Semantik von Programmiersprachen

Vorlesung SoSe 2022
Wolfgang Mulzer

Jim Neuendorf

3. Juni 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

29.04.

1 Semantik

Ziel: Finde eine mathematische Methode, um einem Programm eine *Bedeutung* zuzuordnen.

Motivation:

- Verifikation:
  - Erfüllt mein Programm die Spezifikation (tut es das, was es soll)?
  - Setzt der Übersetzer/Interpretierer die Spezifikation der Sprache korrekt um?
- Programmumformung
  - Haben zwei unterschiedliche Programme die gleiche Bedeutung?
  - Optimierung
- Programmanalyse
  - Ist das Programm "sicher" (secure vs. safe)?
  - Ist das Programm "effizient"?

Definition 1.1 (Programmierparadigma)**.** Programmierparadigma: z. B. deklarativ ("Was?") (funktional vs. logisch), imperativ ("Wie?"). In verschiedenen Paradigmen haben (potenziell) Programme verschiedene Bedeutungen.

Wir konzentrieren uns auf *imperative* Programmierung.

Frage. Was ist die "mathematische Bedeutung" eines imperativen Programms?

Frage (folgend)*.* Was ist ein imperatives Programm?

```
x = 1
y = x + 2
x = y + 5
for ...
```

Listing 1: Imperatives Programm

```
foo :: Int -> Int
foo 0 = 1
foo x = x + 1
foo 3
```

Listing 2: Funktionales Programm

Das zentrale Konzept der imperativen Programmierung ist der *Zustand* (state). Der Zustand ist der Inhalt aller Speicherzellen und Register, die Position des Programmzählers und der Zustand der Eingabe-/Ausgabe-Geräte.

Ein imperatives Programm ist eine Folge von *Anweisungen* (statement / instruction). Diese haben *Wirkungen* (effects), welche den Zustand verändern (selbst `nop` ändert den Programmzähler und somit den Zustand). Darüber hinasu gibt es Nebenwirkungen bzw. Seiteneffekte (side effects). Es gibt unterschiedliche Arten von Anweisungen:

- Zuweisungen (direkte Änderung des Zustandes)
- Kontrollfluss (Änderung des Programmzählers: Verzweigungen, Schleifen, Funktionsaufrufe bzw. Sprünge)
- Eingabe / Ausgabe

2 Mathematische Formalisierung

Definition 2.1 (Zustand)**.** Es gibt eine abzählbar unendliche Menge von Variablen $V = \{x_1, x_2, \ldots, y, z, \ldots\}$ (Speicher ist begrenzt aber beliebig groß). Der Zustand ist eine (partielle) Funktion

$$\sigma : V \to \mathbb{Z} \cup \{\bot, true, false\}$$

($\bot$ bedeutet undefiniert, d. h. eine Speicherzelle hat noch keinen Wert und die Funktion gibt nichts aus).

Die Teile des Zustandes "Eingabe / Ausgabe" ignorieren wir erst einmal, d. h. die initiale Eingabe ist implizit durch den Wert der Variablen am Anfang. Der Programmzähler wird an anderer Stelle thematisiert.

Bemerkung. Diese Definition dient als Bespiel, d. h. in anderen Szenarion mit anderen Variables außer Ganzzahlen und Boolesche Wert kann eine andere Definition sinnvoller sein.

Definition 2.2 (Imperatives Programm)**.** Ein imperatives Programm ist eine Funktion auf der Menge alles Zustände. Jedem Startzustand wird ein Endzustand zugeordnet (wir ignorieren E/A).

Notation. Sei $\Pi \in \Sigma^*$ ein gültiges Programm (eine Zeichenkette). Wir bezeichnen mit

$$S[\![\Pi]\!] \in [State \to State]$$

(S ist die semantische Funktion) die Funktion, welche durch Π definiert wird.

2.1 `while`-Sprache

Definition 2.3. Wir verwenden in dieser Vorlesung eine einfache, turing-vollständige, imperative Programmiersprache als durchgängiges Beispiel namens `while`-*Sprache*, die durch folgende kontextfreie Grammatik gegeben ist:

$$\begin{aligned}
A &\to \texttt{Zahl} \mid \texttt{Var} \mid A + A \mid A * A \mid A - A \\
B &\to \texttt{true} \mid \texttt{false} \mid A = A \mid A \leq A \mid \neg B \mid B \wedge B \\
S &\to \texttt{Var := A} \mid \texttt{skip} \mid \texttt{S; S} \mid \texttt{if B then S else S} \mid \texttt{while B do S}
\end{aligned}$$

Bemerkung. Es gibt die syntaktischen Kategorien "arithmetischer Ausdruck" (A), "Boolescher Ausdruck" (B) und "Statement" (S, Anweisung).

Beispiel.

$$\Pi = \texttt{x = z + 1}$$

$$\mathcal{S}[\![\texttt{x = z + 1}]\!](\underbrace{[x \mapsto 5, z \mapsto -4, a \mapsto 2]}_{\text{Startzustand}}) = \underbrace{[x \mapsto -3, z \mapsto 4, a \mapsto 2]}_{\text{Endzustand}}$$

$$\mathcal{S}[\![\texttt{x = z + 3}]\!]([x \mapsto 10, z \mapsto 12]) = [x \mapsto 10, z \mapsto 12]$$

Für diese Veranstaltung stellen wir uns die Frage: Wie komme ich von Π zu $S[\![\Pi]\!]$?

Dafür gibt es drei Ansätze:

(a) axiomatische Semantik

(b) operationelle Semantik

(c) denotationelle Semantik

2.2 Axiomatische Semantik

Wir verzichten auf die vollständige Spezifikation von $S[\![\cdot]\!]$. Stattdessen arbeiten wir mit *Zusicherungen* (Assertions), welche wesentliche Aspekte des Zustands zu einem gegebenen Zeitpunkt widerspiegeln.

Wir definieren ein logisches System, das Beziehungen zwischen Zuständen aufstellt (Vorbedingungen, Nachebdingungen). Das System muss $S[\![\cdot]\!]$ verträglich sein.

Die Details sind Thema einer anderen Vorlesungen, z. B. Hoare-Kalkül.

Beispiel.

$$\underbrace{\{x = n \land y = m\}}_{\text{Vorbedingung}} \quad \texttt{z := x; x := y; y := z} \quad \underbrace{\{x = m \land y = n\}}_{\text{Nachbedingung}}$$

2.3 Operationelle Semantik

Definiere $S[\![\Pi]\!]$ durch schrittweise Simulation der Ausführung von Π (ein Interpretierer in mathematischer Form / Abstraktion).

