6.5 Continuation-style denotationale Semantik

In der Übung haben wir bereits eine Erweiterung While_X von While um Ausnahmen und deren Behandlung mit den Anweisungen

$$\texttt{raise} \quad \text{und} \quad \texttt{try } c_1 \texttt{ catch } c_2$$

betrachtet. Bei der Big-Step-Semantik haben wir ein zusätzliches Rückgabeflag eingeführt, um normale und außergewöhnliche Termination zu unterscheiden. Entsprechend mussten auch alle bestehenden Regeln sorgfältig angepasst und die Möglichkeit für eine Ausnahme in jedem Schritt eigens behandelt werden. In der Small-Step-Semantik musste dazu eine eigene `raise`-Regel für alle zusammengesetzten Konstrukte (bei uns nur $c_1\texttt{;}\ c_2$) eingeführt werden.

Ganz analog zur Big-Step-Semantik ließe sich auch die denotationale Semantik für While um Exceptions erweitern. Allerdings ist dieser Formalismus insgesamt nicht zufrieden stellend, da Ausnahmen nun einmal die Ausnahme sein und deswegen nicht explizit durch jedes Programmkonstrukt durchgeschleift werden sollten. Dafür gibt es Fortsetzungen (continuations), die die Semantik (d.h. den Effekt) der Ausführung des restlichen Programms beschreiben.

Definition 42 (Fortsetzung, continuation). Eine *Fortsetzung* (*continuation*) ist eine partielle Funktion auf Zuständen, die das Ergebnis der Ausführung des restlichen Programms von einem Zustand aus beschreibt. $\mathsf{Cont} = \Sigma \rightharpoonup \Sigma$ bezeichne die Menge aller Fortsetzungen.

Statt nun anhand eines Flags einem umgebenden Programmkonstrukts die Auswahl der restlichen Berechnung zu überlassen, soll jedes Konstrukt direkt auswählen, ob es normal oder mit der Ausnahmebehandlung weitergehen soll. Dazu muss die Semantik der restlichen normalen bzw. außergewöhnlichen Auswertung direkt bei der Definition eines Konstrukts als Fortsetzung zur Verfügung stehen. Zum Beispiel wählt `raise` die Fortsetzung „Ausnahmebehandlung“ und `skip` die Fortsetzung „normale Ausführung“ aus. Die beiden möglichen Fortsetzungen müssen also als Parameter an die Semantikfunktion $\mathcal{C}[\![\_]\!]$ gegeben werden, die damit den Typ

$$\mathsf{Com} \Rightarrow \underbrace{\mathsf{Cont}}_{\text{normale Fortsetzung}} \Rightarrow \underbrace{\mathsf{Cont}}_{\text{Ausnahmebehandlung}} \Rightarrow \mathsf{Cont}$$

hat. Intiutiv bedeutet $\mathcal{C}[\![c]\!]\ s\ t\ \sigma$ also: Führe c im Zustand σ aus und setze mit s normal bzw. mit t bei einer Ausnahme fort. Formal:

$$\begin{aligned}
\mathcal{C}[\![\texttt{skip}]\!]\ s\ t &= s \\
\mathcal{C}[\![x\ \texttt{:=}\ a]\!]\ s\ t &= \lambda\sigma.\ s(\sigma[x \mapsto \mathcal{A}[\![a]\!]\,\sigma]) \\
\mathcal{C}[\![c_1\texttt{;}\ c_2]\!]\ s\ t &= \mathcal{C}[\![c_1]\!]\,(\mathcal{C}[\![c_2]\!]\ s\ t)\ t \\
\mathcal{C}[\![\texttt{if (}b\texttt{) then } c_1 \texttt{ else } c_2]\!]\ s\ t &= \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ \mathcal{C}[\![c_1]\!]\ s\ t,\ \mathcal{C}[\![c_2]\!]\ s\ t) \\
\mathcal{C}[\![\texttt{while (}b\texttt{) do } c]\!]\ s\ t &= \mathrm{FIX}\,(\lambda f.\ \mathrm{IF}\,(\mathcal{B}[\![b]\!],\ \mathcal{C}[\![c]\!]\ f\ t,\ s)) \\
\mathcal{C}[\![\texttt{raise}]\!]\ s\ t &= t \\
\mathcal{C}[\![\texttt{try } c_1 \texttt{ catch } c_2]\!]\ s\ t &= \mathcal{C}[\![c_1]\!]\ s\ (\mathcal{C}[\![c_2]\!]\ s\ t)
\end{aligned}$$

Für ein Programm verwendet man üblicherweise die anfänglichen Fortsetzungen $s_0 = id$ und $t_0 = \bot$, sofern man Nichttermination und unbehandelte Ausnahmen nicht unterscheiden möchte. Ansonsten muss man Cont auf eine allgemeinere Antwortmenge wie z.B. $\Sigma \rightharpoonup (\mathbb{B} \times \Sigma)$ bei der Big-Step-Semantik verallgemeinern – in diesem Fall wären dann $s_0(\sigma) = (\mathbf{ff}, \sigma)$ und $t_0(\sigma) = (\mathbf{tt}, \sigma)$.

