**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

# Hilfreiche Definitionen

## 1 Einführung

Dieses Dokument beinhaltet einige formale Definitionen und Spezifikationen für den Stoff der Vorlesung, für die Punkte, an denen die Operationen durch Bilder o.ä. definiert wurden. Diese sind nicht zwangsweise die einzige mögliche formalisierung des Stoffs. In der Klausur ist es nur wichtig, dass Sie formal korrekt argumentieren, Sie müssen nicht unbedingt diese formalen Definitionen benutzen. Dieses Dokument hat nicht das Ziel, diese Definitionen zu erklären; dazu dient die Vorlesung.

## 2 Union-Find

Gegeben eine Menge von Objekten $X$, ist der Zustand des Systems nach $t \in \mathbb{N}$ Operationen gegeben durch

- Menge von Mengen $S$
$$S_t \subseteq P(X)$$
- Parentfunktion
$$p_t : \bigcup S_t \to \bigcup S_t$$
- Rankfunktion
$$r_t : \bigcup S_t \to \mathbb{N}_0$$

Welches die folgenden Invarianten besitzt:

- Disjunktheit
$$R_1, R_2 \in S_t \ \wedge \ R_1 \cap R_2 \neq \emptyset \quad \to \quad R_1 = R_2$$
- Endliche Anzahl von Vorfahren bis man zu einem Zyklus der Größe 1 kommt
$$x \in \bigcup S_t \ \to \ \exists i. \ p_t^i(x) = p_t^{i+1}(x)$$

Die $t$-te operation sei
$$op_t$$
und der Ursprungszustand gegeben durch
$$(S_0, p_0, r_0) = (\emptyset, \emptyset, \emptyset)$$

Wir verwenden implizite $t$-Notation und primed notation. Mit anderen Worten, wir benutzen Abkürzungen
$$p(x) \text{ für } p_t(x)$$
und
$$p'(x) \text{ für } p_{t+1}(x)$$
und so weiter, vorrausgesetzt es ist eindeutig um welches $t$ es sich handelt.

Variablen $x, y, z$ haben den Wertebereich $X$. Aufgrund von disjunktheit können wir für $x \in \bigcup S$ die eindeutige Menge $S(x)$ finden, in der $x$ enthalten ist
$$S(x) = \epsilon \{ \ R \in S \mid x \in R \ \}$$

Die folgenden Operationen sind möglich.

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

## Hilfreiche Definitionen

- $op = makeset(x)$ wenn $x \notin \bigcup S$.
  Dann gilt
  $$S' = S \cup \{ \{ x \} \},\ p'(x) = x,\ r'(x) = 0$$
  und alles andere ist unverändert. Die Klausel "alles andere ist unverändert" ist eine Abkürzung dafür, dass sich alle Komponenten der Konfigurationen vor Schritt $t$ und nach Schritt $t+1$ nur auf die oben genannte Art unterscheiden. In diesem Fall gilt also
  $$a \neq x \quad \rightarrow \quad p'(a) = p(a)$$
  sowie
  $$a \neq x \quad \rightarrow \quad r'(a) = r(a)$$

- $op = find(x)$ wenn $x \in \bigcup S$.
  Sei $s$ die Anzahl von Eltern bis man einen Knoten erreicht, der sein eigenes Elter[1] ist
  $$s = \min \left\{ s > 0 \;\middle|\; p^s(x) = p^{s-1}(x) \right\}$$
  Wir definieren eine null-basierte Sequenz $(x_i)$ der Länge $s$ durch
  $$x_{s-1} = x,\ x_{i-1} = p(x_i)$$
  Dann gilt
  $$p'(x_i) = x_0$$
  und alles andere ist unverändert. Es gilt also *zum Beispiel*
  $$(\nexists i.\ a = x_i) \quad \rightarrow \quad r'(a) = r(a)$$
  Weiterhin gilt
  $$find(x) = x_0$$

- $op = link(x, y)$ wenn $x \neq y$, $x, y \in \bigcup S$, $p(x) = x$, $p(y) = y$.
  Dann gilt
  $$S' = (S \setminus \{ S(x), S(y) \}) \cup \{ S(x) \cup S(y) \}$$
  $$\begin{aligned} r(x) < r(y) &\rightarrow p'(x) = y \\ r(x) = r(y) &\rightarrow p'(x) = y \ \wedge\ r'(y) = r(y) + 1 \\ r(x) > r(y) &\rightarrow p'(y) = x \end{aligned}$$
  und alles andere ist unverändert. Es gelten also *zum Beispiel*
  $$r(x) < r(y) \ \wedge\ a \neq x \quad \rightarrow \quad p'(a) = p(a)$$
  und
  $$r(x) < r(y) \quad \rightarrow \quad r'(a) = r(a)$$
  und
  $$r(x) = r(y) \ \wedge\ a \neq y \quad \rightarrow \quad r'(a) = r(a)$$

