Verlässliche Echtzeitsysteme

Abstrakte Interpretation

Fabian Scheler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

`www4.informatik.uni-erlangen.de`

25. April 2012

Gliederung

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

Sag mir, wie hältst du es mit den Defekten?

Die Gretchenfrage der Softwareentwicklung . . .

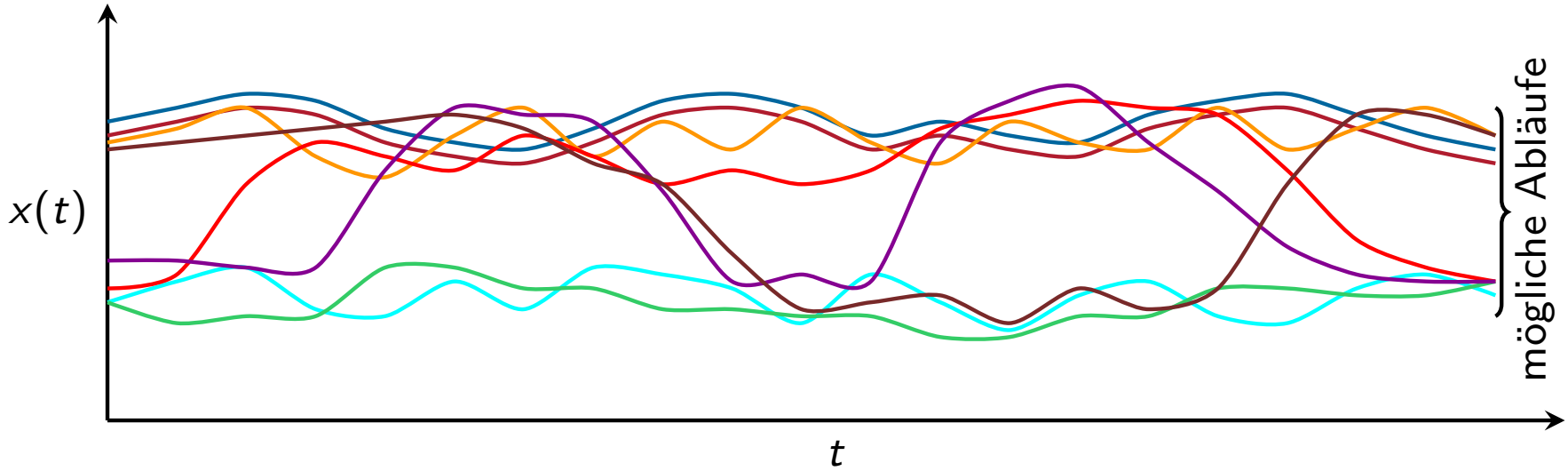
```
1  unsigned int average(unsigned int *array,
2                       unsigned int size)
3  {
4    unsigned int temp = 0;
5
6    for(unsigned int i = 0;i < size;i++) {
7      temp += array[i];
8    }
9
10   return temp/size;
11 }
```

- 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?
- Warum ist das so schwer zu beantworten?
  - Es gibt alleine 2^{32} verschiedene Größen für das Feld `array`
  - In jedem Element des Felds können 2^{32} verschiedene Werte stehen

