Semantik von Programmiersprachen

Vorlesung SoSe 2022
Wolfgang Mulzer

Jim Neuendorf

8. Juli 2022

Zusammenfassung

Diese Vorlesung vermittelt Techniken zur Formalisierung der Semantik (Bedeutungsinhalte) von Programmiersprachen. Zunächst werden unterschiedliche Formalisierungsansätze (die operationelle, denotationelle und axiomatische Semantik) vorgestellt und diskutiert. Anschließend wird die mathematische Theorie der semantischen Bereiche behandelt, die bei der denotationellen Methode, Anwendung findet. Danach wird schrittweise eine umfassende, imperative Programmiersprache entwickelt und die Semantik der einzelnen Sprachelemente denotationell spezifiziert. Dabei wird die Fortsetzungstechnik (continuation sem) systematisch erklärt und verwendet. Schließlich wird auf die Anwendung dieser Techniken eingegangen, insbesondere im Rahmen des Compilerbaus und als Grundlage zur Entwicklung funktionaler Programmiersprachen.

Inhaltsverzeichnis

24.06.

1 Denotationelle Semantik

Bemerkung (Ziel). Die Definition einer semantischen Funktion direkt ohne Umweg über eine simulierte Programmausführung (d. h. ohne eine Übergangsrelation).

Dabei ist die Zusammengsetztheit (*compositionality*) bei der Definition der semantischen Funktion für ein Sprachkonstrukt wichtig, d. h. man darf nur die Werte der semantischen Funktion für die Bestandteile des Sprachkonstrukts verwenden.

Beispiel.

$$\mathcal{A} : \text{AExp} \to \mathbb{Z}, \quad \mathcal{B} : \text{BExp} \to \{\text{w}, \text{f}\}$$

sind denotationell definiert.

Beispiel (Gegenbeispiel)**.** [while$^{\text{w}}_{\text{ns}}$]:

$$\frac{\langle S, \sigma\rangle \to \sigma'', \langle \texttt{while}\ b\ \texttt{do}\ S, \sigma''\rangle \to \sigma}{\langle \underbrace{\texttt{while}\ b\ \texttt{do}\ S}_{\text{gleiches Konstrukt}}, \sigma\rangle \to \sigma'}$$

1.1 Direkte Definition der semantischen Funktion

Definition 1.1. Wir definieren $\mathcal{S}_{\text{ds}} : \text{Stm} \to (\text{State} \to \text{State})$.

Die grundlegenden Definition bleiben gleich:

$$\begin{aligned}
\mathcal{S}_{\text{ds}}[\![\texttt{x := a}]\!](\sigma) &= \sigma[x \mapsto \mathcal{A}[\![a]\!](\sigma)] \\
\mathcal{S}_{\text{ds}}[\![\texttt{skip}]\!](\sigma) &= \sigma \\
\mathcal{S}_{\text{ds}}[\![S_1\texttt{;}\ S_2]\!](\sigma) &= (\underbrace{\mathcal{S}_{\text{ds}}[\![S_2]\!]}_{\text{State}\to\text{State}} \circ \mathcal{S}_{\text{ds}}[\![S_1]\!])(\sigma)
\end{aligned}$$

Achtung: Die Zustandsüberführungsfunktionen können eventuell nur partiell definiert sein. Dann liefert die Komposition auch nur $\perp$.

Zur Definition von Verzweigungen ($\rightsquigarrow$ `if`) benutzen wir eine Hilfsfunktion *cond* (siehe Definition 1.2). Damit ist die Definition von Verzweigungen wie folgt:

$$\mathcal{S}_{\text{ds}}[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{then}\ S_2]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\text{ds}}[\![S_1]\!], \mathcal{S}_{\text{ds}}[\![S_2]\!])$$

Definition 1.2 (cond)**.**

$$\text{cond} : (\text{State} \to \{\text{w}, \text{f}\}) \times (\text{State} \to \text{State}) \times (\text{State} \to \text{State}) \to (\text{State} \to \text{State})$$

$$\text{cond}(p, f_1, f_2) = \begin{cases} f_1(\sigma) & \text{falls } p(\sigma) = \text{w} \\ f_2(\sigma) & \text{falls } p(\sigma) = \text{f} \end{cases}$$

Herausforderung: Definition von $\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!]$ unter Berücksichtigung der Zusammengesetztheit.

