2005

