Karlsruher Institut für Technologie

Lehrstuhl für Programmierparadigmen

Denis Lohner denis.lohner@kit.edu

Sebastian Ullrich sebastian.ullrich@kit.edu

Semantik von Programmiersprachen – SS 2017

http://pp.ipd.kit.edu/lehre/SS2017/semantik

Lösungen zu Blatt 8: Typsicherheit Besprechung: 19.06.2017

1. Welche der folgenden Aussagen sind richtig, welche falsch? (H)

(a) $[\mathtt{x} \mapsto \mathsf{int},\ \mathtt{y} \mapsto \mathsf{bool}] \vdash \texttt{(x + 3 <= 17) || (not y)} :: \mathsf{bool}$

(b) Im Kontext $[\mathtt{x} \mapsto \mathsf{bool}]$ ist `true == x` ist ein typkorrekter Ausdruck.

(c) $\mathcal{E}\,[\![\texttt{0 * (x - y)}]\!]\,[\mathtt{x} \mapsto 5,\ \mathtt{y} \mapsto \mathbf{tt}] = 0$

(d) Jeder Ausdruck e hat in jedem Kontext Γ höchstens einen Typ τ.

(e) Der Typ eines Wertes v ließe sich auch so definieren:

$$\text{type}(v) = \tau \quad \text{gdw.} \quad \Gamma \vdash v :: \tau$$

(f) Es gibt keinen Kontext Γ mit

$$\Gamma \vdash \texttt{\{ var x = x; z := x \}; \{ var y = x; if (y) then z := 5 else skip \}} \surd$$

(g) Wenn $\Gamma \vdash c \surd$ und $\langle c,\ \sigma\rangle \to_1 \langle c',\ \sigma'\rangle$ mit $\sigma' :: \Gamma$ und $\Gamma \vdash c' \surd$, dann auch $\sigma :: \Gamma$.

(h) Wenn $\Gamma \vdash c \surd$ und $\sigma :: \Gamma$, dann gibt es c' und σ' mit $\langle c,\ \sigma\rangle \to_1 \langle c',\ \sigma'\rangle$.

Lösung:

(1a) Das hängt vom Syntaxbaum des Ausdrucks ab. Durch die geänderte Grammatik gibt es zwei mögliche Syntaxbäume:

`((x + 3) <= 17) || (not y)` und `(x + (3 <= 17)) || (not y)`

Im linken Fall gilt die Aussage, obwohl für `+` und `||` *keine* Typregeln definiert wurden – diese sind nur syntaktischer Zucker und damit automatisch abgedeckt. Ohne syntaktischen Zucker schreibt sich dieser Ausdruck wie folgt:

```
not ((not ((x - (0 - 3)) <= 17)) && (not (not y)))
```

Der Ableitungsbaum im Typsystem müsste für diesen Ausdruck aufgestellt werden.

Im rechten Fall gilt die Typaussage nicht, weil int `+` bool nicht typisierbar ist.

(1b) Falsch. `true == x` ist zwar ein Ausdruck, aber nicht typkorrekt. e_1 `==` e_2 ist syntaktischer Zucker für `(`e_1 `<=` e_2`) && (`e_2 `<=` e_1`)`. Nach den Typregeln für `<=` müssen e_1 und e_2 den Typ int besitzen.

Wollte man Gleichheit auch für boolesche Ausdrücke, gibt es zwei Möglichkeiten:

i. Explizite Produktion für `==` in Exp. Dann bräuchte man dafür auch eine eigene Typregel

$$\frac{\Gamma \vdash e_1 :: \tau \qquad \Gamma \vdash e_2 :: \tau}{\Gamma \vdash e_1 \texttt{ == } e_2 :: \mathsf{bool}}$$

und entsprechend auch eigene Semantikregeln, Beweisfälle etc.

ii. Man füht zwei Ersetzungsmöglichkeiten für e_1 `==` e_2 ein:

$(e_1$ `<=` $e_2)$ `&&` $(e_2$ `<=` $e_1)$ und $(e_1$ `&&` $e_2)$ `||` `((not` $e_1)$ `&&` `(not` $e_2))$

Dann ist es aber unmöglich, mit `==` einen eindeutigen Syntaxbaum zu beschreiben. Erst ein Typcheck könnte „unsinnige“ Syntaxbäume eliminieren.