Genauer gesagt bedeutet das: Wir definieren ein *Transitionssystem*

$$\langle \Pi, s\rangle \Rightarrow \langle \Pi', s'\rangle$$
$$\langle \Pi, s\rangle \Rightarrow s'$$

das die Ausführung von Π auf Zustand s darstellt.

Beispiel.

$$\begin{aligned}
&\langle \texttt{'z = x; x = y; y = z;'}, [x \mapsto 2, y \mapsto 3, z \mapsto 6]\rangle \\
\rightsquigarrow\ &\langle \texttt{'x = y; y = z;'}, [x \mapsto 2, y \mapsto 3, z \mapsto 2]\rangle \qquad \text{(1. Befehl ausgeführt)} \\
\rightsquigarrow\ &\langle \texttt{'y = z;'}, [x \mapsto 3, y \mapsto 3, z \mapsto 2]\rangle \\
\rightsquigarrow\ &[x \mapsto 3, y \mapsto 2, z \mapsto 2] \qquad \text{(Endzustand)}
\end{aligned}$$

06.05.

2.4 Denotationelle Semantik

Definiere $S[\![\Pi]\!]$ direkt als mathematische Funktion anhand der Syntax von Π, z. B.

$$\mathcal{S}[\![\texttt{z := x; x := y; y := z}]\!] = \mathcal{S}[\![\texttt{y := z}]\!] \circ \mathcal{S}[\![\texttt{x := y}]\!] \circ \mathcal{S}[\![\texttt{z := x}]\!]$$

Es wird also z. B. die sequenzielle Ausführung von Anweisungen als Funktionskomposition übersetzt.

Bemerkung (Problem). Wie kann man beispielsweise Schleifen darstellen (insbesondere `while`)? Ein möglicher Ansatz sind Grenzwerte, aber das geht tiefer in die Analysis.

Bei der operationellen Semantik wird die Schleife durch das Transitionssystem realisiert.

3 Operationelle Semantik

Das Folgende bezieht sich auf die `while`-Sprache (siehe Abschnitt 2.1).

Definition 3.1. AExp, BExp, SExp: Mengen aller gültigen Ableitungen aus A, B bzw. S als Syntaxbaum. Der Ausdruck `5+7-2*8` lässt sich aus A ableiten. Der entsprechende Syntaxbaum (siehe Abbildung 1) ist dann Teil von AExp.

$$a \in \text{AExp}$$

Mengen wie AExp bezeichnen wir als **syntaktische Kategorien**.

Zahl ist eine ganze Zahl aus $\mathbb{Z}$. Die zugehörige syntaktische Kategorie Num. Num ist die Menge aller Zeichenkette, die ganze Zahlen darstellen.

$$\texttt{1234} \in \text{Num}$$
$$1234 \in \mathbb{Z}$$

Beachte, dass eigentlich der Syntaxbaum von `1234` gemeint ist

Var sind Variablen, die nach Belieben vorhanden sind. Es sind abzählbar unendlich viele.

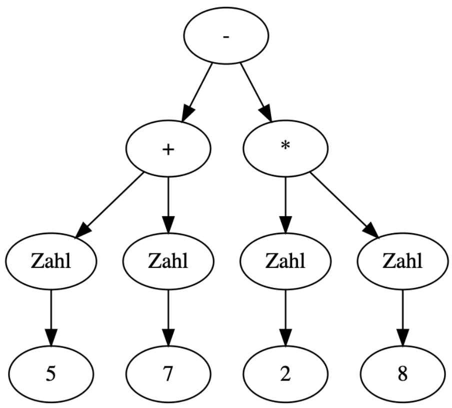

Abbildung 1: `5+7-2*8` $\in$ AExp als verkürzt dargestellter Syntaxbaum

Bemerkung. Unterbäume können auch Elemente einer anderen Kategorie sein.

Beispiel. Division mit Rest

- Eingabe: $a, b > 0$
- Ausgabe: $m, r \geq 0$, $r < b$, $a = m \cdot b + r$

```
m := 0;
while b <= a do (
    m := m + 1;
    a := a - b
)
r := a
```

Listing 3: Division mit Rest

Die *Semantik* in `while` wird gegeben durch eine *semantische Funktion*, eine für jede *syntaktische Kategorie*.

Sei $State = \{\sigma \mid \sigma : \mathrm{Var} \to \mathbb{Z}\}$, z. B.

$$\begin{aligned} &\mathcal{N} : \mathrm{Num} \to \mathbb{Z} \\ &\mathcal{N}[\![\texttt{-123}]\!] = -123 \end{aligned}$$

$$\mathcal{A} : \underbrace{\mathrm{AExp}}_{\text{“Compiler”}} \to \underbrace{(\mathrm{State} \to \mathbb{Z})}_{\text{“Interpreter”}}$$

$$\mathcal{A}[\![\texttt{x+x*5}]\!] = (\sigma \mapsto 6 \cdot \sigma(x)) \quad (\text{da } \texttt{x+x*5} = \texttt{6*x})$$

$$\mathcal{B} : \mathrm{BExp} \to (\mathrm{State} \to \{w, f\})$$

$$\mathcal{B}[\![\texttt{x<=10}]\!] = \left(\sigma \mapsto \begin{cases} w & \sigma(x) \leq 10 \\ f & \sigma(x) > 10 \end{cases}\right)$$

$$S : \mathrm{SExp} \to (\mathrm{State} \rightharpoonup \mathrm{State})$$

Jetzt: Definition von $\mathcal{A}$ durch Induktion über die Struktur des Syntaxbaums. Die Definition von $\mathcal{N}$ und $\mathcal{B}$ ist eine Übung.

Später: Definition von S.

3.1 Semantik arithmetischer Ausdrücke

Definition 3.2 (Induktive Definition von $\mathcal{A}$)**.** Sei $n \in \mathrm{Num}, x \in \mathrm{Var}, \sigma \in \mathrm{State}$ und $a_1, a_2 \in \mathrm{AExp}$. Wir definieren:

(i) $\mathcal{A}[\![n]\!](\sigma) = \mathcal{N}[\![n]\!]$ (konstante Funktion)

(ii) $\mathcal{A}[\![x]\!](\sigma) = \sigma(x)$ alternativ: $\mathcal{A}[\![n]\!] = (\sigma \mapsto \sigma(n))$

(iii) $\mathcal{A}[\![a_1 + a_2]\!](\sigma) = \mathcal{A}[\![a_1]\!](\sigma) + \mathcal{A}[\![a_2]\!](\sigma)$

Bemerkung. In anderen Sprachen könnte der Aufruf des ersten Summanden Seiteneffekte haben, d. h. ggf. muss man an dieser Stelle aufpassen. In diesem Fall würde der potentenziell veränderte Zustand mit zurückgegeben werden.