Beobachtung: $\texttt{while } b \texttt{ do } S$ und $\texttt{if } b \texttt{ then (S; while } b \texttt{ do } S\texttt{) else skip}$ sollen semantisch äquivalent sein.

Also eigentlich wollen wir schreiben:

$$\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \text{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\text{ds}}[\![\underbrace{\texttt{while } b \texttt{ do } S}_{\text{gleiches Konstrukt}}]\!] \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

Das verletzt aber die Zusammengesetztheit.

Wenn wir diese Gleichung betrachten, erkennen wir

Bemerkung (Erkenntnis). $\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!]$ ist die Lösung der Gleichung

$$f = \text{cond}(\mathcal{B}[\![b]\!], f \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

Wie lösen wir diese Gleichung?

1.1.1 Problemtransfer zu Fixpunkt eines Funktionals

Wir überführen das Problem des Lösens der Gleichung zu einem Problem des Findens eines Fixpunktes.

Definition 1.3 (Funktional)**.** Dafür definieren wir ein *Funktional* (Argument und Ergebnis sind Funktionen):

$$F : (\text{State} \to \text{State}) \to (\text{State} \to \text{State})$$

durch

$$F(f) = \text{cond}(\mathcal{B}[\![b]\!], f \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

Ein *Fixpunkt* von F ist ein $f^* : \text{State} \to \text{State}$ mit $F(f^*) = f^*$, d. h. Stellen an denen die Funktion Werte auf sich selbst abbildet.

Durch diese Herangehensweise, können wir die Funktion auch im Umfeld der Lösung betrachten oder sie benutzen, um sich der Lösung anzunähern.

Definition 1.4. Wir definieren einen *Fixpunktoperator*

$$\text{FIX} : ((\text{State} \to \text{State}) \to (\text{State} \to \text{State}) \to (\text{State} \to \text{State}))$$

der einem Funktional F einen Fixpunkt f^* zuordnet.

Dann können wir schreiben

$$\mathcal{S}_{\text{ds}}[\![\texttt{while } b \texttt{ do } S]\!] = \text{FIX}(F)$$

wobei

$$F : f \mapsto \text{cond}(\mathcal{B}[\![b]\!], f \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

Das heißt die Semantik der `while`-Schleife ist der Fixpunkt des Funktionals.

Frage. Was können wir über FIX(f) sagen?

(a) Existiert für *jedes* Funktional $G : (\text{State} \to \text{State}) \to (\text{State} \to \text{State})$ ein Fixpunkt?

Nein.

Betrachte G mit

$$G(g) = \begin{cases} g_1 & \text{falls } g = g_2 \\ g_2 & \text{sonst} \end{cases}$$

für $g_1 \neq g_2$.

Aber vielleicht hat unser Funktional F immer einen Fixpunkt.

(b) Falls es einen Fixpunkt gibt, ist dieser eindeutig?

Im Allgemeinen nein.

Betrachte G mit

$$G(g) = g$$

Hier sind alle g Fixpunkte.

Aber vielleicht hat unser Funktional F immer höchstens einen Fixpunkt.

Wir müssen uns also anschauen, was der Fixpunktoperator FIX mit unserem speziellen F macht.

Bemerkung (Intuition). Wie löst man eigentlich eine Fixpunkt-Gleichung $f = F(f)$?

Fixpunkt-Iteration: Start mit

$$\begin{aligned} &f_0 \\ &f_1 = F(f_0) \\ &f_2 = F(f_1) \\ &f_3 = F(f_2) \\ &\dots \end{aligned}$$

Beispiel (Kein Fixpunkt)**.**

$$\begin{aligned} x &= 2x - 7 \\ x_0 &= 20 \\ x_1 &= 33 \\ x_2 &= 59 \\ x_3 &= 111 \\ &\dots \end{aligned}$$

01.07.