[1] Ja, das Wort steht im Duden! `https://www.duden.de/rechtschreibung/Elter`

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

## Hilfreiche Definitionen

# 3 Programmiersprache

Für unsere Zwecke reicht eine einfache Programmiersprache ohne Heap, mit nur einem Basistyp (`int`), und ohne Funktionsaufrufen/Seiteneffekten in Audrücken. Typdeklarationen und die dadurch definierten Typen sind sonst wie in der Vorlesung eingeführt. Dann führen wir für jeden so definerten Typen $T$ noch einen Typen

$$V_T$$

ein, den wir für Variablen mit Typ $T$ verwenden (diese haben zusätzlich zu einem right-value, den alle Ausdrücke vom Typ $T$ haben, noch einen left-value). Wir überladen Typen und ihre Wertebereiche. Dann gilt

$$V_T \;=\; T\texttt{*} \times T$$

## 3.1 Ausdrücke und Ihre Typen

Folgende Ausdrücke sind erlaubt und haben folgenden Typ:

- Ganzzahlige Operationen $\circ \in \{\, \texttt{<}, \texttt{=}, \texttt{!=}, \texttt{||}, \texttt{\&\&}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/} \,\}$

  $$e, \hat{e} \text{ sind vom Typ } \texttt{int} \quad\to\quad e \circ \hat{e} \text{ ist vom Typ } \texttt{int}$$

- Zahlen

  $$x \in [0:9]^* \;\to\; x \text{ ist vom Typ } \texttt{int}$$

- Nullpointer

  $$\texttt{null} \text{ ist vom Typ } T\texttt{*}$$

- Vergleiche $\circ \in \{\, \texttt{=}, \texttt{!=} \,\}$ auf Pointertypen

  $$e, \hat{e} \text{ sind vom Typ } T\texttt{*} \quad\to\quad e \circ \hat{e} \text{ ist vom Typ } \texttt{int}$$

- Feldzugriff

  $$e \text{ ist vom Typ } V_{\{\ \ldots;\ t_i\ \ n_i;\ \ldots\}} \quad\to\quad e.n_i \text{ ist vom Typ } V_{t_i}$$

- Arrayzugriff

  $$e \text{ ist vom Typ } V_{T\texttt{[}n\texttt{]}} \;\wedge\; \hat{e} \text{ ist vom Typ } \texttt{int} \quad\to\quad e[\hat{e}] \text{ ist vom Typ } V_T$$

- Dereferenzierung

  $$e \text{ ist vom Typ } T\texttt{*} \quad\to\quad e\texttt{*} \text{ ist vom Typ } V_T$$

- Variablen

  $x$ ist eine in der Funktion deklarierte Variable/Parameter mit Typ $T$ $\quad\to\quad$ $x$ ist vom Typ $V_T$

- Right-Value für Pointer- und Basistypen $T = \texttt{int} \;\vee\; T = U\texttt{*}$

  $$e \text{ ist vom Typ } V_T \quad\to\quad e \text{ ist vom Typ } T$$

- Addresse (Left-Value)

  $$e \text{ ist vom Typ } V_T \quad\to\quad \texttt{\&}e \text{ ist vom Typ } T\texttt{*}$$

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

# Hilfreiche Definitionen

### 3.2 Statements

Die Programmiersprache hat die folgenden Statements:

- Zuweisung

  $$d \texttt{ = } e$$

  wobei es einen Typ $T$ geben muss sodass $d$ ein Ausdruck vom Typ $V_T$ ist und $e$ ein Ausdruck vom Typ $T$ ist.