(iv) Analog für Subtraktion

(v) Analog für Multiplikation

Bemerkung. Für die Definition von semantischen Funktionen fordern wir **Zusammengesetztheit**, d. h. in der induktiven Definition darf nur auf Bestandteile des Ausdrucks/Syntaxbaums zugegriffen werden.

Beispiel. Führe Negation ein: $\mathcal{A} \to \ldots \mid -A$.

Erlaubt ist $\mathcal{A}\llbracket -a_1 \rrbracket(\sigma) = 0 - \mathcal{A}\llbracket a_1 \rrbracket(\sigma)$ aber nicht $\mathcal{A}\llbracket -a_1 \rrbracket(\sigma) = \mathcal{A}\llbracket 0 - a_1 \rrbracket(\sigma)$, da der Ausdruck hier um eine Null erweitert wurde.

13.05.

Satz 3.1. *$\mathcal{A}$ besitzt die folgenden Eigenschaften:*

(a) für alle $a \in \mathrm{AExp}$, für alle $\sigma \in \mathrm{State}$ existiert genau eine Zahl $n \in \mathbb{Z}$, sodass $\mathcal{A}\,[\![a]\!](\sigma) = n$. (Voraussetzung: σ ist eine totale Funktion.)

(b) $\mathcal{A}$ ist eine totale Funktion.

Beweis. Skizze:

(b) folgt aus (a)

(a) wird bewiesen durch strukturelle Induktion nach a.

□

Nächste Schritte:

(i) Der Wert eines Ausdrucks hängt *nur* von den Variables ab, die in ihm vorkommen. (Abschnitt 3.2)

(ii) Was passiert in einem arithmetischen Ausdruck, wenn wir eine Variable durch einen Ausdruck ersetzen ($\rightsquigarrow$ *Substitution*)? (Abschnitt 3.3)

3.2 Freie Variablen

Definition 3.3 (Freie Variablen)**.** Sei $a \in \mathrm{AExp}$. $\mathrm{FV}(a)$ ("freie Variablen") ist die Menge aller Variablen, die in a vorkommen. Formal ist das induktiv definiert:

(a) $\mathrm{FV}(n) = \emptyset$ (falls $a = n$ (Zahl))

(b) $\mathrm{FV}(x) = \{x\}$ (falls $a = x$ (Variable))

(c) $\mathrm{FV}(a_1 \square a_2) = \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)$ für $\square \in \{+, -, *\}$

Gebundene Variable betrachten wir im Kontekt der `while`-Sprache nicht. in anderen Sprachen können aber lokale Variablen z. B. als solche betrachtet werden.

Lemma 3.4. *Sei $a \in \mathrm{AExp}$, sei $\sigma, \sigma' \in \mathrm{State}$, sodass $\sigma(x) = \sigma'(x)$ für alle x in $\mathrm{FV}(a)$ gilt.*

Dann gilt:

$$\mathcal{A}\,[\![a]\!](\sigma) = \mathcal{A}\,[\![a]\!](\sigma')$$

Beweis. Durch strukturelle Induktion nach a.

Induktionsanfang:

(a) $a = n$, n Zahl

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma) &= \mathcal{A}\,[\![n]\!](\sigma)\\ &\overset{\text{D. }3.2(i)}{=} \mathcal{N}\,[\![n]\!]\end{aligned}$$

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma') &= \mathcal{A}\,[\![n]\!](\sigma')\\ &= \mathcal{N}\,[\![n]\!]\end{aligned}$$

(b) $a = x$, x Variable

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma) &= \mathcal{A}\,[\![x]\!](\sigma)\\ &\overset{\text{D. }3.2(ii)}{=} \sigma(x)\end{aligned}$$

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma') &= \mathcal{A}\,[\![x]\!](\sigma')\\ &= \sigma'(x)\end{aligned}$$

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

Nach Annahme ist $\sigma(x) = \sigma'(x)$, da $x \in \mathrm{FV}(a)$. Daher folgt

$$\mathcal{A}\,[\![a]\!](\sigma) = \mathcal{A}\,[\![a]\!](\sigma')$$

Induktionsschritt:

$a = a_1 \,\Box\, a_2$ mit $\Box \in \{+, -, *\}$

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma) &= \mathcal{A}\,[\![a_1 \,\Box\, a_2]\!](\sigma)\\ &= \mathcal{A}\,[\![a_1]\!](\sigma) \,\Box\, \mathcal{A}\,[\![a_2]\!](\sigma)\end{aligned}$$

$$\begin{aligned}\mathcal{A}\,[\![a]\!](\sigma') &= \mathcal{A}\,[\![a_1 \,\Box\, a_2]\!](\sigma')\\ &= \mathcal{A}\,[\![a_1]\!](\sigma') \,\Box\, \mathcal{A}\,[\![a_2]\!](\sigma')\end{aligned}$$

$$\begin{aligned}\mathrm{FV}(a) &= \mathrm{FV}(a_1 \,\Box\, a_2)\\ &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)\end{aligned}$$

Insbesondere gilt, dass $\mathrm{FV}(a_1) \subseteq \mathrm{FV}(a)$ und $\mathrm{FV}(a_2) \subseteq \mathrm{FV}(a)$. Daher folgt:

Da nach Annahme $\sigma(x) = \sigma(x')$ für alle $x \in \mathrm{FV}(a)$, gilt

$$\sigma(x) = \sigma'(x) \text{ für alle } x \in \mathrm{FV}(a_1)$$

und

$$\sigma(x) = \sigma'(x) \text{ für alle } x \in \mathrm{FV}(a_2)$$

Also gelten nach Induktionsvoraussetzung

$$\mathcal{A}\,[\![a_1]\!](\sigma) = \mathcal{A}\,[\![a_1]\!](\sigma') \wedge \mathcal{A}\,[\![a_2]\!](\sigma) = \mathcal{A}\,[\![a_2]\!](\sigma')$$

Daher folgt:

$$\mathcal{A}\,[\![a_1]\!](\sigma) \,\Box\, \mathcal{A}\,[\![a_2]\!](\sigma) = \mathcal{A}\,[\![a_1]\!](\sigma') \,\Box\, \mathcal{A}\,[\![a_2]\!](\sigma')$$

□

3.3 Substitution

Definition 3.5 (Substitution für Ausdrücke)**.** Seien $a, a_0 \in \mathrm{AExp}$, $y \in \mathrm{Var}$. Wir definieren $a[y \mapsto a_0]$ als den arithmetischen Ausdruck, in dem jedes Vorkommen von y in a durch a_0 ersetzt wird.