1.2 Der Fixpunktoperator

Bemerkung (Recap). Aus der letzten Vorlesung:

Aufgabe: Definiere die Semantik von `while` b `do` S denotationell.

Idee: Betrachte Funktional F (siehe Definition 1.3). Dann sollte die Zustandsüberführungsfunktion ein Fixpunkt von F sein, d. h. $f^* = F(f^*)$.

Frage. Woher wissen wir, dass F einen Fixpunkt besitzt? Wie sieht dieser Fixpunkt aus?

Versuch einer Visualisierung:

Wir haben $\mathcal{B}[\![b]\!]$ (Punkte sind Zustände mit Wahrheitswerten) und $\mathcal{S}[\![S]\!]$ (partielle Zustandsüberführungsfunktion, gelb), eine Funktion (Kandidat für Fixpunkt, blau) und das f-Funktional $F(f)$ (grün).

F erzeugt also eine neue Zustandsüberführungsfunktion, die für Zustände mit f auf den Zustand selbst abbildet und für Zustände mit w durch $f \circ \mathcal{S}$ abbildet, d. h. erst dem gelben, dann dem blauen Pfeil folgt.

Die Aufgabe des Findes eines Fixpunktes beduetet also, ein f (blau) zu finden, sodass nach Anwendung von $F(f)$ (grün) die gleichen Pfeile entstehen wie in f.

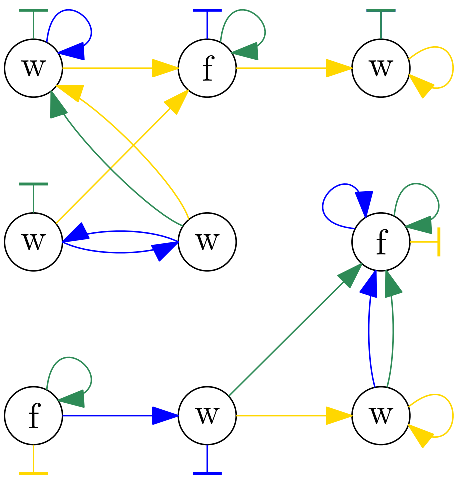
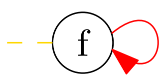


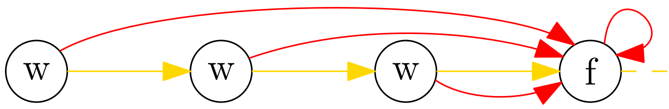
Abbildung 1: Visualisierung der Funktionsverkettung durch F

1.2.1 Wie muss ein Fixpunkt aussehen?

(a) Für alle Zustände mit Wahrheitswert f muss der Fixpunkt eine Schleife (id) liefern.



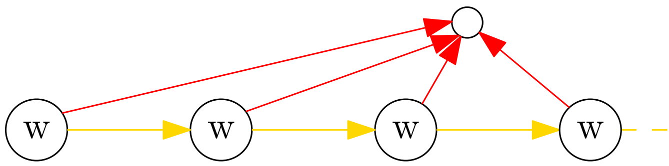
(b) Es gibt Zustände mit Wahrheitswert w, die nach endlich vielen Schritten mit $\mathcal{S}$ (gelb) einen Zustand mit Wahrheitswert f erreichen.



(c) Es gibt Zustände mit Wahrheitswert w, die nicht nach endliche vielen Schritten einen Zustand mit Wahrheitswert f erreichen.

(a) $w \to w \to w \to \ldots$

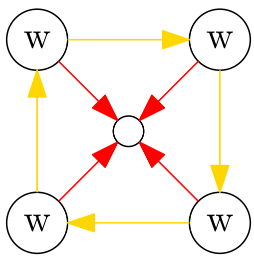
```
x := 1;
while true do
    x := x + 1
```



Hier müssen alle Pfeile zum selben Zustand zeigen, egal welcher genau das ist.

(b) $w_1 \to w_2 \to w_2 \to w_1$

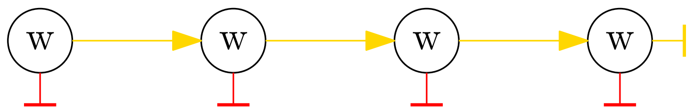
```
x := 0;
while x != -1 do
    x := (x + 1) mod 10
```



Hier müssen ebenfalls alle Pfeile zum selben Zustand zeigen, egal welcher genau das ist.