(1c) Falsch. Wegen $\mathcal{E}[\![\texttt{x - y}]\!][\texttt{x} \mapsto 5,\ \texttt{y} \mapsto \mathbf{tt}] = \bot$ gilt $\mathcal{E}[\![\texttt{0 * (x - y)}]\!][\texttt{x} \mapsto 5,\ \texttt{y} \mapsto \mathbf{tt}] = \bot$.
`0` ist für `*` (wie `false` für `&&`) *nicht* mehr ein absorbierendes Element.

(1d) Richtig. Formal: Wenn $\Gamma \vdash e :: \tau$ und $\Gamma \vdash e :: \tau'$, dann $\tau = \tau'$.
Beweis. Normal bräuchte man Regel-Induktion über $\Gamma \vdash e :: \tau$ (τ' beliebig). Für unser einfaches Typsystem reicht aber Fallunterscheidung:

- Fälle TNum, TMinus, TTimes:
  Zu zeigen: Wenn $\Gamma \vdash e :: \tau'$ (wobei $e = n$, $e = e_1$ `-` e_2 oder $e = e_1$ `*` e_2), dann $\mathsf{int} = \tau'$.
  Beweis durch Regelinversion.
- Fälle TTrue, TLeq, TAnd, TNot: Analog mit bool.
- Fall TVar: Zu zeigen: Wenn $\Gamma \vdash x :: \tau'$ und $\Gamma(x) = \tau$, dann $\tau = \tau'$.
  Beweis: Durch Regelinversion gilt $\Gamma(x) = \tau'$. □

Typsysteme, die jeder syntaktischen Einheit höchstens einen Typ zu weisen, heißen *monomorph*.

(1e) Falsch. Wie in (1d) gezeigt, ist der Typ eines Ausdrucks zwar eindeutig, aber hier wird wieder Syntax mit abstrakten Werten vermischt. v ist aus $\mathbb{Z} + \mathbb{B}$, kein Ausdruck. Richtig wäre z.B. $\text{type}(v) = \tau$ gdw. $\Gamma \vdash \mathcal{V}^{-1}[\![v]\!] :: \tau$.

(1f) Richtig. Aus den Typregeln ergeben sich folgende widersprüchliche Bedingungen:

- Im vorderen Block muss die lokale Variable `x` wegen der Initialisierung den gleichen Typ wie die globale Variable `x` haben. Wegen `z := x` muss `z` den gleichen Typ haben wie die lokale Varialbe `x`. Insgesamt: $\Gamma(\texttt{z}) = \Gamma(\texttt{x})$
- Da die lokale Variable `y` des zweiten Blocks als Bedingung auftaucht, muss sie den Typ bool haben. Wegen der Initialisierung mit der globalen Variable `x` gilt also: $\Gamma(\texttt{x}) = \mathsf{bool}$.
- Wegen `z := 5` muss gelten $\Gamma(\texttt{z}) = \mathsf{int}$.

(1g) Falsch. Dies ist die Umkehrung zur Erhaltung der Zustandskonformanz (Lem. 77). Gegenbeispiel:
$[\texttt{x} \mapsto \mathsf{int}] \vdash x$ `:= 0` $\surd$, $[\texttt{x} \mapsto \mathsf{int}] \vdash$ `skip` $\surd$, $[\texttt{x} \mapsto 0] :: [\texttt{x} \mapsto \mathsf{int}]$
und $\langle x$ `:= 0`, $[\texttt{x} \mapsto \mathbf{tt}]\rangle \to_1 \langle$`skip`, $[\texttt{x} \mapsto 0]\rangle$, aber nicht $[\texttt{x} \mapsto \mathbf{tt}] :: [\texttt{x} \mapsto \mathsf{int}]$.