Formal:

(i) $n[y \mapsto a_0] = n$

(ii) $x[y \mapsto a_0] = \begin{cases} a_0 & \text{falls } x = y \\ x & \text{sonst} \end{cases}$

(iii) $(a_1 \,\Box\, a_2)[y \mapsto a_0] = a_1[y \mapsto a_0] \,\Box\, a_2[y \mapsto a_0] \qquad \text{für } \Box \in \{+, -, *\}$

Beispiel.

$$a = \texttt{x + y*2 - z(x + y)}$$
$$\texttt{y} \mapsto \texttt{4*t + 5}$$

$$a[\texttt{y} \mapsto \texttt{4*t + 5}] = \texttt{x + (4*t + 5)*2 - z(x + (4*t + 5))}$$

Es werden die Blätter am Syntaxbaum ersetzt, woraus implizit die Präzedenz klar ist (wodurch die obigen Klammern entstehen).

Ersetzungen dürfen die zu ersetzende Variable enthalten. Da nur einmal ersetzt wird, ist das in Ordnung.

Definition 3.6 (Substitution für Zustände)**.** Sei $\sigma \in \mathrm{State}$, $x \in \mathrm{Var}$, $n \in \mathbb{Z}$. Dann ist $\sigma[x \mapsto n] \in \mathrm{State}$ der Zustand, der wie folgt definiert ist:

$$\sigma[x \mapsto n](z) = \begin{cases} n & \text{falls } z = x \\ \sigma(z) & \text{sonst} \end{cases}$$

Lemma 3.7. *Sei* $a, a_0 \in \mathrm{AExp}$, $y \in \mathrm{Var}$, $\sigma \in \mathrm{State}$. *Dann gilt:*

$$\mathcal{A}\,[\![a[y \mapsto a_0]]\!](\sigma) = \mathcal{A}\,[\![a]\!]\Big(\sigma\big[y \mapsto \mathcal{A}\,[\![a_0]\!](\sigma)\big]\Big)$$

Das Lemma setzt Semantik und Syntax in Verbindung: Wir können syntaktisch Variable durch Teilausdrücke ersetzen und das ist äquivalent dazu, erst den Teilausdruck auszuwerten und mit diesen Wert im Zustand den ursprünglichen Ausdruck auszuwerten.

Beispiel.

$$\begin{aligned} a &= x + y \\ a_0 &= x * y \\ \sigma &: [x \mapsto 2; y \mapsto 10] \end{aligned}$$

$$\begin{aligned} (x + (x * y))[x \mapsto 2; y \mapsto 10] &\ \square\ (x + y)[x \mapsto 2; y \mapsto 20] \\ 2 + (2 * 10) &\ \square\ 2 + 20 \\ 22 &\ = 22 \end{aligned}$$

Beweis. Durch strukturelle Induktion nach a (nicht a_0!).

Induktionsanfang:

(a) $a = n$, n Zahl

$$\begin{aligned} \mathcal{A}[\![a[y \mapsto a_0]]\!](\sigma) &= \mathcal{A}[\![n[y \mapsto a_0]]\!](\sigma) \\ &\overset{\text{D. } 3.5(i)}{=} \mathcal{A}[\![n]\!](\sigma) \\ &= n \end{aligned}$$

$$\mathcal{A}[\![n]\!](\sigma[y \mapsto \mathcal{A}[\![a_0]\!](\sigma)]) = n \qquad \text{(rechte Seite)}$$

(b) $a = x$, x Variable

$$\mathcal{A}[\![x[y \mapsto a_0]]\!](\sigma) \overset{\text{D. } 3.5(iii)}{=} \begin{cases} \mathcal{A}[\![a_0]\!](\sigma) & \text{falls } x = y \\ \mathcal{A}[\![x]\!](\sigma) = \sigma(x) & \text{sonst} \end{cases}$$

$$\begin{aligned} \mathcal{A}[\![x]\!](\sigma[y \mapsto \mathcal{A}[\![a_0]\!](\sigma)]) &\overset{\text{D. } 3.6}{=} (\sigma[y \mapsto \mathcal{A}[\![a_0]\!](\sigma)])(x) \qquad \text{(rechte Seite)} \\ &= \begin{cases} \mathcal{A}[\![a_0]\!](\sigma) & \text{falls } x = y \\ \sigma(x) & \text{sonst} \end{cases} \end{aligned}$$

Induktionsschritt: Straight forward. $\square$

20.05.

3.4 Natürliche operationelle Semantik ("big step")

$\mathcal{A}[\![\cdot]\!]$ und $\mathcal{B}[\![\cdot]\!]$ *benutzen* den Zustand σ während die Semantik von Anweisungen den Zustand *verändern* soll.

Idee: Definiere $\mathcal{S}[\![\cdot]\!]$ mithilfe einer Zustandsüberführungerelation.

Definition 3.8. Sei $s \in$ SExp eine Anweisungsfolge und seien $\sigma, \sigma' \in$ State Zustände. Die Zustandsüberführungsrelation

$$\langle S, \sigma \rangle \to \sigma'$$

spezifiziert die Beziehung zwischen Startzustand σ und dem Endzustand σ' gemäß der Anweisungsfolge S.

Bemerkung (Bedeutung). Die Ausführung von S auf Startzustand σ terminiert mit Endzustand σ'.

Notation. Wir definieren die Zustandsübergangsrelation mithilfe von Schlussregeln. Eine solche Schlussregel besitzt die folgende Form:

$$\frac{\text{Voraussetzung}}{\text{Folgerung}} \rightsquigarrow \frac{\langle S_1, \sigma_1 \rangle \to \sigma_1', \langle S_2, \sigma_2 \rangle \to \sigma_2', \ \ldots\ , \langle S_n, \sigma_n \rangle \to \sigma_n'}{\langle S, \sigma \rangle \to \sigma'}$$

(ggf. mit zusätzlichen Bedingungen) wobei $S_1, \ldots, S_n$ Bestandteile von S sind.