☞ viel zu viel für eine konkrete Betrachtung

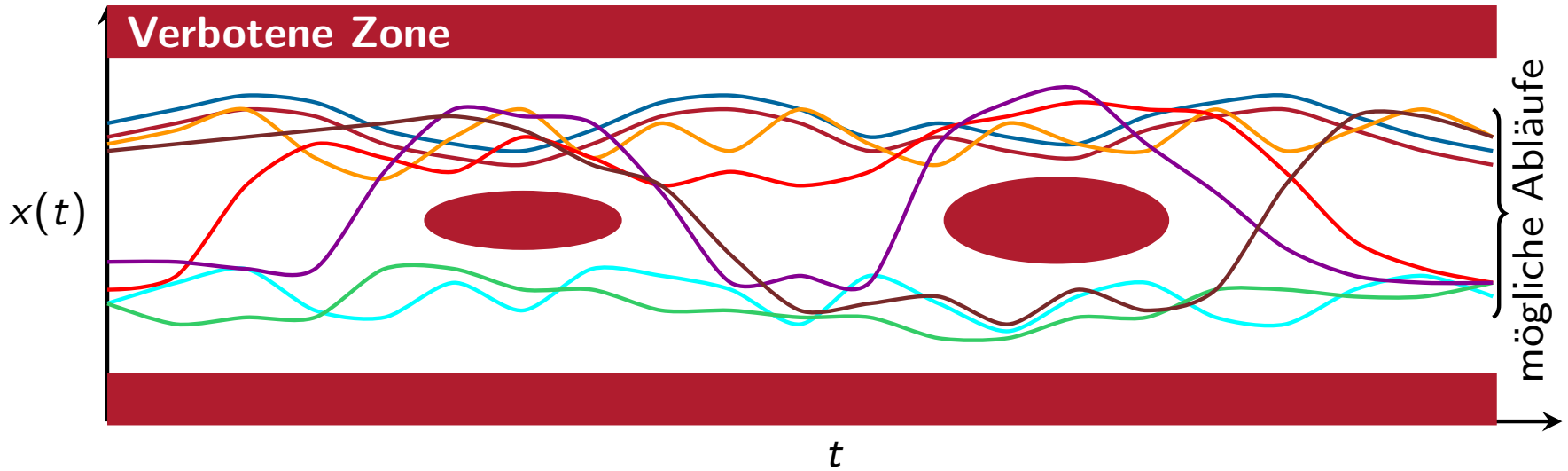
Konkrete Programmsemantik

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
- sie beschreibt ein „unendliches“ mathematisches Objekt
  - im Allgemeinen nicht berechenbar durch einen Algorithmus
  - alle nicht-trivialen Fragestellungen sind nicht entscheidbar

Sicherheitseigenschaft

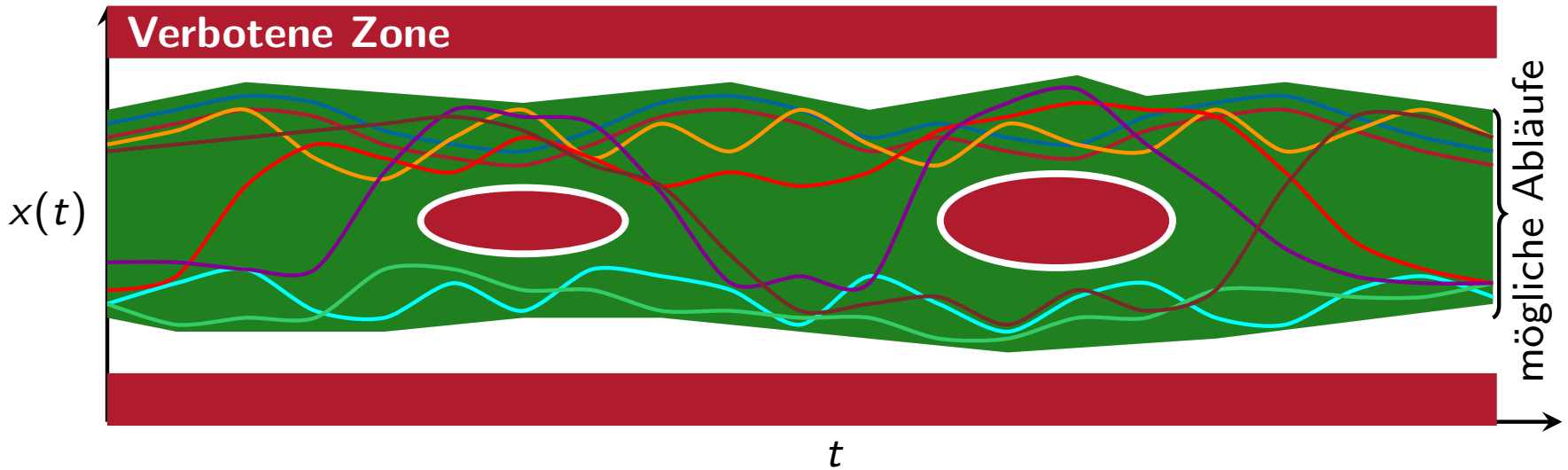


- Sicherheitseigenschaften (engl. *safety properties*) stellen sicher, dass keine fehlerhaften Zustände eingenommen werden
- ein Sicherheitsnachweis (engl. *safety proof*) garantiert, dass die konkrete Semantik nie eine verbotene Zone durchläuft