- Schleife mit Substatement $S$

  $$\begin{array}{l}\texttt{while } e\\ \quad S\end{array}$$

  wobei $e$ ein Ausdruck vom Typ `int` ist

- Konditional mit Substatement $S$

  $$\begin{array}{l}\texttt{if } e\\ \quad S\end{array}$$

  wobei $e$ ein Ausdruck vom Typ `int` ist

- Funktionsaufrufe

  $$e \texttt{ = } f\texttt{(}e_1\texttt{, ..., }e_s\texttt{)}$$

  Wobei $f$ eine Funktion mit Signatur

  $$T\ f\texttt{(}T_1\texttt{, ..., }T_s\texttt{)}$$

  ist und $e$ ein Ausdruck vom Typ $V_T$ ist, und $e_i$ ein Ausdruck vom Typ $T_i$ ist.

- Rücksprung

  $$\texttt{return } e$$

  in einer Funktion mit Rückgabetyp $T$ wobei $e$ den Typ $T$ hat.

- Sequenz von $s \geq 1$ top-level Statements

  $$\begin{array}{l}\{\\ \quad S_1\\ \quad \vdots\\ \quad S_s\\ \}\end{array}$$

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

# Hilfreiche Definitionen

### 3.3 Zustände und Auswertung

Ein Zustand des Programms ist ein Tupel $c$ bestehend aus

- Programmzeiger
$$c.\texttt{pc} \in \mathbb{N}_0$$
der die aktuelle Zeile im Programmtext angibt.

- Stapel (stack)
$$c.\texttt{st} \in \bigcup_n ([0:n-1] \to S)$$
wobei $S$ die Menge aller Schachteln (stack frames) ist, die selbst wiederum Tupel $s$ mit folgenden Komponenten sind:

  - Funktion
  $$s.\texttt{f} \in FN$$
  - Rücksprungaddresse
  $$s.\texttt{ra} \in \mathbb{N}_0$$
  - Rückgabeaddresse
  $$s.\texttt{rd} \in T\texttt{*}$$
  wobei $s.\texttt{f}$ die folgende Signatur hat:
  $$T \;\; s.\texttt{f(...)}$$
  - Parameter+Variablenwerte
  $$s.\texttt{m} : \bigcup (\{\, n_i \,\} \to T_i)$$
  wobei $n_i$ der Name und $T_i$ der Typ des/der $i$-ten deklarierten Parameter/Variablen ist

Wir definieren nun einige Wertebereiche von Typen. Die Idee ist dass wir als arrays 0-basierte listen nehmen, die nichts anderes als Funktionen sind, die jedem index den Wert des Arrays an diesem Index zuweisen. Für structs nehmen wir Tupel, deren Komponenten genau die Komponenten des Structs sind. Pointer sind stets relativ zu einem Objekt zu verstehen und zeigen auf ein Teilobjekt, also zB auf das 5te feld der Komponente $x$:

$$o\texttt{.x[5]}$$

Wir stellen diese als Sequenzen von array-indizes und feldnamen dar, im obigen beispiel

$$(\texttt{x}, 5)$$

Also haben Objekte von Pointertypen die Form

$$T\texttt{*} = (\mathbb{N}_0 \cup FN)^* \cup \{\, \texttt{null} \,\}$$

wobei $FN$ die Menge aller möglichen Feldnamen von Structs ist. Die Objekte von Arraytypen $T\texttt{[}n\texttt{]}$ sind Funktionen die jedem Wert im Wertebreich $[0:n-1]$ ein Objekt des Typs $T$ zuweisen

$$T\texttt{[}n\texttt{]} \;\; = \;\; [0:n-1] \to T$$

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

## Hilfreiche Definitionen

Die Objekte von Structtypen

$$s \in \{\ \ldots;\ t_i\ n_i\ ;\ \ldots\}$$

sind Tupel mit Komponenten $n_i$ vom Typ $t_i$

$$s.n_i \in t_i$$

Deshalb ist jede Konfiguration $c$ selbst ein Objekt eines Structs, mit insbesondere Komponente

$$c.\mathtt{st} \in [0 : |c.\mathtt{st}| - 1] \to S$$

die selbst ein Objekt eines Arraytyps ist.

Das Objekt auf den ein Pointer $xs$ innerhalb eines Objekts $o$ Zeigt ist dann rekursiv wie folgt definiert:

$$m(o, \epsilon) = \epsilon$$
$$m(o, x \circ cs) = \begin{cases} m(o(x), xs) & x \in \mathbb{N}_0 \\ m(o.x, xs) & x \in VN \end{cases}$$

Man folgt also von links nach rechts den Feldnamen oder Array indices, und bekommt in vorherigem beispiel

$$m(o, (\mathtt{x}, 5)) = m(o.\mathtt{x}, (5)) = m(o.\mathtt{x}(5), \epsilon) = o.\mathtt{x}(5)$$

Da jede Konfiguration selbst auch ein Objekt ist können wir insbesondere auch folgendes machen:

$$m(c, (\mathtt{st}, i, \mathtt{m}, x)) = c.\mathtt{st}(i).\mathtt{m}.x$$

und dadurch den Wert der/des Variablen/Parameters $x$ auf dem $i$-ten Stackframe abfragen.