Gibt es keine Voraussetzung, nennen wir die Schlussregel ein *Axiom*.

3.4.1 Schlussregeln für die natürliche Semantik von `while`

Im Folgenden schreiben wir $[\cdot]_{\text{ns}}$ um anzuzeigen, dass es sich um die natürliche Semantik handelt.

(a) Zuweisung $[\text{zuw}_{\text{ns}}]$ (Axiom)

$$\frac{}{\langle x \texttt{ := } a, \sigma \rangle \to \sigma[x \mapsto \mathcal{A}[\![a]\!](\sigma)]}$$

(b) Skip $[\text{skip}_{\text{ns}}]$ (Axiom)

$$\frac{}{\langle \texttt{skip}, \sigma \rangle \to \sigma}$$

(c) Hintereinanderausführung $[\text{seq}_{\text{ns}}]$

$$\frac{\langle S_1, \sigma \rangle \to \sigma', \langle S_2, \sigma' \rangle \to \sigma''}{\langle S_1 ; S_2, \sigma \rangle \to \sigma''}$$

Dabei können S_1 und S_2 zusammengesetzte Anweisungen sein.

(d) Verzweigung $[\text{if}^{\text{w}}_{\text{ns}}]$

$$\frac{\langle S_1, \sigma\rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, \sigma\rangle \to \sigma'}$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{w}$. Dabei muss S_2 nicht terminieren.

(e) Verzweigung $[\text{if}^{\text{f}}_{\text{ns}}]$

$$\frac{\langle S_2, \sigma\rangle \to \sigma'}{\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, \sigma\rangle \to \sigma'}$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{f}$. Dabei muss S_1 nicht terminieren.

(f) Schleife $[\text{while}^{\text{w}}_{\text{ns}}]$

$$\frac{\langle S, \sigma\rangle \to \sigma', \langle \texttt{while } b \texttt{ do } S, \sigma'\rangle \to \sigma''}{\langle \texttt{while } b \texttt{ do } S, \sigma\rangle \to \sigma''}$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{w}$. Im Allgemeinen kann es sein, dass die Schleife nicht terminiert. Deshalb müssen wir diesen Fall in der Definition der semantischen Funktion beachten. Das bedeutet, diese Relation ist nicht total.

(g) Schleife $[\text{while}^{\text{f}}_{\text{ns}}]$

$$\frac{}{\langle \texttt{while } b \texttt{ do } S, \sigma\rangle \to \sigma}$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{f}$.

Wie ist die Zustandsüberführungsfunktion überhaupt definiert?

Definition 3.9. Sei $S \in \text{SExp}$ ein Programm und seien $\sigma, \sigma' \in \text{State}$. Dann gilt

$$\langle S, \sigma\rangle \to \sigma'$$

gdw. ein *endlicher Ableitungsbaum* dafür existiert.

Der Ableitungsbaum entsteht durch wiederholte Anwendung der Schlussregeln. Die Wurzel ist ist $\langle S, \sigma\rangle \to \sigma'$, die Blätter sind Axiome, die Knoten entsprechen korrekte Anwendung der Schlussregeln.

Beispiel. Sei $\sigma \in \text{State}$ mit $\sigma(x) = 1$ und $\sigma(y) = 5$.

Behauptung: $\langle (z := z; x := y); y := z, \sigma\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]$

Nun müssen wir den endlichen Ableitungsbaum erzeugen.

(a) $[\text{seq}_{\text{ns}}]$

$$\frac{\langle z := x; x; = y, \sigma\rangle \to \sigma', \langle y := z, \sigma'\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]}{\langle \underbrace{(z := z; x := y)}_{s_1}; \underbrace{y := z}_{S_2}, \sigma\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]}$$

Welches σ' brauchen wir? $\rightsquigarrow \sigma[z \mapsto 1][x \mapsto 5]$. Somit erhalten wir

$$\frac{\langle z := x; x := y, \sigma\rangle \to \sigma[z \mapsto 1][x \mapsto 5], \langle y := z, \sigma[z \mapsto 1][x \mapsto 5]\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]}{\langle \underbrace{(z := z; x := y)}_{S_1}; \underbrace{y := z}_{S_2}, \sigma\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]}$$

(b) [zuw_{ns}] für den rechten Teil

$$\frac{\mathcal{A}[\![z]\!](\sigma[z \mapsto 1][x \mapsto 5]) \stackrel{?}{=} 1}{\langle y := z, \sigma[z \mapsto 1][x \mapsto 5]\rangle \to \sigma[z \mapsto 1][x \mapsto 5][y \mapsto 1]}$$

$$\mathcal{A}[\![z]\!](\sigma[z \mapsto 1][x \mapsto 5]) \stackrel{Def}{=} \sigma[z \mapsto 1][x \mapsto 5](z) \stackrel{Def}{=} 1$$

(c) [seq_{ns}]

$$\frac{\langle z := x, \sigma\rangle \to \sigma[z \mapsto 1], \langle x := y, \sigma[z \mapsto 1]\rangle \to \sigma[z \mapsto 1][x \mapsto 5]}{\langle z := x; x; = y, \sigma\rangle \to \sigma[z \mapsto 1][x \mapsto 5]}$$

Ab hier analog zum rechten Teil (mit zwei Mal [zuw_{ns}]).

Bemerkung (Rechtseindeutigkeit / Determiniertheit). Es ist noch nicht klar, dass "$\to$" *rechtseindeutig* ist. D. h. möglicherweise existiert ein Programm S, ein Startzustand σ und Zustände $\sigma_1 \neq \sigma_2 \in \text{State}$, sodass sowohl

$$\langle S, \sigma\rangle \to \sigma_1 \text{ als auch } \langle S, \sigma\rangle \to \sigma_2$$

gilt. Daher muss die Rechtseindeutigkeit bzw. *Determiniertheit* bewiesen werden!

Bemerkung (Ableitungsbaum). Der Ableitungsbaum ist statisch. D. h. man kann nicht erkennen, in welcher Reihenfolge die Schlussregeln angewendet werden.

27.05.

Definition 3.10. Sei S ein Programm und σ ein Startzustand. Das Programm S *terminiert* bei Startzustand σ falls ein σ' existiert, sodass $\langle S, \sigma\rangle \to \sigma'$ gilt.