Beispiel 36. Sei $c = \texttt{try (x := 2; raise; x := 3) catch x := 4}$.

$$\begin{aligned}
\mathcal{C}\,[\![c]\!]\;s\;t\;\sigma &= \mathcal{C}\,[\![\texttt{x := 2; raise; x := 3}]\!]\;s\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t)\;\sigma \\
&= \mathcal{C}\,[\![\texttt{x := 2}]\!]\;(\mathcal{C}\,[\![\texttt{raise; x := 3}]\!]\;s\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t))\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t)\;\sigma \\
&= \mathcal{C}\,[\![\texttt{raise; x := 3}]\!]\;s\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t)\;(\sigma[\texttt{x}\mapsto 2]) \\
&= \mathcal{C}\,[\![\texttt{raise}]\!]\;(\mathcal{C}\,[\![\texttt{x := 3}]\!]\;s\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t))\;(\mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t)\;(\sigma[\texttt{x}\mapsto 2]) \\
&= \mathcal{C}\,[\![\texttt{x := 4}]\!]\;s\;t\;(\sigma[\texttt{x}\mapsto 2]) = s(\sigma[\texttt{x}\mapsto 4])
\end{aligned}$$

Damit gilt für $s = s_0 = id$ und $t = t_0 = \bot$: $\mathcal{C}\,[\![c]\!]\;s\;t\;\sigma = \mathcal{C}\,[\![c]\!]\;id\;\bot\;\sigma = \sigma[\texttt{x}\mapsto 4]$.

Noch ein paar Anmerkungen zur Continuation-Semantik $\mathcal{C}\,[\![\_]\!]$:

- $\mathcal{C}\,[\![\texttt{skip}]\!]$ ist nicht mehr einfach die Identität, sondern die Fortsetzung. Das tritt analog auch bei $\texttt{while (}b\texttt{) do }c$ auf.
- Die Reihenfolge von c_1 und c_2 in $\mathcal{C}\,[\![c_1\texttt{;}\ c_2]\!]$ ist nicht mehr wie bei $\mathcal{D}\,[\![c_1\texttt{;}\ c_2]\!]$ vertauscht.
- Das Funktional für den Fixpunktoperator in der Gleichung $\texttt{while (}b\texttt{) do }c$ entstammt der Rekursionsgleichung

  $$\mathcal{C}\,[\![\texttt{while (}b\texttt{) do }c]\!]\;st = \mathrm{IF}\;(\mathcal{B}\,[\![b]\!],\ \mathcal{C}\,[\![c]\!]\;(\mathcal{C}\,[\![\texttt{while (}b\texttt{) do }c]\!]\;s\;t)\;t,\ s)$$

  Es ist dabei implizit von den Parametern s und t abhängig: Sein kleinster Fixpunkt definiert $\mathcal{C}\,[\![\texttt{while (}b\texttt{) do }c]\!]\;s\;t$ nur für feste s und t.

  Analog zu Kap. 6.3 müsste man nun noch nachweisen, dass FIX wirklich definiert ist. Dies funktioniert nur, $\mathcal{C}\,[\![c]\!]\ \ f\ t$ *monoton* und *kettenstetig* in f ist. Einfacher ist es, wenn man auch s und t in den Fixpunktoperator hineinzieht:

  $$\mathrm{FIX}\;(\lambda f\ s\ t.\ \mathrm{IF}\;(\mathcal{B}\,[\![b]\!],\ \mathcal{C}\,[\![c]\!]\;(f\ s\ t)\ t,\ s))$$

  Dafür muss man auch noch die Approximationsordnung $\sqsubseteq$ auf $\mathsf{Cont} \Rightarrow \mathsf{Cont} \Rightarrow \mathsf{Cont}$ ausdehnen sowie zeigen, dass dies wieder eine ccpo ergibt und das Funktional immer monoton und kettenstetig ist. Macht man dies, so kann man allerdings zeigen, dass dabei der gleiche Fixpunkt wie bei unserer obigen Definition konstruiert wird.