

Semantik von Programmiersprachen – SS 2019

<http://pp.ipd.kit.edu/lehre/SS2019/semantik>

Lösungen zu Blatt 8: Typsicherheit

Besprechung: 17.06.2019

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

- (a) $[x \mapsto \text{int}, y \mapsto \text{bool}] \vdash (x + 3 \leq 17) \ || \ (\text{not } y) : \text{bool}$
- (b) Im Kontext $[x \mapsto \text{bool}]$ ist $\text{true} == x$ ein typkorrekter Ausdruck.
- (c) $\mathcal{E} \llbracket 0 * (x - y) \rrbracket [x \mapsto 5, y \mapsto \text{tt}] = 0$
- (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

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

- (g) Wenn $\Gamma \vdash c \checkmark$ und $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$ mit $\sigma' : \Gamma$ und $\Gamma \vdash c' \checkmark$, dann auch $\sigma : \Gamma$.
- (h) Wenn $\Gamma \vdash c \checkmark$ und $\sigma : \Gamma$, dann gibt es c' und σ' mit $\langle c, \sigma \rangle \rightarrow_1 \langle c', \sigma' \rangle$.

Lösung:

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

$$((x + 3) \leq 17) \ || \ (\text{not } y) \quad \text{und} \quad (x + (3 \leq 17)) \ || \ (\text{not } y)$$

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:

$$\text{not } ((\text{not } ((x - (0 - 3)) \leq 17)) \ \&\& \ (\text{not } (\text{not } y)))$$

Der Ableitungsbaum im Typsystem müsste für diesen Ausdruck aufgestellt werden.

Im rechten Fall gilt die Typaussage nicht, weil $\text{int} + \text{bool}$ nicht typisierbar ist.

- (1b) Falsch. $\text{true} == x$ ist zwar ein Ausdruck, aber nicht typkorrekt. $e_1 == e_2$ ist syntaktischer Zucker für $(e_1 \leq e_2) \ \&\& \ (e_2 \leq e_1)$. Nach den Typregeln für \leq 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

$$\frac{\Gamma \vdash e_1 : \tau \quad \Gamma \vdash e_2 : \tau}{\Gamma \vdash e_1 == e_2 : \text{bool}}$$

und entsprechend auch eigene Semantikregeln, Beweisfälle etc.

ii. Man führt zwei Ersetzungsmöglichkeiten für $e_1 == e_2$ ein:

$$(e_1 \leq e_2) \ \&\& \ (e_2 \leq e_1) \quad \text{und} \quad (e_1 \ \&\& \ e_2) \ || \ ((\text{not } e_1) \ \&\& \ (\text{not } e_2))$$

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

(1c) Falsch. Wegen $\mathcal{E} \llbracket x - y \rrbracket [x \mapsto 5, y \mapsto \mathbf{tt}] = \perp$ gilt $\mathcal{E} \llbracket 0 * (x - y) \rrbracket [x \mapsto 5, y \mapsto \mathbf{tt}] = \perp$. 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 $\text{int} = \tau'$.

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 zuweisen, 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. $\Gamma \vdash \mathcal{V}^{-1} \llbracket v \rrbracket : \tau$.

(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 $\mathbf{z} := x$ muss \mathbf{z} den gleichen Typ haben wie die lokale Variable x . Insgesamt: $\Gamma(\mathbf{z}) = \Gamma(x)$
- 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: $\Gamma(x) = \mathbf{bool}$.
- Wegen $\mathbf{z} := 5$ muss gelten $\Gamma(\mathbf{z}) = \mathbf{int}$.

(1g) Falsch. Dies ist die Umkehrung zur Erhaltung der Zustandskonformanz (Lem. 77). Gegenbeispiel:

Sei $\Gamma = [x \mapsto \mathbf{int}]$. Dann gilt $\Gamma \vdash x := 0 \checkmark$, $\langle x := 0, [x \mapsto \mathbf{tt}] \rangle \rightarrow_1 \langle \mathbf{skip}, [x \mapsto 0] \rangle$ mit $[x \mapsto 0] : \Gamma$ und $\Gamma \vdash \mathbf{skip} \checkmark$, aber nicht $[x \mapsto \mathbf{tt}] : \Gamma$.

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

2. Ausgaben (H)

Erweitern Sie die Sprache While_T um Ausgaben:

- Ergänzen Sie die Syntax um eine neue Anweisung **print** e für Ausgaben.
- Passen Sie die Typ-Regeln an. Sowohl Zahlen als auch Wahrheitswerte sind als Ausgabe erlaubt.
- 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.
- Passen Sie auch die Small-Step-Semantik an Ausgaben an. Unterscheidet sie sich in der Ausdrucksmächtigkeit von der Big-Step-Semantik?
- Ist die erweiterte Sprache typsicher? Begründen Sie!