? Analog *terminiert S nicht* bei Startzustand σ.

Das Programm S *terminiert immer*, falls S für jeden Startzustand σ terminiert. S *terminiert nie*, falls S für keinen Startzustand σ terminiert.

Definition 3.11. Seien S_1, S_2 zwei Programme. S_1 und S_2 heißen *semantisch äquivalent*, falls für alle Zustände σ, σ' gilt

$$\langle S_1, \sigma\rangle \to \sigma' \Leftrightarrow \langle S_2, \sigma\rangle \to \sigma'$$

Beispiel. Die Programme

$$S_1 = \texttt{while b do S}$$

und

$$S_2 = \texttt{if b then (S; while b do S) else skip}$$

sind semantisch äquivalent.

Beweis. Seien σ, σ' Zustände. Z. z.: $\langle S_1, \sigma\rangle \to \sigma' \Leftrightarrow \langle S_2, \sigma\rangle \to \sigma'$.

(a) "$\Rightarrow$"

Nimm an, es gilt $\langle S_1, \sigma\rangle \to \sigma'$. Also exisitert nach Definition ein endlicher Ableitungsbaum für $\langle S_1, \sigma\rangle \to \sigma'$.

(a) $\mathcal{B}[\![b]\!](\sigma) = \mathrm{w}$

Dann hat der Ableitungsbaum T_1 die Form

$$[\text{while}^{\mathrm{w}}_{\mathrm{ns}}]\ \frac{\dfrac{T_a}{\langle S, \sigma\rangle \to \sigma''},\ \dfrac{T_b}{\langle \texttt{while b do S}, \sigma''\rangle \to \sigma'}}{\langle \texttt{while b do S}, \sigma\rangle \to \sigma'}$$

Z. z.: es existiert ein endlicher Ableitungsbaum für $\langle S_2, \sigma\rangle \to \sigma'$

$$[\text{if}^{\mathrm{w}}_{\mathrm{ns}}]\ \frac{[\text{seq}_{\mathrm{ns}}]\ \dfrac{\dfrac{T_a}{\langle S, \sigma\rangle \to \sigma''},\ \dfrac{T_b}{\langle \texttt{while b do S}, \sigma''\rangle \to \sigma'}}{\langle \texttt{S; while b do S}, \sigma\rangle \to \sigma'}}{\langle \texttt{if b then (S; while b do S) else skip}, \sigma\rangle \to \sigma'}\quad \mathcal{B}[\![b]\!](\sigma) = \mathrm{w}$$

(b) $\mathcal{B}[\![b]\!](\sigma) = \mathrm{f}$

Dann hat T_1 die Form

$$[\text{while}^{\mathrm{f}}_{\mathrm{ns}}]\ \frac{}{\langle \texttt{while b do S}, \sigma\rangle \to \sigma}\quad \mathcal{B}[\![b]\!](\sigma) = \mathrm{f}$$

Dieses Axiom ist wahr, gdw. $\sigma = \sigma'$.

Z. z.: Es existiert ein Ableitungsbaum für $\langle S_2, \sigma\rangle \to \sigma$.

$$[\text{if}^{\text{f}}_{\text{ns}}]\ \frac{[\text{skip}_{\text{ns}}]\ \dfrac{}{\langle \texttt{skip}, \sigma\rangle \to \sigma}}{\langle \texttt{if b then (S; while b do S) else skip}, \sigma\rangle \to \sigma}\quad \mathcal{B}[\![b]\!](\sigma) = \text{f}$$

(b) "$\Leftarrow$"

Analog.

□

Lemma 3.12. *Die natürliche Semantik der* `while`*-Sprache ist* determiniert*, d. h. für alle Anweisungen S und Zustände $\sigma, \sigma_1, \sigma_2$ gilt:*

Wenn $\langle S, \sigma\rangle \to \sigma_1$ und $\langle S, \sigma\rangle \to \sigma_2$, dann $\sigma_1 = \sigma_2$.

Beweis. Durch strukturelle Induktion nach Tiefe des Ableitungsbaums. (Übung) □

Definition 3.13. Die semantische Funktion $\mathcal{S}_{\text{ns}}$: State $\to$ (State $\to$ State) ist definiert als

$$\mathcal{S}_{\text{ns}}[\![S]\!](\sigma) = \begin{cases} \sigma' & \text{falls } \exists\, \sigma' : \langle S, \sigma\rangle \to \sigma' \\ \perp & \text{sonst} \end{cases}$$

3.5 Strukturelle operationelle Semantik ("small step")

Hier geht es um die genaue Reihenfolge der Schritte bei der Ausführung. Das ist bespielsweise nützlich bei der parallelen Ausführungen eines Programms.

Wir definieren wieder eine Zustandsüberführungsrelation "⇒". Sie hat die Form

$$\langle S, \sigma \rangle \Rightarrow \langle S', \sigma' \rangle \qquad (*)$$

oder

$$\langle S, \sigma \rangle \Rightarrow \sigma' \qquad (**)$$

Interpretation:

(*) Ausführung ist noch nicht vorbei, sondern erreicht in *einem Schritt* die *Zwischenkonfiguration* $\langle s', \sigma' \rangle$.

(**) Ausführung ist nach einem Schritt vorbei und erreicht den Endzustand σ'.

Wir definieren ⇒ durch folgende Schlussregeln. Im Folgenden schreiben wir $[\cdot]_{\text{sos}}$ um anzuzeigen, dass es sich um die strukturelle operationelle Semantik handelt.