(c) $w \to w \to w \to \bot$

```
x := 1;
while x < 20 do
    x := x + 1;
    if x = 10 then
        while true do skip
    else
        ...
```



Für die Fälle (a) und (b) ist der Fixpunkt also fix aber beliebig (inklusive $\perp$).

Das Ergebnis unserer informellen Überlegung ist:

- Wir erwarten, dass *immer ein Fixpunkt existiert.*
- Ein solcher Fixpunkt ist nicht immer eindeutig. Für Zustände, bei denen die `while`-Schleife nicht terminiert, kann es ggf. mehrere Möglichkeiten für einen Fixpunkt geben.

  In diesem Fall hätten wir gern den Fixpunkt, der $\perp$ für nicht-terminierende Schleifen liefert.

1.2.2 Fixpunktiteration

Bemerkung (Idee). Finde die richtigen Fixpunkt durch Fixpunktiteration. Starte mit der Zustandsüberführungsfunktion

$$f_\perp : f_\perp(\sigma) = \perp \quad \forall \sigma \in \text{State}$$

welche F wiederholt auf $f_\perp$ an, schaue was passiert.

$\Rightarrow$ Der "Grenzwert" ist der gewünschte Fixpunkt.

Z. B. bei (c.a) und (c.b) bleibt die Funktion nach wiederholter Anwendung überall $\perp$.

Definition 1.5 (Fixpunktoperator)**.**

$$\begin{aligned} f_0 &= f_\perp \\ f_n &= F(f_{n-1}) \quad n \geq 1 \\ \text{FIX}(f) &:= \lim_{n \to \infty} f_n \end{aligned}$$

Aber wie ist der Grenzwert in diesem Kontext definiert?

Definition 1.6. Sei $\mathcal{F} = \{f \mid f : \text{State} \to \text{State}\}$ die Menge aller *partiellen* Zustandsüberführungsfunktionen. Wir definieren auf $\mathcal{F}$ eine Relation $\sqsubseteq$ durch

$$f \sqsubseteq g :\Longleftrightarrow \forall \sigma, \sigma' \in \text{State} : f(\sigma) = \sigma' \Rightarrow g(\sigma) = \sigma'$$

d. h. überall, wo f definiert ist, ist auch g definiert und liefert denselben Wert.

Beispiel. $g_1, g_2, g_3, g_4 : \text{State} \to \text{State}$

$$g_1(\sigma) = \sigma \quad \forall \sigma \in \text{State}$$
$$g_2(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(x) \geq 0 \\ \perp & \text{sonst} \end{cases}$$
$$g_3(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(x) \leq 0 \\ \perp & \text{sonst} \end{cases}$$
$$g_4(\sigma) = \begin{cases} \sigma & \text{falls } \sigma(x) = 0 \\ \perp & \text{sonst} \end{cases}$$

Behauptung:

$$g_4 \sqsubseteq g_2 \sqsubseteq g_1$$
$$g_4 \sqsubseteq g_3 \sqsubseteq g_1$$
$$g_4 \sqsubseteq g_1 \tag{*}$$

$(*)$ folgt aus Transivität da die Relation eine Ordnungsrelation ist, aber das haben wir noch nicht bewiesen.

Jedoch sind g_2 und g_3 nicht vergleichbar.

Bemerkung (Fakt). $\sqsubseteq$ ist eine Ordnungsrelation auf $\mathcal{F}$. Das bedeutet, sie ist

- reflexiv: $f \sqsubseteq f$
- transitiv: $f \sqsubseteq g \wedge g \sqsubseteq h \Rightarrow f \sqsubseteq h$
- antisymmetrisch: $f \sqsubseteq g \wedge g \sqsubseteq f \Rightarrow f = g$

$(\mathcal{F}, \sqsubseteq)$ ist eine *Halbordnung* (poset bzw. partially ordered set).