☞ das ist ein unentscheidbares Problem

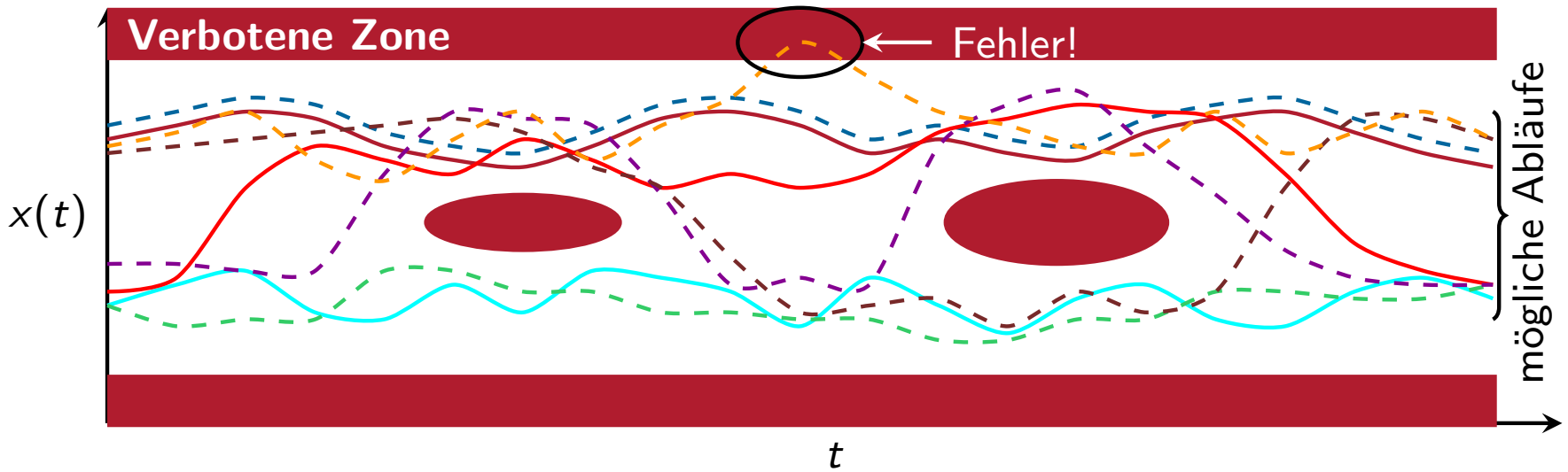
  - die konkrete Programmsemantik ist nicht berechenbar

Abstrakte Interpretation



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

Vergleich: Testen



- 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

Formale Methoden sind abstrakte Interpretationen

Die abstrakte Semantik wird aber auf unterschiedliche Weise bestimmt

Model Checking

- abstrakte Semantik wird explizit vom Nutzer angegeben

$\rightsquigarrow$ 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

$\rightsquigarrow$ nur so kann die Abwesenheit von Defekten gezeigt werden

  - ansonsten wäre gegenüber reinem Testen nichts gewonnen

Präzision

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

geringe Komplexität

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

Gliederung

Transitionssysteme

Definition: Transitionssystem

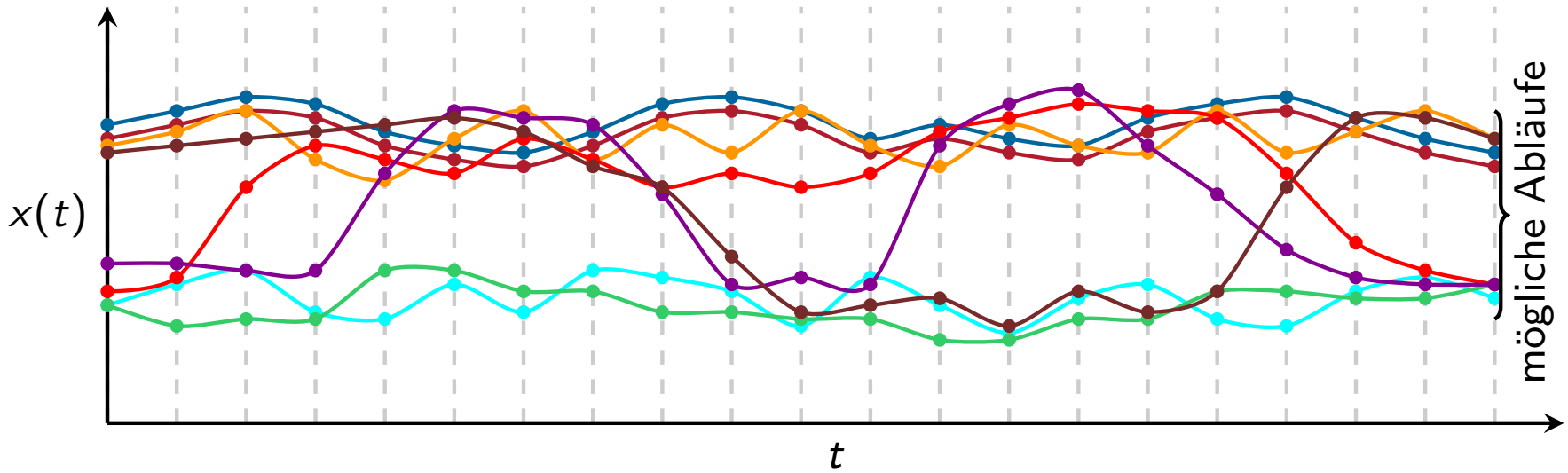
Ein Transitionssystem ist ein Quadrupel $(S, I, F, \rightarrow)$

