Kontynuacje
MIMUW 2018/19

Michał Szafraniuk

4 lutego 2019

1 Semantyka denotacyjna w stylu kontynuacyjnym dla języka TINY

Dziedziny semantyczne

- $\mathsf{Val} = \mathbb{Z}$
- $\mathsf{Bool} = \{\mathsf{tt}, \mathsf{ff}\}$
- Ans: zbiór finalnych „odpowiedzi” programu
- $\mathsf{Cont} = \mathsf{Store} \rightharpoonup \mathsf{Ans}$: zbiór kontynuacji dla instrukcji
- $\mathsf{ECont} = \mathsf{Val} \to \mathsf{Cont}$: zbiór kontynuacji dla wyrażeń arytmetycznych
- $\mathsf{BCont} = \mathsf{Bool} \to \mathsf{Cont}$: zbiór kontynuacji dla wyrażeń boolowskich
- $\mathsf{DCont} = \mathsf{Env} \to \mathsf{Cont}$: zbiór kontynuacji dla deklaracji

Semantyka kontynuacyjna instrukcji

Typ

$$[\![\cdot]\!]_{\mathcal{I}} : \mathit{Instr} \to \mathsf{Env} \to \mathsf{Cont} \to \mathsf{Cont}$$

Intuicja

$$[\![i]\!]\rho = \lambda\kappa_1.\kappa_0$$

jest „maszynką”, do której jeśli wrzucimy przyszłość po wykonaniu instrukcji i (κ_1) to dostaniemy teraźniejszość tuż przed wykonaniem i (κ_0).

Denotacje

Narazie zakładamy, że semantyka wyrażeń jest dana.

- `skip`

  Teraźniejszość jest przyszłością:

$$[\![\texttt{skip}]\!]\rho = \lambda\kappa.\kappa$$

- przypisanie:

  Teraźniejszość jest przyszłością zrealizowaną w dzisiejszym składzie zmodyfikowanym o przypisanie:

  $$[\![x := e]\!]\rho = \lambda\kappa.\lambda s.\kappa s[\rho(x) \mapsto [\![e]\!]\rho s]$$

  Z pełną denotacją kontynuacyjną dla wyrażeń:

  $$[\![x := e]\!]\rho = \lambda\kappa.[\![e]\!]\rho(\lambda n.\lambda s.\kappa s[\rho(x) \mapsto n])$$

- złożenie instrukcji:

  $$[\![i_1;i_2]\!]\rho = \lambda\kappa_2.(\mathsf{let}\ \kappa_1 = [\![i_2]\!]\rho\kappa_2\ \mathsf{in}\ \kappa_0 = [\![i_1]\!]\rho\kappa_1)$$

  Jeśli przyszłością po i_2 jest κ_2 a teraźniejszością przed i_2 (czyli po i_1) jest κ_1 to κ_0 jest teraźniejszością przed i_1 jest κ_0.

  W skrócie:

  $$[\![i_1;i_2]\!]\rho = [\![i_1]\!]\rho \circ [\![i_2]\!]\rho$$

  czyli na odwrót niż zwykle bo idziemy od końca.

- `if`

  $$[\![\texttt{if}\ b\ \texttt{then}\ i_1\ \texttt{else}\ i_2]\!] = \lambda\kappa.(\lambda s.ifte([\![b]\!]\rho s, [\![i_1]\!]\rho\kappa s, [\![i_2]\!]\rho\kappa s))$$

- `while`

  $$[\![\texttt{while}\ b\ \mathsf{do}\ i]\!] = \lambda\kappa_1.\,\mathrm{fix}\ \Phi$$

  where

  $$\Phi\colon \mathsf{Cont} \to \mathsf{Cont}$$

  $$\Phi = \lambda\kappa_0.\lambda s. \begin{cases} ([\![i]\!]\rho)\kappa_0 s & \text{if} \quad [\![b]\!]\rho s = \mathtt{tt} \\ \kappa_1 s & \text{oth} \end{cases}$$

  gdyż równianie stałopunktowe:

  $$\kappa_0 = \lambda s.ifte([\![b]\!]\rho s, ([\![i]\!]\rho)\kappa_0 s, \kappa_1 s)$$

  κ_1 jest przyszłością po `while`'u - oczywiście nie zmieni się ona na skutek instrukcji `while`. Jeśli dozór okazałby się fałszywy to moja (pętląca się) teraźniejszość κ_0 jest tożsama z przyszłością κ_1 a jeśli okazałby się prawdziwy to petląca się teraźniejszość po wykonaniu i musi być tożsama pętlącą się teraźniejszością przed wykonaniem i.

Semantyka kontynuacyjna wyrażeń arytmetycznych

Typ

$$[\![\cdot]\!]_{\mathcal{E}} : \mathit{Instr} \to \mathsf{Env} \to \mathsf{ECont} \to \mathsf{Cont}$$

$$\mathsf{ECont} = \mathsf{Val} \to \mathsf{Cont}$$

$$\mathsf{Cont} = \mathsf{Store} \rightharpoonup \mathsf{Ans}$$

Intuicja

$$[\![e]\!]\rho = \lambda\kappa_E.\kappa$$

- $\kappa_E \in \mathsf{ECont}$ jest „wyjściem” wyrażenia: kontynuacją dla przyszłości po wyliczeniu e
- $\kappa \in \mathsf{Cont}$ jest „wejściem” wyrażenia: kontynuacją dla teraźniejszości sprzed wyliczenia e
- czyli dla przyszłości wyrażenia reprezentowanej przez κ_E jego teraźniejszością jest κ

Denotacje

- stała numeryczna

  $$[\![n]\!]\rho = \lambda\kappa_E.\lambda s.\kappa_E \underline{n} s = \lambda\kappa_E.\kappa_E \underline{n}$$

  Jeżeli po wyliczeniu n jest przyszłość κ_E to teraźniejszością jest ta przyszłość z zaaplikowaną wartością $\underline{n}$.