Lösung:

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

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

(2b) Neue Typregel:

$$\text{TPRINT: } \frac{\Gamma \vdash e : \tau}{\Gamma \vdash \text{print } e \checkmark}$$

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

$$\text{SKIP}_{\text{BS}}^{\text{O}}: \langle \text{skip}, \sigma \rangle \Downarrow_{\square} \sigma \quad \text{ASS}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v}{\langle x := e, \sigma \rangle \Downarrow_{\square} \sigma[x \mapsto v]}$$

$$\text{SEQ}_{\text{BS}}^{\text{O}}: \frac{\langle c_1, \sigma \rangle \Downarrow_{o_1} \sigma' \quad \langle c_2, \sigma' \rangle \Downarrow_{o_2} \sigma''}{\langle c_1; c_2, \sigma \rangle \Downarrow_{o_1 ++ o_2} \sigma''}$$

$$\text{IFTT}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt} \quad \langle c_1, \sigma \rangle \Downarrow_o \sigma'}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Downarrow_o \sigma'}$$

$$\text{IFFF}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{ff} \quad \langle c_2, \sigma \rangle \Downarrow_o \sigma'}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Downarrow_o \sigma'}$$

$$\text{WHILEFF}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{ff}}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow_{\square} \sigma}$$

$$\text{WHILETT}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt} \quad \langle c, \sigma \rangle \Downarrow_{o_1} \sigma' \quad \langle \text{while } (e) \text{ do } c, \sigma' \rangle \Downarrow_{o_2} \sigma''}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow_{o_1 ++ o_2} \sigma''}$$

$$\text{BLOCK}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v \quad \langle c, \sigma[x \mapsto v] \rangle \Downarrow_o \sigma'}{\langle \{ \text{var } x = e; c \}, \sigma \rangle \Downarrow_o \sigma'[x \mapsto \sigma(x)]}$$

Die neue Regel für $\text{print } e$:

$$\text{PRINT}_{\text{BS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v}{\langle \text{print } e, \sigma \rangle \Downarrow_{[v]} \sigma}$$

(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).

$$\text{ASS}_{\text{SS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v}{\langle x := e, \sigma \rangle \xrightarrow{\square} \langle \text{skip}, \sigma[x \mapsto v] \rangle} \quad \text{PRINT}_{\text{SS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v}{\langle \text{print } e, \sigma \rangle \xrightarrow{[v]} \langle \text{skip}, \sigma \rangle}$$

$$\text{SEQ1}_{\text{SS}}^{\text{O}}: \frac{\langle c_1, \sigma \rangle \xrightarrow{o} \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \xrightarrow{o} \langle c'_1; c_2, \sigma' \rangle} \quad \text{IFTT}_{\text{SS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt}}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \xrightarrow{\square} \langle c_1, \sigma \rangle}$$

$$\text{SEQ2}_{\text{SS}}^{\text{O}}: \langle \text{skip}; c, \sigma \rangle \xrightarrow{\square} \langle c, \sigma \rangle \quad \text{IFFF}_{\text{SS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{ff}}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \xrightarrow{\square} \langle c_2, \sigma \rangle}$$

$$\text{WHILE}_{\text{SS}}^{\text{O}}: \langle \text{while } (e) \text{ do } c, \sigma \rangle \xrightarrow{\square} \langle \text{if } (e) \text{ then } c; \text{while } (e) \text{ do } c \text{ else skip}, \sigma \rangle$$

$$\text{BLOCK1}_{\text{SS}}^{\text{O}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v \quad \langle c, \sigma[x \mapsto v] \rangle \xrightarrow{o} \langle c', \sigma' \rangle}{\langle \{ \text{var } x = e; c \}, \sigma \rangle \xrightarrow{o} \langle \{ \text{var } x = \mathcal{V}^{-1} \llbracket \sigma'(x) \rrbracket; c' \}, \sigma'[x \mapsto \sigma(x)] \rangle}$$

$$\text{BLOCK2}_{SS}^O: \frac{\mathcal{E} \llbracket e \rrbracket \sigma \neq \perp}{\langle \{ \text{var } x = e; \text{skip} \}, \sigma \rangle \Downarrow \langle \text{skip}, \sigma \rangle}$$

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

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 `WhileT`:

- 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`.
- Zeigen Sie, dass typkorrekte Programme von konformanten Zuständen aus nie Typfehler erzeugen: Wenn $\Gamma \vdash c \checkmark$, $\sigma : \Gamma$ und $\langle c, \sigma \rangle \Downarrow \bar{\sigma}'$, dann $\bar{\sigma}' \neq \text{Err}$ und $\bar{\sigma}' : \Gamma$.
- 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`:

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

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

$$\text{SKIP}_{BS}^T: \langle \text{skip}, \sigma \rangle \Downarrow \sigma \quad \text{ASS}_{BS}^T: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v}{\langle x := e, \sigma \rangle \Downarrow (\sigma[x \mapsto v])}$$

$$\text{SEQ}_{BS}^T: \frac{\langle c_1, \sigma \rangle \Downarrow \sigma' \quad \langle c_2, \sigma' \rangle \Downarrow \bar{\sigma}''}{\langle c_1; c_2, \sigma \rangle \Downarrow \bar{\sigma}''}$$

$$\text{IFTT}_{BS}^T: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt} \quad \langle c_1, \sigma \rangle \Downarrow \bar{\sigma}'}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Downarrow \bar{\sigma}'}$$

$$\text{IFFF}_{BS}^T: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{ff} \quad \langle c_2, \sigma \rangle \Downarrow \bar{\sigma}'}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Downarrow \bar{\sigma}'}$$

$$\text{WHILEFF}_{BS}^T: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{ff}}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow \sigma}$$

$$\text{WHILETT}_{\text{BS}}^{\text{T}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt} \quad \langle c, \sigma \rangle \Downarrow \sigma' \quad \langle \text{while } (e) \text{ do } c, \sigma' \rangle \Downarrow \bar{\sigma}''}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow \bar{\sigma}''}$$

$$\text{BLOCK}_{\text{BS}}^{\text{T}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v \quad \langle c, \sigma[x \mapsto v] \rangle \Downarrow \sigma'}{\langle \{ \text{var } x = e; c \}, \sigma \rangle \Downarrow (\sigma'[x \mapsto \sigma(x)])}$$

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 $\text{SEQ}_{\text{BS}}^{\text{T}}$ und $\text{WHILETT}_{\text{BS}}^{\text{T}}$ oder durch Änderung des Zustands wie in $\text{BLOCK}_{\text{BS}}^{\text{T}}$), dann darf dies nur bei normaler Auswertung σ möglich sein.

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

$$\text{ASS}_{\text{BS}}^{\text{ERR}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \perp}{\langle x := e, \sigma \rangle \Downarrow \text{Err}} \quad \text{SEQ}_{\text{BS}}^{\text{ERR}}: \frac{\langle c_1, \sigma \rangle \Downarrow \text{Err}}{\langle c_1; c_2, \sigma \rangle \Downarrow \text{Err}}$$

$$\text{IF}_{\text{BS}}^{\text{ERR}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma \notin \mathbb{B}}{\langle \text{if } (e) \text{ then } c_1 \text{ else } c_2, \sigma \rangle \Downarrow \text{Err}}$$

$$\text{WHILE}_{\text{BS}}^{\text{ERR1}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma \notin \mathbb{B}}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow \text{Err}} \quad \text{WHILE}_{\text{BS}}^{\text{ERR2}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \text{tt} \quad \langle c, \sigma \rangle \Downarrow \text{Err}}{\langle \text{while } (e) \text{ do } c, \sigma \rangle \Downarrow \text{Err}}$$

$$\text{BLOCK}_{\text{BS}}^{\text{ERR1}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = \perp}{\langle \{ \text{var } x = e; c \}, \sigma \rangle \Downarrow \text{Err}} \quad \text{BLOCK}_{\text{BS}}^{\text{ERR2}}: \frac{\mathcal{E} \llbracket e \rrbracket \sigma = v \quad \langle c, \sigma[x \mapsto v] \rangle \Downarrow \text{Err}}{\langle \{ \text{var } x = e; c \}, \sigma \rangle \Downarrow \text{Err}}$$

(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 $\Gamma \vdash c \checkmark$ 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 $\text{WHILETT}_{\text{BS}}^{\text{T}}$ ist dies die einzige Beweismöglichkeit.