(1h) Falsch. Dies ist nur *fast* das Fortschrittslemma 76. Für den Fall $c =$ `skip` gilt es nicht.

2. Ausgaben (H)

Erweitern Sie die Sprache While_T um Ausgaben:

(a) Ergänzen Sie die Syntax um eine neue Anweisung `print` e für Ausgaben.

(b) Passen Sie die Typ-Regeln an. Sowohl Zahlen als auch Wahrheitswerte sind als Ausgabe erlaubt.

(c) Ändern Sie die Big-Step-Semantik, sodass die neue Auswertungsrelation $\langle c,\ \sigma\rangle \Downarrow_o \sigma'$ die Anweisung c im Anfangszustand σ zum Endzustand σ' auswertet, wobei die Liste der Ausgaben o entsteht.

(d) Passen Sie auch die Small-Step-Semantik an Ausgaben an. Unterscheidet sie sich in der Ausdrucksmächtigkeit von der Big-Step-Semantik?

(e) Ist die erweiterte Sprache typsicher? Begründen Sie!

Lösung:

(2a) Man braucht nur die Anweisungssyntax zu erweitern:

$$\mathsf{Com}\quad c \quad ::= \quad \texttt{skip} \mid x \texttt{ := } e \mid c_1\texttt{;}\ c_2 \mid \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2 \mid \texttt{while (}e\texttt{) do } c \mid \texttt{\{ var } x \texttt{ = } e\texttt{;}\ c\ \texttt{\}} \mid \texttt{print } e$$

(2b) Neue Typregel:

$$\textsc{TPrint}\colon \frac{\Gamma \vdash e :: \tau}{\Gamma \vdash \texttt{print } e\ \surd}$$

(2c) Man muss jede Regel anpassen – auch erst einmal für While_T:

$$\textsc{Skip}^{\mathrm{O}}_{\mathrm{BS}}\colon \langle \texttt{skip},\ \sigma\rangle \Downarrow_{[]} \sigma \qquad \textsc{Ass}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v}{\langle x \texttt{ := } e,\ \sigma\rangle \Downarrow_{[]} \sigma[x \mapsto v]}$$

$$\textsc{Seq}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\langle c_1,\ \sigma\rangle \Downarrow_{o_1} \sigma' \qquad \langle c_2,\ \sigma'\rangle \Downarrow_{o_2} \sigma''}{\langle c_1\texttt{;}\ c_2,\ \sigma\rangle \Downarrow_{o_1 +\!+ o_2} \sigma''}$$

$$\textsc{IfTT}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt} \qquad \langle c_1,\ \sigma\rangle \Downarrow_o \sigma'}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \Downarrow_o \sigma'}$$

$$\textsc{IfFF}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{ff} \qquad \langle c_2,\ \sigma\rangle \Downarrow_o \sigma'}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \Downarrow_o \sigma'}$$

$$\textsc{WhileFF}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{ff}}{\langle \texttt{while (}e\texttt{) do } c,\ \sigma\rangle \Downarrow_{[]} \sigma}$$

$$\textsc{WhileTT}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt} \qquad \langle c,\ \sigma\rangle \Downarrow_{o_1} \sigma' \qquad \langle \texttt{while (}e\texttt{) do } c,\ \sigma'\rangle \Downarrow_{o_2} \sigma''}{\langle \texttt{while (}e\texttt{) do } c,\ \sigma\rangle \Downarrow_{o_1 +\!+ o_2} \sigma''}$$

$$\textsc{Block}^{\mathrm{O}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v \qquad \langle c,\ \sigma[x \mapsto v]\rangle \Downarrow_o \sigma'}{\langle \texttt{\{ var } x \texttt{ = } e\texttt{;}\ c\ \texttt{\}},\ \sigma\rangle \Downarrow_o \sigma'[x \mapsto \sigma(x)]}$$