Definition 1.7. Ein Element $f \in \mathcal{F}$ heißt *Minimum*, falls für alle $g \in \mathcal{F}$ gilt

$$f \sqsubseteq g$$

Bemerkung (Beobachtungen). Im Allgemeinen, muss es kein Minimum in einem poset geben (z. B. unendliche Ordnung oder mehrere nicht vergleichbare minimale Elemente). Wenn ein *Minimum* existiert, dann ist es *eindeutig* (wegen Antisymmetrie).

Aber $(\mathcal{F}, \sqsubseteq)$ hat ein Minimum und zwar $f_\perp$.

08.07.

Bemerkung (Ziel für den Rest der Vorlesung). Konvergenz der Fixpunktiteration beweisen.

Definition 1.8 (Obere Schranke, Supremum)**.** Sei $\mathcal{G} \subseteq \mathcal{F}$, dann ist auch $(\mathcal{G}, \sqsubseteq)$ eine Halbordnung.

Ein Element $f \in \mathcal{F}$ heißt *obere Schranke* von $\mathcal{G}$, falls

$$\forall g \in \mathcal{G} : g \sqsubseteq f$$

Eine obere Schranke von $\mathcal{G}$ heißt *Supremum* "$\sup \mathcal{G}$", falls für alle oberen Schranken f' von $\mathcal{G}$ gilt $f \sqsubseteq f'$ d. h. es ist die kleinste obere Schranke.

Falls ein *Supremum* existiert, so ist es *eindeutig*. Falls die obere Schranke innerhalb von $\mathcal{G}$ liegt, so ist sie auch das Maximum und Supremum von $\mathcal{G}$.

Definition 1.9. Sei $\mathcal{G} \subseteq \mathcal{F}$. Wir nennen $\mathcal{G}$ eine *Kette* (chain), falls $\mathcal{G}$ total geordnet ist, d. h.

$$\forall g_1, g_2 \in \mathcal{G} : g1 \sqsubseteq g_2 \lor g_2 \sqsubseteq g_1$$

Beispiel. Folgende Beispiele sind Ketten:

- Die leere Menge $\varnothing \subseteq \mathcal{F}$ ist eine Kette.
- Jede 1-elementige Menge ist eine Kette.
- Sei $x \in \mathrm{Var}$ eine Variable. Für $n \in \mathbb{N}$ definiere wir $g_n \in \mathcal{F}$ durch

$$g_n : \mathrm{State} \to \mathrm{State}$$
$$g_n(\sigma) = \begin{cases} \bot & \text{falls } \sigma(x) > n \\ \sigma[x \mapsto -1] & \text{falls } \sigma(x) \in \{0, \dots, n\} \\ \sigma & \text{falls } \sigma(x) < 0 \end{cases}$$

Die Menge $\{g_n \mid n \in \mathbb{N}\}$ ist eine Kette in $(\mathcal{F}, \sqsubseteq)$, denn es gilt

$$g_n \sqsubseteq g_m \Leftrightarrow n \leq m$$

```
while x >= 0 do
    x := x - 1
```

Listing 1: Snippet für g_n

Ziel: Die Funktion

$$g_\infty : \mathrm{State} \to \mathrm{State}$$
$$g_\infty = \begin{cases} \sigma[x \mapsto -1] & \text{falls } \sigma(x) \geq 0 \\ \sigma & \text{sonst} \end{cases}$$

ist die Semantik von Listing 1.

Nun sehen wir, dass

$$g_\infty = \sup\{g_n \mid n \in \mathbb{N}\}$$

Definition 1.10 (Kettenvollständigkeit)**.** Eine Halbordnung heißt *kettenvollständig* (ccpo: chain-complete-partial order), falls jede Kette ein Supremum besitzt.

Beispiel. $(\mathbb{N}, \leq)$ ist nicht kettenvollständig. Die Menge der geraden Zahlen $\{2, 4, \dots\}$ ist eine Kette ohne Supremum.

$(\mathcal{P}(\{1, 2, 3\}), \subseteq)$ ist kettenvollständig.