- S ist eine Menge von Zuständen
- $I \subseteq S$ ist eine nicht-leere Menge von Startzuständen
- $F \subseteq S$ ist eine optionale Menge von Endzuständen
- $\rightarrow \subseteq S \times S$ ist die Übergangsrelation

- Beispiel: euklidischer Algorithmus [2, Woche 1]
  - $S = \mathbb{N} \times \mathbb{N}$
  - $I = \{(x, y) | x, y \in \mathbb{N}\}$
  - $F = \{(n, n) | n \in \mathbb{N}\}$
  - Übergangsrelation $\rightarrow$:

$$\rightarrow: \begin{cases} (n, m) \rightarrow (n - m, m) & ; n > m \\ (n, m) \rightarrow (n, m - n) & ; m > n \end{cases}$$

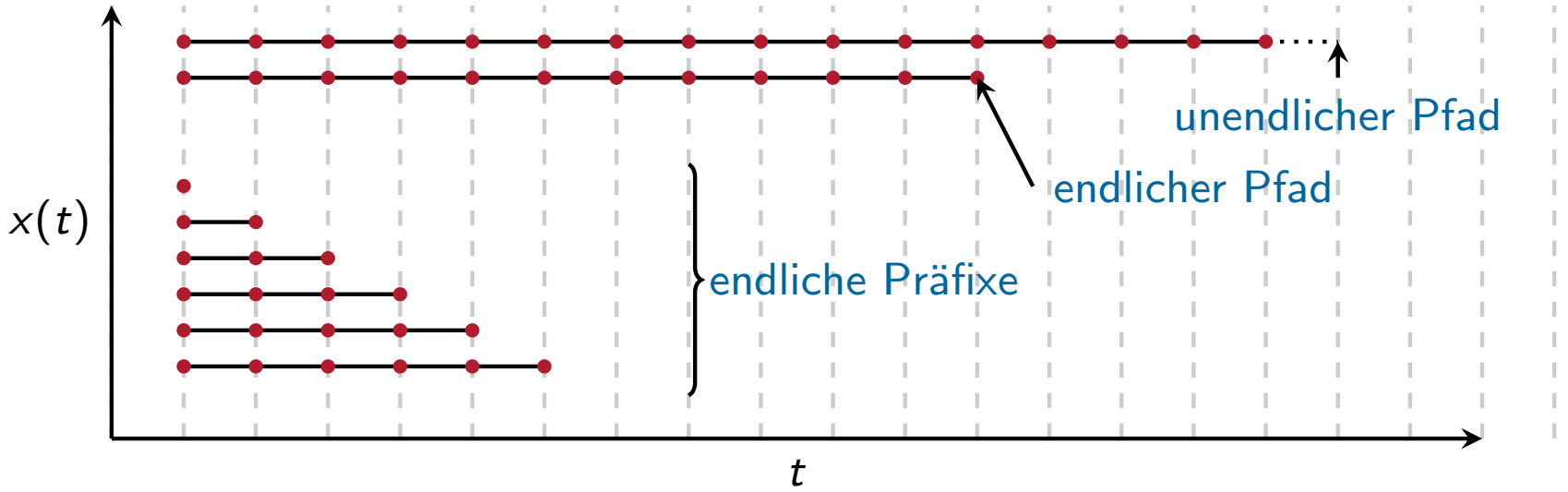
Pfadsemantik



- 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
  - Unterscheidung einzelner Zustände ⇝ Sammelsemantik

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:

$$\text{Präfixe} = \{x | x \in I\} \cup \{x_1 \to^* x_2 \to x_3 | x_1 \to^* x_2 \in \text{Präfixe} \wedge x_2 \to x_3 \in \to\}$$

- zu lösen ist die Fixpunktiteration Präfixe $= F(\text{Präfixe})$
  - üblicherweise besitzt diese Gleichung mehrere Lösungen