In einem Zustand $c$ sind der Left-Value und Right-Value des Ausdrucks $e$, denotiert durch

$$[e]'_c, [e]_c$$

rekursiv definiert:

$$[e \circ \hat{e}]_c = [e]_c \circ [\hat{e}]_c$$

(vorrausgesetzt es liegt noch im Wertebereich), wobei für Wahrheitswerte 1 (wahr) bzw 0 (falsch) verwendet werden

$$[z]_c = bin(z)$$

(vorrausgesetzt die Zahl $z$ liegt noch im Wertebereich)

$$[\mathtt{null}]_c = \mathtt{null}$$
$$[e.n]'_c = [e]'_c \circ n$$
$$[e.n]_c = e[c].n$$
$$[e[\hat{e}]]'_c = [e]'_c \circ [\hat{e}]_c$$
$$[e[\hat{e}]]_c = [e]_c([\hat{e}]_c)$$

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

# Hilfreiche Definitionen

(vorrausgesetzt $[\hat{e}]_c$ ist im entsprechenden Wertebereich)

$$\begin{aligned}
[e\texttt{*}]'_c &= [e]_c \\
[e\texttt{*}]_c &= m(c, [e]_c) \\
[x]'_c &= (\texttt{st}, |c.\texttt{st}| - 1, \texttt{m}, x) \\
[x]_c &= m(c, [x]'_c) \\
[\texttt{\&}e]_c &= [e]'_c
\end{aligned}$$

Wir definieren auch einen Default-wert für Variablen rekursiv. Beim Default-wert werden alle Integerfelder mit 0 und alle Pointerfelder mit `null` aufgefüllt

$$\begin{aligned}
default(\texttt{int}) &= 0 \\
default(T\texttt{*}) &= \texttt{null} \\
default(\{\ \ldots;\ t_i\ n_i\ ;\ \ldots\}).n_i &= default(t_i) \\
default(T\texttt{[}n\texttt{]})(i) &= default(T)
\end{aligned}$$

### 3.4 Ausführung

Die Programausführung betrachtet nun das Programm als Sequenz $(P_i)$ von Statements, wobei das Statement an Position $i$ im Programmtext in Zeile $i$ steht. Wir definieren von aussen nach innen einen Syntaktischen Nachfolger $N(i)$ jeder Zeile $i$.

- für eine in Zeile $z$ stehende Sequenz von $s$ top-level Statements $S_1, \ldots, S_s$ von denen $S_i$ in Zeile $s_i$ steht

  $$\begin{array}{c|l} z & \{ \\ z_1 & \quad S_1 \\ \vdots & \quad \vdots \\ z_s & \quad S_s \\ & \} \end{array}$$

  Definieren wir als Nachfolger von Zeile $z_i$ für $1 \leq i < s$

  $$N(z_i) = z_{i+1}$$

  und für $z_s$

  $$N(z_s) = N(z)$$

- für ein Konditional in Zeile $z$ mit Substatement in Zeile $\hat{z}$

  $$\begin{array}{c|l} z & \texttt{if}\ e \\ \hat{z} & \quad S \end{array}$$

  Definieren wir

  $$N(\hat{z}) = N(z)$$

- für eine Schleife in Zeile $z$ mit Substatement in Zeile $\hat{z}$

## Hilfreiche Definitionen

$$\begin{array}{l|l} z & \texttt{while } e \\ \hat{z} & \quad S \end{array}$$

Definieren wir

$$N(\hat{z}) = z$$

(also wird nach Ausführung des Schleifenkörpers nocheinmal die Schleife begonnen)

Sei nun $c$ eine Konfiguration. Die nächste Konfiguration $c'$ ist definiert durch Fallunterscheidung des Statements $P_{c.\texttt{pc}}$.