- zmienna

  $$[\![x]\!]\rho = \lambda\kappa_E.\lambda s.\kappa_E(s\rho x)s$$

  Jeżeli przyszłością ma być κ_E to teraźniejszością jest funkcja, która dla danego składu wyliczy wartość zmiennej w tym składzie (i środowisku) $(s\rho x)$ a następnie zaaplikuje tą wartość i ten skład do przyszłości.

- suma wyrażeń

  $$[\![e_1 + e_2]\!]\rho = \lambda\kappa_E.[\![e_1]\!]\rho\big(\lambda n_1.[\![e_2]\!]\rho(\lambda n_2.\kappa_E(n_1 + n_2))\big)$$

  Ustalmy, że $\kappa_E \in \mathsf{ECont}$ jest kontynuacją dla przyszłości po wyliczeniu $e_1 + e_2$. Chcemy znaleźć kontynuację dla teraźniejszości sprzed tego wyliczenia. W tym celu najpierw musimy poznać kontynuację $\kappa'_E \in \mathsf{ECont}$ pośrednią pomiędzy wyliczaniem e_1 a e_2. Nie używamy tutaj składów, gdyż w standardowej semantyce wyrażenia nie mają efektów ubocznych.

Semantyka kontynuacyjna wyrażeń boolowskich

Typ

$$[\![\cdot]\!]_{\mathcal{B}} : BExpr \to \mathsf{Env} \to \mathsf{BCont} \to \mathsf{Cont}$$
$$\mathsf{BCont} = \mathsf{Bool} \to \mathsf{Cont}$$
$$\mathsf{Cont} = \mathsf{Store} \rightharpoonup \mathsf{Ans}$$

Intuicja

$$[\![b]\!]\rho = \lambda\kappa_B.\kappa$$

jak dla wyrażeń arytmetycznych.

Denotacje

- stałe

$$[\![\texttt{true}]\!]\rho = \lambda\kappa_B.\kappa_B\mathsf{tt}$$
$$[\![\texttt{false}]\!]\rho = \lambda\kappa_B.\kappa_B\mathsf{ff}$$

  Jeżeli kontynuacją dla przyszłości jest κ_B to teraźniejszością jest ta przyszłość z zaaplikowaną wartością tt/ff.

- nierówność

$$[\![e_1 \leqslant e_2]\!]\rho = \lambda\kappa_B.[\![e_1]\!]\rho\big(\lambda n_1.[\![e_2]\!]\rho(\lambda n_2.\kappa_B\ ifte(n_1 \leqslant n_2, \mathsf{tt}, \mathsf{ff}))\big)$$

Semantyka kontynuacyjna deklaracji zmiennych

$$VDecl \ni d_V ::= \texttt{var}\ x; d_V \mid \epsilon$$

Typ

$$[\![\cdot]\!]_{\mathcal{D}} :\ VDecl \to \mathsf{VEnv} \to \mathsf{VDCont} \to \mathsf{Cont}$$
$$\mathsf{VDCont} = \mathsf{VEnv} \to \mathsf{Cont}$$
$$\mathsf{Cont} = \mathsf{Store} \rightharpoonup \mathsf{Ans}$$

Intuicja

$$[\![b]\!]\rho = \lambda\kappa_B.\kappa$$

jak dla wyrażeń arytmetycznych.

Denotacje

- deklaracja pusta

$$[\![\epsilon]\!]\rho = \lambda\kappa_D.\kappa_D\rho$$

- deklaracja zmiennej nienicjowanej

$$[\![\texttt{var}\ x]\!]\rho = \lambda\kappa_D.\kappa_D\rho[x \mapsto newloc(s)]$$

- deklaracja zmiennej inicjowanej zerem

$$[\![\texttt{var}\ x := 0]\!]\rho = \lambda\kappa_D.\lambda s.\kappa_D\rho[x \mapsto l]s[l \mapsto 0]$$

  where $l = newloc(s)$.

- deklaracja zmiennej inicjowanej

$$[\![\texttt{var}\ x := e]\!]\rho = \lambda\kappa_D.[\![e]\!]\rho(\lambda n.\lambda s.\kappa_D\rho[x \mapsto l]s[l \mapsto n])$$

  where $l = newloc(s)$.

- złożenie deklaracji

$$[\![d_1; d_2]\!]\rho = \lambda\kappa_D.[\![d_1]\!]\rho\big(\lambda\rho'.[\![d_2]\!]\rho'\kappa_D\big)$$

Semantyka kontynuacyjna rozszerzonych instrukcji

Typ

$$[\![\cdot]\!]_{\mathcal{I}} : \mathit{Instr} \to \mathsf{Env} \to \mathsf{Cont} \to \mathsf{Cont}$$

$$[\![\cdot]\!]_{\mathcal{D}} : \mathit{VDecl} \to \mathsf{Env} \to \mathsf{DCont} \to \mathsf{Cont}$$

$$\mathsf{DCont} = \mathsf{Env} \to \mathsf{Cont}$$

$$\mathsf{Cont} = \mathsf{Store} \rightharpoonup \mathsf{Ans}$$

- blok z deklaracją zmiennych

$$[\![\texttt{begin}\, d; i\ \texttt{end}]\!] = \lambda\kappa \in \mathsf{Cont}.[\![d]\!]\rho\big(\lambda\rho' \in \mathsf{Env}.[\![i]\!]\rho'\kappa\big)$$