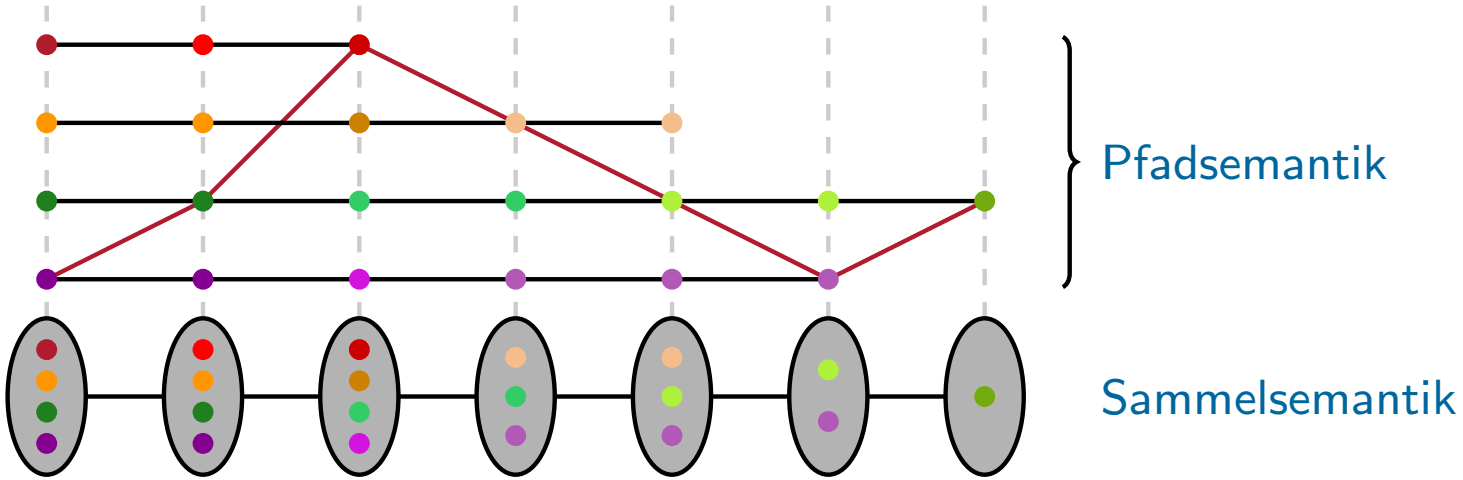
  $\rightsquigarrow$ ordne die Lösungen nach der Teilmengenbeziehung $\subseteq$

  $\rightsquigarrow$ wähle die kleinste Teilmenge als Lösung

  $\rightsquigarrow$ least fixpoint prefix trace semantics

- Vereinfachungen ermöglichen effektive, iterative Analysealgorithmen
  - Vereinfachung im Sinne von Abstraktion bzw. Approximation

Sammelsemantik (engl. *collecting semantics*)



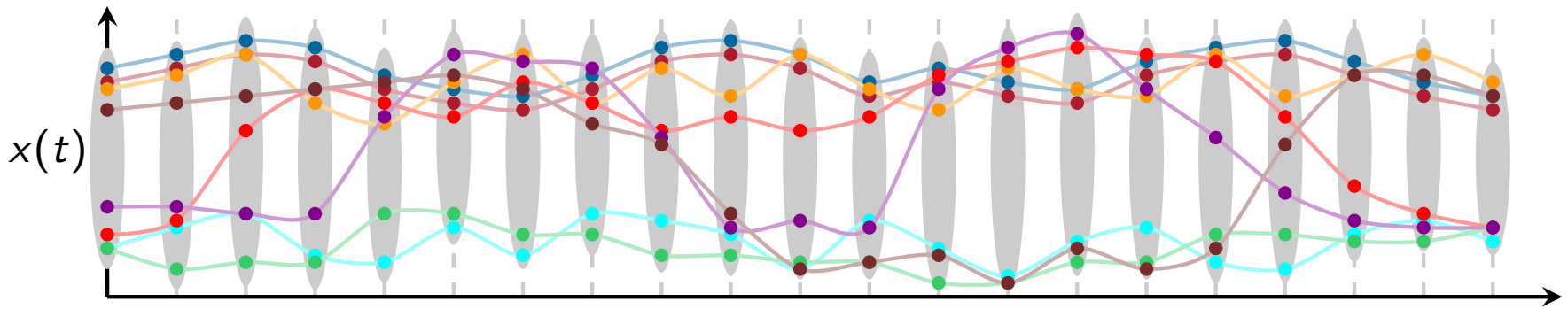
- sammelt die Zustände aller Pfade zu einem bestimmten Zeitpunkt
  - aufgrund der Größe, wird sie i. d. R. approximiert
- das ist eine verlustbehaftete Abstraktion
  - Beispiel: Existiert der rote Pfad?
    - Pfadsemantik $\mapsto$ Nein
    - Pfadpräfixe $\mapsto$???

☞ Der Laufzeitgewinn wird durch Unschärfe erkauft!

- das Ergebnis „Weiß nicht . . . “ ist typisch für solche Methoden
- und die Ursache vieler Vorbehalte . . .