- Zuweisung

  $$d \texttt{ = } e$$

  Dann gilt

  $$\begin{aligned} c'.\texttt{pc} &= N(c.\texttt{pc}) \\ m(c', [d]'_c) &= [e]_c \end{aligned}$$

  Sonst alles unverändert

- Schleife oder Konditional mit Substatement $S$

  $$\begin{array}{l|l} z & \texttt{while/if } e \\ z_s & \quad S \end{array}$$

  Es gilt

  $$c'.\texttt{pc} = \begin{cases} z_s & [e]_c \neq 0 \\ N(z) & \text{sonst} \end{cases}$$

  Sonst alles unverändert

- Funktionsaufrufe

  $$e \texttt{ = } f\texttt{(}e_1\texttt{, ..., }e_s\texttt{)}$$

  wobei $f$ Paramter+Variablen $n_i$ von Typ $T_i$ hat. Wir setzen eine neue Schachtel auf, die zur Funktion $f$ gehört, hinter das Aktuelle Statement zurückspringt, den Wert in die Variable $e$ zurückschreibt, und Parametern+Variablen einen unten definierten Wert $v$ gibt

  $$c'.\texttt{st}(|c.\texttt{st}|) = (f, N(c.\texttt{pc}), [e]'_c, v)$$

  wobei $v$ Parametern den Wert des Audrucks zuweist und Variablen den Default-wert. Sei also $n_i$ Name und $T_i$ Typ des/der $i$-ten Parameters/Variablen, dann gilt

  $$v.n_i = \begin{cases} [e_i]_c & i \leq s \\ default(T_i) & \text{sonst} \end{cases}$$

  Sei die Funktionsdeklaration mit Substatement $S$ in Zeile $z_s$ also wie folgt

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

## Hilfreiche Definitionen

$$\begin{array}{c|l} & T\ f(t_1\ n_1,\ \ldots,\ t_s\ n_s)\ \{ \\ & \quad t_{s+1}\ n_{s+1} \\ & \quad \vdots \\ & \quad t_k\ n_k \\ z_s & \quad S \end{array}$$

dann Springt die Ausführung nun zu Zeile $z_s$

$$c'.\texttt{pc} = z_s$$

Ansonsten bleibt die Ausführung unverändert.

- Rücksprung

  $$\texttt{return}\ e$$

  Ist dies die letzte Schachtel auf dem Stapel, so wird die Programmausführung beendet und das Programm gibt den Wert $e$ aus. Sonst wird der Rückgabevariablen der Wert von $e$ zugewiesen

  $$m(c', c.\texttt{st}(|c.\texttt{st} - 1|).\texttt{rd}) = [e]_c$$

  die alte Schachtel wird verlassen

  $$|c'.\texttt{st}| = |c.\texttt{st}| - 1$$

  und es wird zur Rücksprungaddresse zurückgesprungen

  $$c'.\texttt{pc} = c.\texttt{st}(|c.\texttt{st} - 1|).\texttt{ra}$$

  Sonst ändert sich nichts.

- Sequenz von $s \geq 1$ top-level Statements, von denen das erste in Zeile $z_1$ beginnt

  $$\begin{array}{c|l} & \{ \\ z_1 & \quad S_1 \\ \vdots & \quad \vdots \\ z_s & \quad S_s \\ & \} \end{array}$$

  Dann wird zu Zeile $z_1$ gesprungen

  $$c'.\texttt{pc} = z_1$$

Achtung: in vielen Fällen ist obiges keine Definition; zB wenn es kein Statement $P_{c.\texttt{pc}}$ gibt. Dann ist die nächste Konfiguration undefiniert. Um zu Zeigen dass ihr Programm gewisse Eigenschaften hat, müssen Sie stets Zeigen, dass der nächste Zustand stets definiert ist, zumindest bis zum letzten `return` (wenn der Stapel leer ist).

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**

# Hilfreiche Definitionen

### 3.5 Bigstep Semantik

Angenommen Sie haben für eine Teilmenge $F$ von Funktionen $f \in F$ in ihrem Programm eine Relation zwischen Stapeln

$$st \to_f st'$$

sowie eine Vorbedingung auf die Argumente

$$P_f(st, v_1, \ldots, v_s)$$

und eine Funktion die den Rückgabewert angibt

$$f(st, v_1, \ldots, v_s) = r$$

Dann können Sie die Semantik eines Funktionsaufrufs

$$e \texttt{ = } f\texttt{(}e_1\texttt{, ..., }e_s\texttt{)}$$

wie folgt ändern: Angenommen dass die Vorbedingung in der Konfiguration $c$ mit dem Stack und den Werten der Argumente gilt