Die neue Regel für `print` e:

$$\textsc{Print}^{\mathrm{O}}_{\mathrm{BS}}\colon\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v}{\langle \texttt{print } e,\ \sigma\rangle \Downarrow_{[v]} \sigma}$$

(2d) Auch die Small-Step-Relation $\langle c,\ \sigma\rangle \xrightarrow{o} \langle c',\ \sigma'\rangle$ muss komplett angepasst werden. Über dem Pfeil steht jetzt die Ausgabe des aktuellen Schritts (als maximal einelementige Liste).

$$\textsc{Ass}^{\mathrm{O}}_{\mathrm{SS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v}{\langle x \texttt{ := } e,\ \sigma\rangle \xrightarrow{[]} \langle \texttt{skip},\ \sigma[x \mapsto v]\rangle} \qquad \textsc{Print}^{\mathrm{O}}_{\mathrm{SS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v}{\langle \texttt{print } e,\ \sigma\rangle \xrightarrow{[v]} \langle \texttt{skip},\ \sigma\rangle}$$

$$\textsc{Seq1}^{\mathrm{O}}_{\mathrm{SS}}\colon \frac{\langle c_1,\ \sigma\rangle \xrightarrow{o} \langle c_1',\ \sigma'\rangle}{\langle c_1\texttt{;}\ c_2,\ \sigma\rangle \xrightarrow{o} \langle c_1'\texttt{;}\ c_2,\ \sigma'\rangle} \qquad \textsc{IfTT}^{\mathrm{O}}_{\mathrm{SS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt}}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \xrightarrow{[]} \langle c_1,\ \sigma\rangle}$$

$$\textsc{Seq2}^{\mathrm{O}}_{\mathrm{SS}}\colon \langle \texttt{skip;}\ c,\ \sigma\rangle \xrightarrow{[]} \langle c,\ \sigma\rangle \qquad \textsc{IfFF}^{\mathrm{O}}_{\mathrm{SS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{ff}}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \xrightarrow{[]} \langle c_2,\ \sigma\rangle}$$

$$\textsc{While}^{\mathrm{O}}_{\mathrm{SS}}\colon \langle \texttt{while (}e\texttt{) do } c,\ \sigma\rangle \xrightarrow{[]} \langle \texttt{if (}e\texttt{) then } c\texttt{; while (}e\texttt{) do } c \texttt{ else skip},\ \sigma\rangle$$

$$\textsc{Block1}^{\mathrm{O}}_{SS}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v \qquad \langle c,\ \sigma[x \mapsto v]\rangle \xrightarrow{o} \langle c',\ \sigma'\rangle}{\langle \texttt{\{ var } x \texttt{ = } e\texttt{;}\ c\ \texttt{\}},\ \sigma\rangle \xrightarrow{o} \langle \texttt{\{ var } x \texttt{ = } \mathcal{V}^{-1}\,[\![\sigma'(x)]\!]\texttt{;}\ c'\ \texttt{\}},\ \sigma'[x \mapsto \sigma(x)]\rangle}$$

$$\textsc{Block2}^{\text{O}}_{SS}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma \neq \bot}{\langle \texttt{\{ var } x \texttt{ = } e\texttt{; skip \}},\ \sigma\rangle \xrightarrow{[]} \langle \texttt{skip},\ \sigma\rangle}$$