Intervallabstraktion

Als Approximation der Sammelsemantik

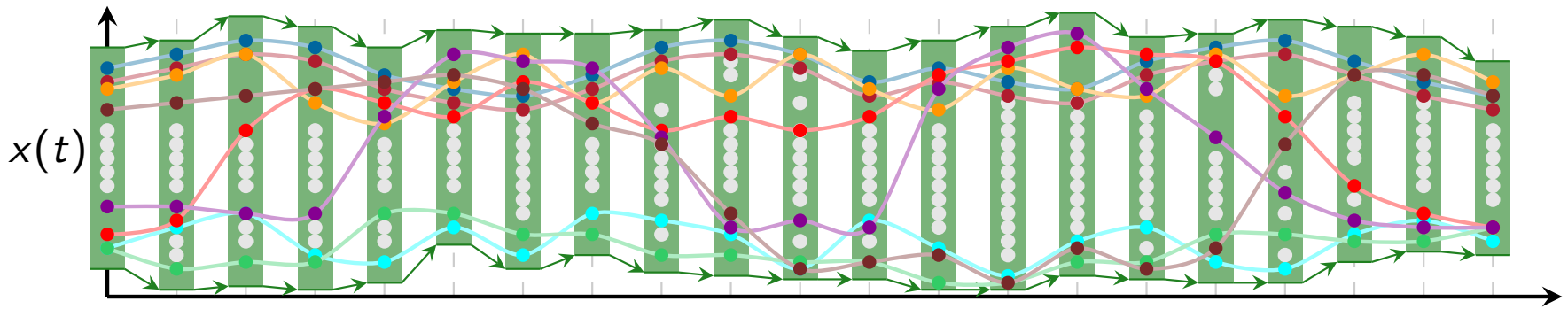


- die Sammelsemantik verwaltet Zustandsmengen

☞ die Intervallabstraktion nur ihre oberen und unteren Schranken

- die zu verwaltenden Daten werden dadurch beträchtliche reduziert
- allerdings wird auch die Präzision reduziert

  $\rightsquigarrow$ bestimmte Zustände im appromixierten Zustandsraum werden nicht erreicht



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 4 $x_4 = (x_1 \cup x_5) \cap [-\infty, 9999]$

Zeile 5 $x_5 = x_4 \oplus [1, 1]$

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

  $\rightsquigarrow$ es gäbe 10000 verschiedene Pfade durch den Zustandsraum
  - nehme eine Schleifenobergrenze `unsigned short y` statt 10000 an

  $\rightsquigarrow$ es gäbe $\leq (2^{16})^2$ verschiedene Pfade durch den Zustandsraum

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 4 $x_4 = (x_1 \cup x_5) \cap [-\infty, 9999]$

Zeile 5 $x_5 = x_4 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_5) \cap [10000, \infty]$

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

Iteration 1:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 1]$

Zeile 5 $x_5 = [2, 2]$

Zeile 7 $x_7 = \emptyset$

Iteration 2:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 2]$

Zeile 5 $x_5 = [2, 3]$

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 4 $x_4 = (x_1 \cup x_5) \cap [-\infty, 9999]$

Zeile 5 $x_5 = x_4 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_5) \cap [10000, \infty]$

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

Iteration 3:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 3]$

Zeile 5 $x_5 = [2, 4]$

Zeile 7 $x_7 = \emptyset$

viele, viele Iterationen später:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 9999]$

Zeile 5 $x_5 = [2, 10000]$

Zeile 7 $x_7 = [10000, 10000]$

Gliederung

Warum funktioniert das eigentlich . . . ?

- Wann ist eine Abstraktion korrekt?
  - ⇝ Wenn sie durch eine Galoisverbindung beschrieben wird!
- Fixpunkte . . . wer sagt, dass die Iteration überhaupt konvergiert?
  - ⇝ Aufsteigende Kettenbedingung!
- Das waren ziemlich viele Iterationen . . . geht das auch schneller?
  - ⇝ Die Verwendung von Widening- und Narrowing-Operatoren hilft!

- **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

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*)

Vollständige partielle Ordnungen

ω-Kette

Sei $(S, \sqsubseteq)$ ein partiell geordnete Menge. Eine ω-Kette einer partiell geordneten Menge ist eine aufsteigende Kette aus Elementen $x_0, x_1, x_2, \ldots \in S$ für die gilt:

$$x_0 \sqsubseteq x_1 \sqsubseteq x_2 \sqsubseteq \cdots \sqsubseteq x_i \sqsubseteq \cdots$$

- Eine partiell geordnete Menge erfüllt die Aufsteigende Kettenbedingung, wenn jede aufsteigende ω-Kette endlich ist.

Vollständige partielle Ordnung

Sei $(S, \sqsubseteq)$ ein partiell geordnete Menge. Liegt für jede ω-Kette die kleinste obere Schranke der Menge $\{x_i | i \in \omega\}$ in S, so heißt $(S, \sqsubseteq)$ vollständige partielle Ordnung.