(a) $[\text{zuw}_{\text{sos}}]$

$$\langle x \texttt{ := } a, \sigma \rangle \Rightarrow \sigma[x \mapsto \mathcal{A}[\![a]\!](\sigma)]$$

(b) $[\text{skip}_{\text{sos}}]$

$$\langle \texttt{skip}, \sigma \rangle \Rightarrow \sigma$$

(c) $[\text{seq}^1_{\text{sos}}]$

$$\frac{\langle S_1, \sigma \rangle \Rightarrow \langle S_1', \sigma' \rangle}{\langle S_1 ; S_2, \sigma \rangle \Rightarrow \langle S_1' ; S_2, \sigma' \rangle}$$

(d) $[\text{seq}^2_{\text{sos}}]$

$$\frac{\langle S_1, \sigma \rangle \Rightarrow \sigma'}{\langle S_1 ; S_2, \sigma \rangle \Rightarrow \langle S_2, \sigma' \rangle}$$

(e) $[\text{if}^{\text{w}}_{\text{sos}}]$

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, \sigma \rangle \Rightarrow \langle S_1, \sigma \rangle$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{w}$

(f) $[\text{if}^{\text{f}}_{\text{sos}}]$

$$\langle \texttt{if } b \texttt{ then } S_1 \texttt{ else } S_2, \sigma \rangle \Rightarrow \langle S_2, \sigma \rangle$$

falls $\mathcal{B}[\![b]\!](\sigma) = \text{f}$

(g) $[\text{while}_{\text{sos}}]$

$$\langle \texttt{while } b \texttt{ do } S, \sigma \rangle \Rightarrow \langle \texttt{if } b \texttt{ then (S; while } b \texttt{ do } S\texttt{) else skip}, \sigma \rangle$$

Definition 3.14. Sei S eine Anweisung und σ ein Zustand.

Eine Ableitungsfolge für $\langle S, \sigma \rangle$ ist entweder

(a) eine endliche Folge $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \cdots \Rightarrow \gamma_k$ von Konfigurationen, sodass $\gamma_0 = \langle S, \sigma \rangle$ ist, $\gamma_i \Rightarrow \gamma_{i+1}$ für $0, \dots, k-1$ gilt und γ_k entweder ein Zustand σ' oder eine Konfiguration $\langle S', \sigma' \rangle$ ist, für die es mit $\Rightarrow$ nicht weiter geht *(steckengebliebe Konfiguration)*.

(b) eine unendliche Folge $\gamma_0 \Rightarrow \gamma_1 \Rightarrow \dots$ von Konfigurationen mit $\gamma_0 = \langle S, \sigma \rangle$ mit $\gamma_i \Rightarrow \gamma_{i+1}$ für $i \geq 0$.

Notation. Wir schreiben

$\gamma_0 \Rightarrow^i \gamma'$ für "γ' geht aus i Schritten hervor" und

$\gamma_0 \Rightarrow^* \gamma'$ für "γ' geht aus endlich vielen Schritten hervor" (auch null).

03.06.

Beispiel. Sei σ ein Zustand mit $\sigma(x) = 5, \sigma(y) = 7$. Betrachte die Auswertung von `(z := x; x := y); y = z;` in der SOS für Startzustand σ.

$$
\begin{aligned}
& \langle \texttt{(z := x; x := y); y = z;}, \sigma\rangle \\
\overset{[\mathrm{seq}^1_{\mathrm{sos}}]}{\Rightarrow}\ & \langle \texttt{x := y; y := z}, \sigma[z \mapsto 5]\rangle && \text{(i)} \\
\overset{[\mathrm{seq}^2_{\mathrm{sos}}]}{\Rightarrow}\ & \langle \texttt{y := z}, \sigma[z \mapsto 5][y \mapsto 7]\rangle && \text{(ii)} \\
\overset{[\mathrm{zuw}_{\mathrm{sos}}]}{\Rightarrow}\ & \sigma[z \mapsto 5][x \mapsto 7][y \mapsto 5]
\end{aligned}
$$

zu (i):

$$
[\mathrm{seq}^1_{\mathrm{sos}}]\ \dfrac{[\mathrm{seq}^2_{\mathrm{sos}}]\ \dfrac{[\mathrm{zuw}_{\mathrm{sos}}]\ \dfrac{}{\langle \texttt{z := x}, \sigma\rangle \Rightarrow \sigma[z \mapsto 5]}}{\langle \texttt{z := x; x := y}, \sigma\rangle \Rightarrow \langle \texttt{x := y}, \sigma[z \mapsto 5]\rangle}}{\langle \texttt{(z := x; x := y); y := z}, \sigma\rangle \Rightarrow \langle \texttt{x := y; y := z}, \sigma[z \mapsto 5]\rangle}
$$

zu (ii):

$$
[\mathrm{seq}^2_{\mathrm{sos}}]\ \dfrac{[\mathrm{zuw}_{\mathrm{sos}}]\ \dfrac{}{\langle x := y, \sigma[z \mapsto 5]\rangle \Rightarrow \langle y := z, \sigma[z \mapsto 5][x \mapsto 7]\rangle}}{\langle x := y; y := z, \sigma[z \mapsto 5]\rangle \Rightarrow \langle y := z, \sigma[z \mapsto 5][x \mapsto 7]\rangle}
$$

3.6 Eigenschaften der SOS

Lemma 3.15. *Seiten S_1, S_2 Anweisungen, σ, σ'' Zustände und $k \in \mathbb{N}$. Dann gilt: Falls $\langle S_1; S_2, \sigma\rangle \Rightarrow^k \sigma''$ gilt, es existieren zwei Zahlen $k_1, k_2 \in \mathbb{N}$ mit*

$$\langle S_1, \sigma\rangle \Rightarrow^{k_1} \sigma' \ , \quad \langle S_2, \sigma'\rangle \Rightarrow^{k_2} \sigma''$$

und

$$k_1 + k_2 = k$$

Beweis. Induktion nach k.

Induktionsanfang:

(a) $k = 1$: Voraussetzung kann dafür nicht erfüllt sein, also stimmt die Aussage.

(b) $k = 2$: Wie kann $\langle S_1; S_2, \sigma\rangle \Rightarrow^2 \sigma''$ gelten? Das kann nur sein, wenn im ersten Schritt $[\mathrm{seq}^2_{\mathrm{sos}}]$ augewendet wird. D. h. die Schlussregel

$$\frac{\langle S_1, \sigma\rangle \Rightarrow \sigma'}{\langle S_1; S_2, \sigma\rangle \Rightarrow \langle S_2, \sigma'\rangle}$$

wurde erfüllt für ein σ'.

Wir wissen also, es existiert ein Zwischenzustand σ' mit $\langle S_1, \sigma\rangle \Rightarrow \sigma'$ und erster Schritt von $\langle S_1; S_2, \sigma\rangle \Rightarrow^2 \sigma''$ ist $\langle S_1; S_2, \sigma\rangle \Rightarrow \langle S_2, \sigma'\rangle$ Der zweite Schritt $\langle S_1; S_2, \sigma\rangle \Rightarrow^2 \sigma''$ muss jetzt aber der Form $\langle S_2, \sigma'\rangle \Rightarrow \sigma''$ sein.