$$P_f(c.\texttt{st}, [e_1]_c, \ldots, [e_s]_c)$$

dann gibt es einen Stack $st'$ der in Relation zu $c.\texttt{st}$ steht

$$c.\texttt{st} \to_f st'$$

und für die nächste Konfiguration $c'$ gilt

$$\begin{aligned} c'.\texttt{pc} &= N(c.\texttt{pc}) \\ c'.\texttt{st}(i).(\texttt{rd},\texttt{ra}) &= st'(i).(\texttt{rd},\texttt{ra}) \\ c'.\texttt{st}(i).\texttt{m}(x) &= \begin{cases} f(c.\texttt{st}, [e_1]_c, \ldots, [e_s]_c) & (\texttt{st}, i, \texttt{m}) \circ x = [e]'_c \\ st'(i).\texttt{m}(x) & \text{sonst} \end{cases} \end{aligned}$$

vorrausgesetzt Sie können Zeigen, dass

- in der normalen (small-step) Semantik jede Funktion $f \in F$ Terminiert und
- in der Bigstep Semantik jede Funktion $f \in F$ vor dem Return-zustand einen Zustand erreicht, dessen Stack in relation zum Stack der Ursprünglichen funktion steht, und dessen Rückgabewert anhand der Vorschrift $f(\ldots)$ berechnet wurde. Formal: Habe die Funktion $f$ die Form

$$\begin{array}{r|l} & T\ \ f\texttt{(}t_1\ \ n_1\texttt{, ..., }t_s\ \ n_s\texttt{) \{} \\ & \quad t_{s+1}\ \ n_{s+1} \\ & \quad \vdots \\ & \quad t_k\ \ n_k \\ z_s & \quad S \end{array}$$

## Hilfreiche Definitionen

und sei $c = c^0$ eine beliebige Konfiguration mit Stapelhöhe $h = |c.\texttt{st}|$ die in Zeile $z_s$

$$c.\texttt{pc} = z_s$$

mit default initialiserten variablen beginnt

$$i > s \quad \rightarrow \quad [n_i]_c = default(t_i)$$

und ausserdem die Vorbedinging erfüllt

$$Q_f(c.\texttt{st}[0:h-2], [n_1]_c, \ldots, [n_s]_c)$$

Sei nun $c^t$ die Konfiguration nach $t$ Schritten. Wenn gilt

$$P_{c^t.\texttt{pc}} = \texttt{return } e$$

und die Höhe des Stapels sich nicht verändert hat

$$|c^t.\texttt{st}| = |c.\texttt{st}|$$

und auch nie unter diese Höhe gesunken ist

$$\forall i < t.\ |c^i.\texttt{st}| \geq |c.\texttt{st}|$$

dann muss folgendes Gelten:

$$c.\texttt{st}[0:h-2] \rightarrow_f c^t.\texttt{st}[0:h-2]$$

und

$$[e]_{c^t} = f(c.\texttt{st}[0:h-2], [n_1]_c, \ldots, [n_s]_c)$$

Das heisst, wenn Sie gezeigt haben dass die Funktionen $f \in F$ die Semantik haben die durch $\rightarrow_f$ und $f(\ldots)$ beschrieben wird, können Sie die Funktionsaufrufe von $f$ ersetzen durch einen einzelnen Schritt, der diese Semantik anwendet.

**Saarland University**
**Department 6.2 - Computer Science**
**Prof. Dr. W. J. Paul**
**Dr. Jonas Oberhauser**



# 4 Listen

Für Mengen $A$ ist $A^n$ die Menge der Listen der Länge $n$, welche Funktionen

$$a[1:n] : [1:n] \to A$$

sind. Dabei wird die Länge auch denotiert durch

$$|a| = n$$

Konkatenation von Listen $a[1:n]$ und $b[1:m]$ gibt eine Liste

$$(a \circ b)[1:n+m]$$

die definiert ist durch

$$(a \circ b)[x] = \begin{cases} a[x] & x \leq |a| \\ b[x - |a|] & \text{sonst} \end{cases}$$

Sublisten $a[i:j]$ sind entweder die leere Liste falls $j < i$

$$j < i \;\to\; a[i:j] = \epsilon$$

oder eine Liste der Länge $j - i + 1$

$$(a[i:j])[1:j-i+1]$$

die wie folgt definiert ist:

$$(a[i:j])[k] = a[k+i-1]$$