Verbände

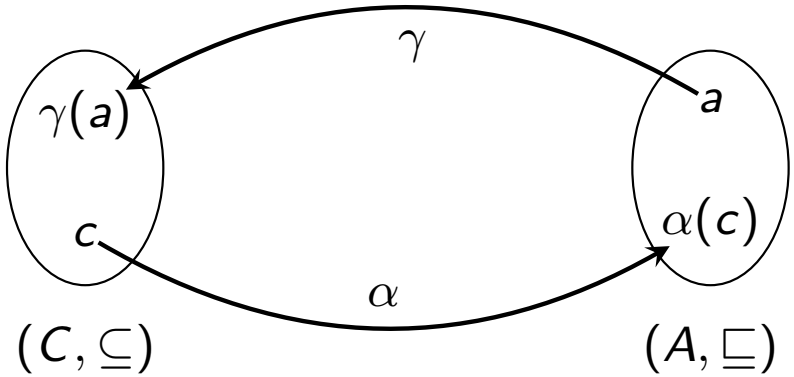
Vollständiger Verband (engl. *complete lattice*)

Ein vollständiger Verband ist eine partiell geordnete Menge $(S, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ 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 $\sqcap P$
- $\bot = \sqcap S$ 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 $+\infty$

Galoisverbindungen

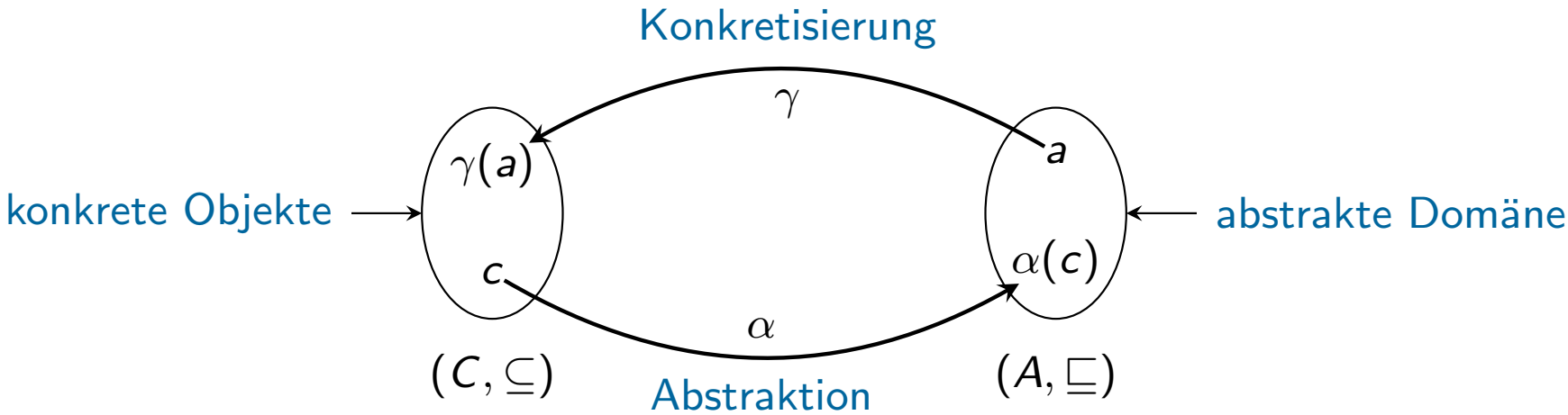


Galoisverbindung (engl. *galois connection*)

Eine Galoisverbindung $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ ist ein Paar (α, γ) von Abbildungen $\alpha : C \mapsto A$, $\gamma : A \mapsto C$ zwischen zwei partiell geordneten Mengen $(C, \subseteq)$ und $(A, \sqsubseteq)$:

- α und γ sind monoton
  - $\forall c, c' \in C : c \subseteq c' \Rightarrow \alpha(c) \sqsubseteq \alpha(c')$
  - $\forall a, a' \in A : a \sqsubseteq a' \Rightarrow \gamma(a) \sqsubseteq \gamma(a')$
- $\alpha \circ \gamma$ ist intensiv (verkleinernd): $\forall a \in A : \alpha(\gamma(a)) \sqsubseteq a$
- $\gamma \circ \alpha$ ist extensiv (erweiternd): $\forall c \in C : c \subseteq \gamma(\alpha(c))$

Abstraktion durch Galoisverbindungen



- wähle eine abstrakte Domäne (engl. *abstract domain*)
  - ersetzt die Menge konkreter Objekte S auf 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

Abstrakte Interpretation

- häufig verwendet man Galoiseinbettungen
  - diese sind Galoisverbindungen $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$
  - mit der Eigenschaft $\alpha(\gamma(a)) = a$
    - Konkretisierung gefolgt von Abstraktion impliziert keinen Präzisionsverlust
- nun benötigt man noch lokal konsistente Funktionen

Lokal konsistente Funktionen

Sei $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ eine Galoiseinbettung. Zwei Funktionen $f : C \mapsto C$ und $f' : A \mapsto A$ heißen lokal konsistent, falls gilt:

$$\forall c \in C : f(c) \subseteq \gamma(f'(\alpha(c)))$$

- 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

Abstrakte Interpretation (Forts.)