D. h. die Aussage gilt mit $k_1 = 1, k_2 = 1$ und σ'.

Induktionsschritt: $k - 1 \mapsto k$ mit $k \geq 3$

Betrachten den ersten Schritt $\langle S_1; S_2, \sigma\rangle \Rightarrow^k \sigma''$. Zwei Fälle

(a) [$\text{seq}^1_{\text{sos}}$] Der erste Schritt hat die Form $\langle S_1; S_2, \sigma\rangle \Rightarrow^k \langle S_1'; S_2, \sigma'''\rangle$

Dann muss aber gelten $\langle S_1'; S_2, \sigma'''\rangle \Rightarrow^{k-1} \sigma''$. Nach IV existiert $k_1', k_2' \in \mathbb{N}, \sigma'$, sodass $\langle S_1', \sigma'''\rangle \Rightarrow^{k_1'} \sigma'$ und $\langle S_2', \sigma'\rangle \Rightarrow^{k_2'} \sigma''$ und $k_1' + k_2' = k - 1$.

Da wir im ersten Schritt [$\text{seq}^1_{\text{sos}}$] angewandt haben, muss die Schlussregel dafür erfüllt gewesen sein, d. h. es gilt $\langle S_1, \sigma\rangle \Rightarrow \langle S_1', \sigma'''\rangle$. Also gilt auch $\langle S_1', \sigma\rangle \Rightarrow^{k_1'} \sigma'$ also $\langle S_1, \sigma\rangle \Rightarrow^{k_1+1} \sigma'$.

Also gilt die Aussage für $k_1 + 1, k_2 = k_2', \sigma'$.

(b) [$\text{seq}^2_{\text{sos}}$] Der erste Schritt hat die Form $\langle S_1; S_2, \sigma\rangle \Rightarrow \langle S_2, \sigma'\rangle$ und es gilt $\langle S_1, \sigma\rangle \Rightarrow \sigma'$.

Also gilt die Aussage für $k_1 = 1, k_2 = k - 1, \sigma'$.

□

Lemma 3.16 (Determinierheit). *SOS ist* determiniert. *Anders als bei der natürlichen Semantik müssen auch alle Zwischenzustände gleich sein, d. h.*

Für jedes S, σ existiert gibt es eine Ableitungsfolge, die mit $\langle S, \sigma\rangle$ beginnt.

Definition 3.17 (Semantische Äquivalenz). Seien S_1, S_2 zwei Anweisungen. S_1, S_2 heißen *semantische äquivalent* gdw. folgendes für allen Zustände σ gilt:

(a) Für alle steckengebliebenen Konfigurationen γ und alle Endzustände σ' gilt

$$\langle S_1, \sigma\rangle \Rightarrow^* \gamma \Leftrightarrow \langle S_2, \sigma\rangle \Rightarrow^* \gamma$$

und

$$\langle S_1, \sigma\rangle \Rightarrow^* \sigma' \Leftrightarrow \langle S_2, \sigma\rangle \Rightarrow^* \sigma'$$

(b) Es existiert eine undendliche Ableitungsfolge für $\langle S_1, \sigma\rangle$ gdw. es existiert eine unendliche Folge für $\langle S_2, \sigma\rangle$.

Beispiel. S_1; $(S_2;\ S_3)$ und $(S_1;\ S_2)$; S_3 sind semantische äquivalent.

3.7 Semantische Funktion $\mathcal{S}_{\text{sos}}$

Definition 3.18. Definiere $\mathcal{S}_{\text{sos}}$: SExp → (State → State) als

$$\mathcal{S}_{\text{sos}}[\![S]\!](\sigma) = \begin{cases} \sigma' & \text{falls} \langle S, \sigma\rangle \Rightarrow^* \sigma' \\ \perp & \text{sonst} \end{cases}$$

Diese Funktion ist wohldefiniert, da SOS determiniert ist.

Satz 3.2. *Sei S eine Anweisung und seien σ, σ' Zustände. Dann gilt*

$$\mathcal{S}_{ns}[\![S]\!](\sigma) = \sigma' \quad \Leftrightarrow \quad \mathcal{S}_{sos}[\![S]\!](\sigma) = \sigma'$$

D. h. SOS und NS sind äquivalent für unser konkretes Beispiel *der `while`-Sprache.*

Beweis. Zwei Richtungen:

(a) "$\Rightarrow$": $\mathcal{S}_{\text{ns}}[\![S]\!](\sigma) = \sigma' \Rightarrow \mathcal{S}_{\text{sos}}[\![S]\!](\sigma) = \sigma'$

d. h. $\langle S, \sigma\rangle \to \sigma' \implies \langle S, \sigma\rangle \Rightarrow^* \sigma'$

Wir wissen $\langle S, \sigma\rangle \to \sigma'$, d. h. es existiert ein endliche Ableitungsbaum T dafür. Mache Induktion nach der Tiefe von T.

Induktionsanfang: T hat Tiefe 0, d. h. T besteht nur aus einer Wurzel. Das bedeutet, $\langle S, \sigma\rangle \to \sigma'$ erfolgt durch Anwendung eines Axioms. Davon gibt es drei Stück: $[\text{zuw}_{\text{ns}}]$, $[\text{skip}_{\text{ns}}]$, $[\text{while}^{\text{f}}_{\text{ns}}]$

Exemplarisch für $[\text{while}^{\text{f}}_{\text{ns}}]$:

Wir wissen S hat die Form `while` b `do` S und $\mathcal{B}[\![b]\!](\sigma) = \text{f}$. D. h. T hat die Form

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

Jetzt gilt

$$\begin{aligned}
& \langle \texttt{while } b \texttt{ do } S', \sigma\rangle \\
\overset{[\text{while}_{\text{sos}}]}{\Rightarrow}\ & \langle \texttt{if } b \texttt{ then } (S'\texttt{; while } b \texttt{ do } S') \texttt{ else skip}, \sigma\rangle \\
\overset{[\text{if}^{\text{f}}_{\text{sos}}]}{\Rightarrow}\ & \langle \texttt{skip}, \sigma\rangle \qquad \text{da } \mathcal{B}[\![b]\!](\sigma) = \text{f} \\
\overset{[\text{skip}_{\text{sos}}]}{\Rightarrow}\ & \sigma
\end{aligned}$$

Also $\langle \texttt{while } b \texttt{ do } S', \sigma\rangle \Rightarrow^* \sigma$ wie gewünscht.

Induktionsschritt:

□