Jetzt muss man auch die Semantik durch Ableitungsfolgen anders definieren. Eine endliche Ableitungsfolge $\langle c_0,\ \sigma_0\rangle \overset{o_0}{\to} \langle c_1,\ \sigma_1\rangle \overset{o_1}{\to} \ldots \overset{o_{n-1}}{\to} \langle c_n,\ \sigma_n\rangle$ fassen wir nun mit $\langle c,\ \sigma\rangle \overset{o}{\to}{}^{*} \langle c',\ \sigma'\rangle$ zusammen, wobei $o = o_0 +\!\!+ o_1 +\!\!+ \ldots o_{n-1}$. Entsprechend verkürzen wir eine unendliche Ableitungsfolge $\langle c_0,\ \sigma_0\rangle \overset{o_0}{\to} \langle c_1,\ \sigma_1\rangle \overset{o_1}{\to} \ldots$ auf $\langle c_0,\ \sigma_0\rangle \overset{o}{\to}{}^{\infty}$, wobei $o = o_0 +\!\!+ o_1 +\!\!+ \ldots$.

Big- und Small-Step-Semantik sind jetzt nicht mehr gleich ausdrucksstark: Die Ausgaben nichtterminierender Programme sind in der Small-Step-Semantik ausdrückbar; in der Big-Step-Semantik gibt es nur Nichtableitbarkeit, aber keine Unterscheidung der generierten Ausgaben.

(2e) Ja, die neue Sprache ist typsicher. Der Typsicherheitsbeweis mit Fortschrittslemma, Erhaltung der Zustandskonformanz und Subject Reduction lässt sich leicht um die neuen Fälle für `print` e ergänzen. Die zusätzlichen Ausgaben in allen Regeln werden von Typsystem und Konformanz ignoriert und führen auch keine neuen Bedingungen in den Small-Step-Regeln durch die Hintertür ein.

3. Typsicherheit mit der Big-Step-Semantik (Ü)

Typsicherheit lässt sich auch für eine Big-Step-Semantik zeigen, in dieser Aufgabe für While_T:

(a) Ändern Sie die Auswertungsrelation so, dass eine Anweisung mit einem Zustand statt zu einem Endzustand auch zu einem speziellen Fehlerwert Err auswerten kann. Fügen Sie nun für jede Regel, die einen Typcheck (implizit) durchführt, eine neue Regel für den Fehlerfall hinzu. Vergessen Sie auch nicht Propagationsregeln für den Typfehler Err.

(b) Zeigen Sie, dass typkorrekte Programme von konformanten Zuständen aus nie Typfehler erzeugen: Wenn $\Gamma \vdash c\ \surd$, $\sigma :: \Gamma$ und $\langle c,\ \sigma\rangle \Downarrow \bar{\sigma}'$, dann $\bar{\sigma}' \neq \mathsf{Err}$ und $\bar{\sigma}' :: \Gamma$.

(c) Vergleichen Sie diesen Ansatz, Typsicherheit einer Sprache zu zeigen, mit dem Small-Step-Ansatz aus der Vorlesung.

Lösung:

(3a) Unterscheidung in korrekte Auswertung σ und Typfehler Err:

$$\langle \_,\ \_\rangle \Downarrow \_ \subseteq \mathsf{Com} \times \Sigma \times (\Sigma + \{\mathsf{Err}\})$$

Normale Regeln der Big-Step-Semantik mit der Variablenkonvention $\bar{\sigma} \in (\Sigma + \{\mathsf{Err}\})$:

$$\textsc{Skip}^{\text{T}}_{\text{BS}}\colon \langle \texttt{skip},\ \sigma\rangle \Downarrow \sigma \qquad \textsc{Ass}^{\text{T}}_{\text{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v}{\langle x\ \texttt{:=}\ e,\ \sigma\rangle \Downarrow (\sigma[x \mapsto v])}$$

$$\textsc{Seq}^{\text{T}}_{\text{BS}}\colon \frac{\langle c_1,\ \sigma\rangle \Downarrow \sigma' \qquad \langle c_2,\ \sigma'\rangle \Downarrow \bar{\sigma}''}{\langle c_1\texttt{;}\ c_2,\ \sigma\rangle \Downarrow \bar{\sigma}''}$$