Abstrakte Interpretation

Eine Abstraktion Interpretation besteht aus

- einer Galoiseinbettung $(C, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (A, \sqsubseteq)$ und
- zwei lokal konsistenten Funktionen
  - $f : C \mapsto C$
  - $f' : A \mapsto A$

☞ Approximation von f durch die abstrakte Funktion f'

- . . . da war noch was . . . Fixpunkte!
  - auch hierfür kann man auf abstrakte Funktion zurückgreifen
    - kleinster Fixpunkt (engl. *least fixpoint, lfp*): $lfp(f) \subseteq \gamma(lfp(f'))$
    - größter Fixpunkt (engl. *greatest fixpoint, gfp*): $gfp(f) \subseteq \gamma(gfp(f'))$
  - bestimme den Fixpunkt der abstrakten Funktion und konkretisiere ihn
  - . . . terminiert das?

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 [2, Woche 5]

- Approximation der Potenzmenge über ganzen Zahlen $\mathcal{P}(\mathbb{Z})$

$$I = \bot \cup \{[x, y] \,|\, x \in \mathbb{Z} \cup \{-\infty\} \wedge y \in \mathbb{Z} \cup \{+\infty\} \wedge x \leq y\}$$

- die Galoiseinbettung $(\mathcal{P}(\mathbb{Z}), \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (I, \sqsubseteq)$ ist gegeben durch
  - die Konkretisierungsfunktion γ
  - die Abstraktionsfunktion α

$$\gamma : \begin{cases} \bot \mapsto \emptyset \\ [a, b] \mapsto \{n \in \mathbb{Z} | a \leq n \leq b\} \end{cases} \qquad \alpha : \begin{cases} \emptyset \mapsto \bot \\ S \mapsto [\min S, \max S] \end{cases}$$

  - Abstraktion $\rightsquigarrow$ bestimme die Randpunkte des Intervalls
  - Konkretisierung $\rightsquigarrow$ bestimme den Inhalt des Intervalls

Beispiel: Intervallabstraktion (Forts.)

- kleinste obere Schranke

$$X \sqcup \bot = X$$
$$\bot \sqcup Y = Y$$
$$[a, b] \sqcup [c, d] = [\min(a, c), \max(b, d)]$$

- größte untere Schranke

$$X \sqcap \bot = \bot$$
$$\bot \sqcap Y = \bot$$
$$[a, b] \sqcap [c, d] = \begin{cases} [\max(a, c), \min(b, d)]; \max(a, c) \leq \max(b, d) \\ \bot; \text{sonst} \end{cases}$$

Beispiel: Intervallabstraktion (Forts.)

- Addition von Intervallen

$$X + \bot = \bot$$
$$\bot + X = \bot$$
$$[a, b] + [c, d] = [a + c, b + d]$$

- Widening-Operator

$$X \triangledown \bot = X \quad \bot \triangledown X = X$$
$$[a, b] \triangledown [c, d] = \left[\begin{cases} -\infty; c < a \\ a; c \geq a \end{cases}, \begin{cases} +\infty; d < b \\ b; d \leq b \end{cases} \right]$$

- Narrowing-Operator

$$X \triangle \bot = X \quad \bot \triangle X = X$$
$$[a, b] \triangle [c, d] = \left[\begin{cases} c; a = -\infty \\ a; \text{sonst} \end{cases}, \begin{cases} d; b = +\infty \\ b; \text{sonst} \end{cases} \right]$$

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 4 $x_4 = (x_1 \cup x_5) \cap [-\infty, 9999]$

Zeile 5 $x_5 = x_4 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_5) \cap [10000, \infty]$

- Approximation mit Hilfe des Widening-Operators

Iteration 1:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 1]$

Zeile 5 $x_5 = [2, 2]$

Zeile 7 $x_7 = \emptyset$

Iteration 2:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, +\infty]$

Zeile 5 $x_5 = [2, +\infty]$

Zeile 7 $x_7 = \emptyset$

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 4 $x_4 = (x_1 \cup x_5) \cap [-\infty, 9999]$

Zeile 5 $x_5 = x_4 \oplus [1, 1]$

Zeile 7 $x_7 = (x_1 \cup x_5) \cap [10000, \infty]$

- Approximation mit Hilfe des Widening-Operators

Iteration 3:

Zeile 1 $x_1 = [1, 1]$

Zeile 4 $x_4 = [1, 9999]$

Zeile 5 $x_5 = [2, 10000]$

Zeile 7 $x_7 = [10000, 10000]$

☞ Konvergenz in der 3. Iteration!

Gliederung

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

  $\rightsquigarrow$ 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

[2] Midtgaard, J. :
Abstract Interpretation.
http://www.cs.au.dk/~jmi/AbsInt/, 2012