$(\mathcal{P}(\mathbb{N}), \subseteq)$ ist auch kettenvollständig. Sei $X \in (\mathcal{P}(\mathbb{N}), \subseteq)$ eine Kette, also eine Menge von Teilmengen von $\mathbb{N}$, sodass $\forall A, B \in X : A \subseteq B \vee B \subseteq A$.
Es ist $\sup X = \bigcup_{A \in X} A = Z$, da

(a) Sei $A = X$. Dann ist $A \subseteq Z$.

(b) Z ist kleinste obere Schranke. Sei Y obere Schranke von X.

Zu zeigen: $Z \subseteq Y$.

Nimm $z \in Z$, zeige, dass $z \in Y$.

$$z \in Z \Rightarrow \exists A \subseteq Z \text{ mit } z \in A$$

Da Y obere Schranke von X ist und $a \in X$, muss $A \subseteq Y$. Also folgt, $z \in Y$.

Bemerkung (Fakt)*.* Sei $(\mathcal{D}, \sqsubseteq)$ ccpo. Dann besitzt $\mathcal{D}$ ein Minimum, nämlich

$$d_\perp = \sup \varnothing$$

Beweis. Sei $d_\perp = \sup \varnothing$. $d_\perp$ existiert nach Annahme. Und sei $d \in \mathcal{D}$. Nach Definition ist d eine obere Schranke von $\varnothing$. Da $d_\perp = \sup \varnothing$, muss $d_\perp \sqsubseteq d$ sein, d. h. $d_\perp$ ist Minimum. □

Satz 1.1. $(\mathcal{F}, \sqsubseteq)$ *ist kettenvollständig.*

Sei $\mathcal{G} \subseteq \mathcal{F}$ *eine Kette. Dann ist* $\sup \mathcal{G}$ *die Funktion mit*

$$\mathrm{graph}(\sup \mathcal{G}) = \bigcup \{\mathrm{graph}(g) \mid g \in \mathcal{G}\}$$

d. h.

$$(\sup \mathcal{G})(\sigma) = \sigma' \Leftrightarrow \exists g \in \mathcal{G} : g(\sigma) = \sigma'$$

Beweis. Wir müssen die folgenden Eigenschaften beweisen:

(a) Wohldefiniertheit

Seien $g_1, g_2 \in \mathcal{G}$ und sei $\sigma \in \text{State}$, sodass $g_1(\sigma) \neq \bot \neq g_2(\sigma)$. Da $\mathcal{G}$ eine Kette ist, muss gelten $g_1 \sqsubseteq g_2 \vee g_2 \sqsubseteq g_1$. In beiden Fällen folgt $g_1(\sigma) = g_2(\sigma)$. Also ist der Wert von $(\sup \mathcal{G})(\sigma)$ wohldefiniert.

(b) obere Schranke

Aus der Definition folgt direkt, dass $(\sup \mathcal{G})$ eine obere Schrank ist:

$$g \in \mathcal{G},\ \sigma, \sigma' \in \text{State},\ g(\sigma) = \sigma' \implies (\sup \mathcal{G})(\sigma) = \sigma'$$

(c) kleinste obere Schranke

$(\sup \mathcal{G})$ ist obere Schranke. Sei h eine obere Schranke.

Zu zeigen: $(\sup \mathcal{G}) \sqsubseteq h$.

Sei $\sigma, \sigma' \in \text{State}$. $(\sup \mathcal{G})(\sigma) = \sigma'$, d. h. $\exists g \in \mathcal{G} : g(\sigma) = \sigma'$, d. h. h ist obere Schranke, d. h. $g \sqsubseteq h$, also muss $h(\sigma) = \sigma'$.

$\square$

Idee der Fixpunktiteration:

$$\begin{aligned} f_0 &= f_\bot \\ f_1 &= F(f_0) \\ &\vdots \\ f_\infty &= \sup \{ f_n \mid n \in \mathbb{N} \} \end{aligned}$$

Dann soll gelten: $f_\infty = F(f_\infty)$.

Wenn also gelten würde

$$F\left(\lim_{n\to\infty} f_n\right) = \lim_{n\to\infty} F(f_n)$$

wären wir fertig. Das ist die *Stetigkeit*.