$$\textsc{IfTT}^{\text{T}}_{\text{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt} \qquad \langle c_1,\ \sigma\rangle \Downarrow \bar{\sigma}'}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \Downarrow \bar{\sigma}'}$$

$$\textsc{IfFF}^{\text{T}}_{\text{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{ff} \qquad \langle c_2,\ \sigma\rangle \Downarrow \bar{\sigma}'}{\langle \texttt{if (}e\texttt{) then } c_1 \texttt{ else } c_2,\ \sigma\rangle \Downarrow \bar{\sigma}'}$$

$$\textsc{WhileFF}^{\text{T}}_{\text{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{ff}}{\langle \texttt{while (}e\texttt{) do } c,\ \sigma\rangle \Downarrow \sigma}$$

$$\textsc{WhileTT}^{\mathrm{T}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt} \qquad \langle c,\ \sigma\rangle \Downarrow \sigma' \qquad \langle \texttt{while (}e\texttt{) do }c,\ \sigma'\rangle \Downarrow \bar{\sigma}''}{\langle \texttt{while (}e\texttt{) do }c,\ \sigma\rangle \Downarrow \bar{\sigma}''}$$

$$\textsc{Block}^{\mathrm{T}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v \qquad \langle c,\ \sigma[x \mapsto v]\rangle \Downarrow \sigma'}{\langle \texttt{\{ var }x\texttt{ = }e\texttt{; }c\texttt{ \}},\ \sigma\rangle \Downarrow (\sigma'[x \mapsto \sigma(x)])}$$

Anmerkung zu den Änderungen: Ergebnisse müssen immer die Form σ oder Err haben. Wenn das Resultat einer Regel weiterverarbeitet werden soll (in weiteren Prämissen wie $\textsc{Seq}^{\mathrm{T}}_{\mathrm{BS}}$ und $\textsc{WhileTT}^{\mathrm{T}}_{\mathrm{BS}}$ oder durch Änderung des Zustands wie in $\textsc{Block}^{\mathrm{T}}_{\mathrm{BS}}$), dann darf dies nur bei normaler Auswertung σ möglich sein.

Und jetzt noch die ganzen Regeln für Typfehler (keine vergessen!):

$$\textsc{Ass}^{\textsc{Err}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \bot}{\langle x\ \texttt{:=}\ e,\ \sigma\rangle \Downarrow \mathsf{Err}} \qquad \textsc{Seq}^{\textsc{Err}}_{\mathrm{BS}}\colon \frac{\langle c_1,\ \sigma\rangle \Downarrow \mathsf{Err}}{\langle c_1\texttt{;}\ c_2,\ \sigma\rangle \Downarrow \mathsf{Err}}$$

$$\textsc{If}^{\textsc{Err}}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma \notin \mathbb{B}}{\langle \texttt{if (}e\texttt{) then }c_1\texttt{ else }c_2,\ \sigma\rangle \Downarrow \mathsf{Err}}$$

$$\textsc{While}^{\textsc{Err}1}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma \notin \mathbb{B}}{\langle \texttt{while (}e\texttt{) do }c,\ \sigma\rangle \Downarrow \mathsf{Err}} \qquad \textsc{While}^{\textsc{Err}2}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \mathbf{tt} \qquad \langle c,\ \sigma\rangle \Downarrow \mathsf{Err}}{\langle \texttt{while (}e\texttt{) do }c,\ \sigma\rangle \Downarrow \mathsf{Err}}$$

$$\textsc{Block}^{\textsc{Err}1}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = \bot}{\langle \texttt{\{ var }x\texttt{ = }e\texttt{; }c\texttt{ \}},\ \sigma\rangle \Downarrow \mathsf{Err}} \qquad \textsc{Block}^{\textsc{Err}2}_{\mathrm{BS}}\colon \frac{\mathcal{E}\,[\![e]\!]\,\sigma = v \quad \langle c,\ \sigma[x \mapsto v]\rangle \Downarrow \mathsf{Err}}{\langle \texttt{\{ var }x\texttt{ = }e\texttt{; }c\texttt{ \}},\ \sigma\rangle \Downarrow \mathsf{Err}}$$

(3b) Diese Aussage beinhaltet im Wesentlichen den Beweis zur Zustandskonformanzerhaltung bei Small-Step (Lem. 77).