- Fälle $\text{SKIP}_{\text{BS}}^{\text{T}}$, $\text{WHILEFF}_{\text{BS}}^{\text{T}}$: Trivial.
- Fall $\text{ASS}_{\text{BS}}^{\text{T}}$: Beweis analog zur Zustandskonformanzerhaltung bei Small-Step (Lem. 77).
- Fall $\text{ASS}_{\text{BS}}^{\text{ERR}}$: Aus $\Gamma \vdash x := e \checkmark$ erhält man durch Regelinversion (TASS) ein τ mit $\Gamma(x) = \tau$ und $\Gamma \vdash e : \tau$. Zusammen mit $\sigma : \Gamma$ gilt nach Lem. 75: $\mathcal{E} \llbracket e \rrbracket \sigma \neq \perp$. Widerspruch zur Fallannahme $\mathcal{E} \llbracket e \rrbracket \sigma = \perp$.
- Fall $\text{SEQ}_{\text{BS}}^{\text{T}}$: Induktionsannahmen:
 - (i) Wenn $\Gamma \vdash c_1 \checkmark$ und $\sigma : \Gamma$, dann ist σ' von der Form... naja, ist ja schon von der Form. Wir erhalten allerdings noch $\sigma' : \Gamma$.
 - (ii) Wenn $\Gamma \vdash c_2 \checkmark$ und $\sigma' : \Gamma$, dann ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' : \Gamma$.
 Zu zeigen: Wenn $\Gamma \vdash c_1; c_2 \checkmark$ und $\sigma : \Gamma$, dann ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' : \Gamma$.
 Aus $\Gamma \vdash c_1; c_2 \checkmark$ erhält man $\Gamma \vdash c_1 \checkmark$ und $\Gamma \vdash c_2 \checkmark$ 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 $\text{SEQ}_{\text{BS}}^{\text{ERR}}$: Induktionsannahme: Wenn $\Gamma \vdash c_1 \checkmark$ und $\sigma : \Gamma$, dann ist Err von der Form σ' mit $\sigma' : \Gamma$.
 Zu zeigen: Wenn $\Gamma \vdash c_1; c_2 \checkmark$ und $\sigma : \Gamma$, dann ist Err von der Form σ' mit $\sigma' : \Gamma$.
 Regelinversion auf $\Gamma \vdash c_1; c_2 \checkmark$ liefert $\Gamma \vdash c_1 \checkmark$. Damit kann man die Induktionsannahme anwenden, die eine absurde Aussage liefert, und eine absurde Aussage ist so gut wie jede andere.
- Fall $\text{WHILETT}_{\text{BS}}^{\text{T}}$: Induktionsannahmen:
 - (i) Wenn $\Gamma \vdash c \checkmark$ und $\sigma : \Gamma$, dann ist σ' von eben der Form und $\sigma' : \Gamma$.
 - (ii) Wenn $\Gamma \vdash \text{while } (e) \text{ do } c \checkmark$ und $\sigma' : \Gamma$, ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' : \Gamma$.
 Zu zeigen: Wenn $\Gamma \vdash \text{while } (e) \text{ do } c \checkmark$ und $\sigma : \Gamma$, ist $\bar{\sigma}''$ von der Form σ''' mit $\sigma''' : \Gamma$.

Aus $\Gamma \vdash \text{while } (e) \text{ do } c \checkmark$ folgt $\Gamma \vdash c \checkmark$. Mit Induktionsannahme (i) erhält man wieder $\sigma' : \Gamma$. Induktionsannahme (ii) liefert die Behauptung.

Dieser Fall ist ähnlich zu $\text{SEQ}_{\text{BS}}^{\text{T}}$, 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 $\text{WHILE}_{\text{BS}}^{\text{ERR1}}$: Aus Lem. 75 folgt mit $\Gamma \vdash e : \mathbb{B}$ (Regelinversion TWHILE) und $\sigma : \Gamma$, dass $\mathcal{E} \llbracket e \rrbracket \sigma = v$ mit $v \in \mathbb{B}$. Widerspruch zur Fallannahme $\mathcal{E} \llbracket e \rrbracket \sigma \notin \mathbb{B}$.
- Fälle $\text{WHILE}_{\text{BS}}^{\text{ERR2}}$, $\text{BLOCK}_{\text{BS}}^{\text{ERR2}}$: Analog zu $\text{SEQ}_{\text{BS}}^{\text{ERR}}$.
- Fälle $\text{IFTT}_{\text{BS}}^{\text{T}}$, $\text{IFF}_{\text{BS}}^{\text{T}}$, $\text{IF}_{\text{BS}}^{\text{ERR}}$: Analog zu **while**.
- Fall $\text{BLOCK}_{\text{BS}}^{\text{T}}$: Analog zur Erhaltung der Zustandskonformanz in Lem. 77. Induktionsannahme braucht Γ beliebig.
- Fall $\text{BLOCK}_{\text{BS}}^{\text{ERR1}}$: Analog zu $\text{Ass}_{\text{BS}}^{\text{ERR}}$. □

(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.