Zu zeigen: Wenn $\langle c,\ \sigma\rangle \Downarrow \bar{\sigma}'$ mit $\Gamma \vdash c\ \surd$ und $\sigma :: \Gamma$, dann ist σ' von der Form σ'' mit $\sigma'' :: \Gamma$.

Beweis. Regel-Induktion über $\langle c,\ \sigma\rangle \Downarrow \bar{\sigma}'$ (Γ beliebig). Wegen der nichtterminierenden Regel $\textsc{WhileTT}^{\mathrm{T}}_{\mathrm{BS}}$ ist dies die einzige Beweismöglichkeit.

- Fälle $\textsc{Skip}^{\mathrm{T}}_{\mathrm{BS}}$, $\textsc{WhileFF}^{\mathrm{T}}_{\mathrm{BS}}$: Trivial.
- Fall $\textsc{Ass}^{\mathrm{T}}_{\mathrm{BS}}$: Beweis analog zur Zustandskonformanzerhaltung bei Small-Step (Lem. 77).
- Fall $\textsc{Ass}^{\textsc{Err}}_{\mathrm{BS}}$: Aus $\Gamma \vdash x\ \texttt{:=}\ e\ \surd$ erhält man durch Regelinversion (TAss) ein τ mit $\Gamma(x) = \tau$ und $\Gamma \vdash e :: \tau$. Zusammen mit $\sigma :: \Gamma$ gilt nach Lem. 75: $\mathcal{E}\,[\![e]\!]\,\sigma \neq \bot$. Widerspruch zur Fallannahme $\mathcal{E}\,[\![e]\!]\,\sigma = \bot$.
- Fall $\textsc{Seq}^{\mathrm{T}}_{\mathrm{BS}}$: Induktionsannahmen:
  (i) Wenn $\Gamma \vdash c_1\ \surd$ und $\sigma :: \Gamma$, dann ist σ' von der Form... naja, ist ja schon von der Form. Wir erhalten allerdings noch $\sigma' :: \Gamma$.
  (ii) Wenn $\Gamma \vdash c_2\ \surd$ und $\sigma' :: \Gamma$, dann ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' :: \Gamma$.
  Zu zeigen: Wenn $\Gamma \vdash c_1\texttt{;}\ c_2\ \surd$ und $\sigma :: \Gamma$, dann ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' :: \Gamma$.
  Aus $\Gamma \vdash c_1\texttt{;}\ c_2\ \surd$ erhält man $\Gamma \vdash c_1\ \surd$ und $\Gamma \vdash c_2\ \surd$ durch Regelinversion (TSeq).
  Mit $\sigma :: \Gamma$ folgt nach Induktionsannahme (i), dass $\sigma' :: \Gamma$ (hier benötigt man den Erhalt der Zustandskonformanz). Aus Induktionsannahme (ii) folgt dann direkt die Behauptung.
- Fall $\textsc{Seq}^{\textsc{Err}}_{\mathrm{BS}}$: Induktionsannahme: Wenn $\Gamma \vdash c_1\ \surd$ und $\sigma :: \Gamma$, dann ist Err von der Form σ' mit $\sigma' :: \Gamma$.
  Zu zeigen: Wenn $\Gamma \vdash c_1\texttt{;}\ c_2\ \surd$ und $\sigma :: \Gamma$, dann ist Err von der Form σ' mit $\sigma' :: \Gamma$.
  Regelinversion auf $\Gamma \vdash c_1\texttt{;}\ c_2\ \surd$ liefert $\Gamma \vdash c_1\ \surd$. Damit kann man die die Induktionsannahme anwenden, die eine absurde Aussage liefert, und eine absurde Aussage ist so gut wie jede andere.
- Fall $\textsc{WhileTT}^{\mathrm{T}}_{\mathrm{BS}}$: Induktionsannahmen:
  (i) Wenn $\Gamma \vdash c\ \surd$ und $\sigma :: \Gamma$, dann ist σ' von eben der Form und $\sigma' :: \Gamma$.
  (ii) Wenn $\Gamma \vdash \texttt{while (}e\texttt{) do }c\ \surd$ und $\sigma' :: \Gamma$, ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' :: \Gamma$.
  Zu zeigen: Wenn $\Gamma \vdash \texttt{while (}e\texttt{) do }c\ \surd$ und $\sigma :: \Gamma$, ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' :: \Gamma$.

Aus $\Gamma \vdash \texttt{while (}e\texttt{) do } c \surd$ folgt $\Gamma \vdash c \surd$. Mit Induktionsannahme (i) erhält man wieder $\sigma' :: \Gamma$. Induktionsannahme (ii) liefert die Behauptung.
Dieser Fall ist ähnlich zu $\textsc{Seq}^{\mathrm{T}}_{\mathrm{BS}}$, braucht aber die Induktion über $\langle c,\ \sigma\rangle \Downarrow \bar{\sigma}'$ für die Induktionsannahme (ii), deren Anweisung nicht kleiner ist als die Konklusion.

- Fall $\textsc{While}^{\textsc{Err}1}_{\mathrm{BS}}$: Aus Lem. 75 folgt mit $\Gamma \vdash e :: \mathbb{B}$ (Regelinversion TWhile) und $\sigma :: \Gamma$, dass $\mathcal{E}\,[\![e]\!]\,\sigma = v$ mit $v \in \mathbb{B}$. Widerspruch zur Fallannahme $\mathcal{E}\,[\![e]\!]\,\sigma \notin \mathbb{B}$.
- Fälle $\textsc{While}^{\textsc{Err}2}_{\mathrm{BS}}$, $\textsc{Block}^{\textsc{Err}2}_{\mathrm{BS}}$: Analog zu $\textsc{Seq}^{\textsc{Err}}_{\mathrm{BS}}$.
- Fälle $\textsc{IfTT}^{\mathrm{T}}_{\mathrm{BS}}$, $\textsc{IfFF}^{\mathrm{T}}_{\mathrm{BS}}$, $\textsc{If}^{\textsc{Err}}_{\mathrm{BS}}$: Analog zu `while`.
- Fall $\textsc{Block}^{\mathrm{T}}_{\mathrm{BS}}$: Analog zur Erhaltung der Zustandskonformanz in Lem. 77. Induktionsannahme braucht Γ beliebig.
- Fall $\textsc{Block}^{\textsc{Err}1}_{\mathrm{BS}}$: Analog zu $\textsc{Ass}^{\textsc{Err}}_{\mathrm{BS}}$. □

(3c) Typfehler werden in der Small-Step-Semantik durch Blockieren modelliert. Bei Big-Step ist dies nicht möglich, da Nichtableitbarkeit auch durch Nichttermination entstehen kann.

Vorteile Big-Step:

- Intuitiverer Ansatz, da Typfehler explizit modelliert werden.
- Beweis insgesamt einfacher, da kein Fortschritts- oder Subject-Reduction-Lemma gezeigt werden muss.

Nachteile Big-Step:

- Massive Veränderung an der Semantik nötig, i.d.R. doppelt so viele Regeln (und damit auch Beweis-Fälle in *jedem* Beweis über die Big-Step-Semantik.
- Typfehlerregeln sind semantisch irrelevant, weil für typkorrekte Programme nie anwendbar. Deswegen gehören sie auch nicht in die Semantik.
- Keine Kontrolle, ob wirklich alle Typfehlerregeln eingefügt wurden.