

SEMWER zadanie 2, am418402

December 30, 2021

Dziedziny semantyczne

Stan

Ponieważ wszystkie zmienne są globalne, oraz traktujemy je jako zadeklarowane i z nadaną wartością, to nie potrzebujemy środowiska zmiennych, a stan ma po prostu postać

$$\text{State} = \text{Var} \longrightarrow \text{Int}$$

i skoro wszystkie zmienne mają wartość, nie jest to funkcja częściowa. Powstaje tutaj pytanie, jaką wartość mają zmienne, którym nie nadano jeszcze w programie żadnej wartości - wszak można się do nich odwołać. To jednak nie zależy od denotacji instrukcji i wyrażeń, więc nie jest częścią zadania. Można to ustalić np. podając stan "początkowy" w denotacji programów.

Typy kontynuacji

Skoro wynikiem działania instrukcji ma być stan końcowy, to kontynuacje mają postać

$$\text{Cont} = \text{State} \rightarrow \text{State}$$

Ale dla rozróżnienia wprowadzę typ $\text{Ans} = \text{State}$ i będę pisał

$$\text{Cont} = \text{State} \rightarrow \text{Ans}$$

Dalej, dla kontynuacji wyrażeń arytmetycznych:

$$\text{Cont}_E = \text{Int} \longrightarrow \text{State} \rightarrow \text{Ans}$$

Tutaj istotne jest, że wyrażenia mogą zmieniać stan, więc nie może być $\text{Cont}_E = \text{Num} \rightarrow \text{Ans}$. Dla wyrażeń boolowskich:

$$\text{Cont}_B = \text{Bool} \longrightarrow \text{State} \rightarrow \text{Ans}$$

a dla deklaracji:

$$\text{Cont}_D = \text{FEnv} \rightarrow \text{Ans}$$

Gdzie FEnv to opisane niżej środowisko funkcji. Tym razem jest istotne, że deklaracje nie zmieniają stanu.

Środowisko funkcji

Środowisko funkcji ma postać

$$\text{FEnv} = \text{FName} \rightarrow \text{Fun}$$

Gdzie FName to nazwy funkcji, natomiast funkcje reprezentuje typ Fun:

$$\text{Fun} = (\text{Cont} \rightarrow \text{Cont}_E \rightarrow \text{State} \rightarrow \text{Ans}) \times \text{EXPR}$$

gdzie

$$\text{EXPR} = \text{FEnv} \rightarrow \text{Cont}_E \rightarrow \text{State} \rightarrow \text{Ans}$$

Jak warto zauważyć, w pierwszej części produktu podajemy jako argumenty dwie różne kontynuacje. Ma to na celu umożliwienie zarówno wykonania całości ciała funkcji, jak i wyjście z niej wcześniej za pomocą instrukcji `return`. Ta pierwsza część intuicyjnie jest odpowiedzialna za wykonanie ciała funkcji. Natomiast druga część produktu to wyrażenie odpowiedzialne za domyślny wynik działania funkcji. W środowisku należy pamiętać całe to wyrażenie, gdyż będzie ono wyliczane przed każdym wywołaniem funkcji. Typ EXPR jest taki sam jak opisany niżej typ $\mathcal{E}[e]$, czyli - co istotne - nie ma jeszcze zaaplikowanego środowiska funkcji. Skutkuje to tym, że przy wyliczaniu wyrażenia przed wywołaniem funkcji obowiązuje każdorazowo środowisko funkcji z momentu wywołania. Jest to świadoma decyzja z mojej strony. W zadaniu nie jest doprecyzowane, czy ma wtedy obowiązywać środowisko z miejsca wywołania, czy deklaracji - mowa tylko o stanie. W mojej opinii takie rozwiązanie jest bardziej intuicyjne.

Typy funkcji semantycznych

Dla instrukcji:

$$\mathcal{J}[] : \text{Instr} \rightarrow \text{FEnv} \rightarrow \text{Cont} \rightarrow \text{Cont}_E \rightarrow \text{State} \rightarrow \text{Ans}$$

Ponownie, i z analogicznych powodów, mamy tu dwie różne kontynuacje. Dla wyrażeń arytmetycznych jest:

$$\mathcal{E}[] : \text{Expr} \rightarrow \text{FEnv} \rightarrow \text{Cont}_E \rightarrow \text{State} \rightarrow \text{Ans}$$

i tym razem oczywiście nie potrzebujemy dwóch różnych kontynuacji. Dla wyrażeń boolowskich analogicznie:

$$\mathcal{B}[] : \text{BExpr} \rightarrow \text{FEnv} \rightarrow \text{Cont}_B \rightarrow \text{State} \rightarrow \text{Ans}$$

Dla deklaracji:

$$\mathcal{D}[] : \text{FDecl} \rightarrow \text{FEnv} \rightarrow \text{Cont}_D \rightarrow \text{Ans}$$

Denotacje

Denotacje wyrażeń

$$\mathcal{E}[n] \rho_F \kappa_E s = \kappa_E(\mathcal{N}[n]) s$$

$$\mathcal{E}[\![x]\!] \rho_F \kappa_E s = \kappa_E(s \ x) \ s$$

$$\mathcal{E}[\![e_1 + e_2]\!] \rho_F \kappa_E s = \mathcal{E}[\![e_1]\!] \rho_F (\lambda n_1. \mathcal{E}[\![e_2]\!] \rho_F (\lambda n_2. \kappa_E(n_1 + n_2))) \ s$$

Denotacje dla odejmowania i mnożenia są analogiczne.

$$\mathcal{E}[\![f()]\!] \rho_F \kappa_E s = e_d \rho_F (\lambda n. \beta \kappa_E(n) \kappa_E) \ s$$

gdzie

$$\rho_F(f) = (\beta, e_d)$$

Innymi słowy, wyciągamy ze środowiska funkcji krotkę oznaczającą ciało funkcji i jej wyrażenie domyślne (tj. określające domyślny wynik), i przekazujemy kontrolę wyrażeniu domyślnemu, mówiąc, że tym, co się stanie z jego wynikiem n , będzie nowo skonstruowana kontynuacja wyrażenia arytmetycznego zależna od n . Ta kontynuacja z kolei jest - nieformalnie mówiąc - wykonaniem ciała funkcji, biorąc jako kontynuację domyślną $\kappa_E(n)$, a jako kontynuację dla instrukcji **return** - κ_E .

$$\mathcal{B}[\![\text{true}]\!] \rho_F \kappa_B s = \kappa_B \text{ tt } s$$

analogicznie dla fałszu.

$$\mathcal{B}[\![\text{not } b]\!] \rho_F \kappa_B s = \mathcal{B}[\![b]\!] \rho_F (\lambda d. \kappa_B (\neg d))s$$

$$\mathcal{B}[\![b_1 \wedge b_2]\!] \rho_F \kappa_B s = \mathcal{B}[\![b_1]\!] \rho_F (\lambda d_1. \mathcal{B}[\![b_2]\!] \rho_F (\lambda d_2. \kappa_B(d_1 \wedge d_2)))s$$

$$\mathcal{B}[\![e_1 < e_2]\!] \rho_F \kappa_B s = \mathcal{E}[\![e_1]\!] \rho_F (\lambda n_1. \mathcal{E}[\![e_2]\!] \rho_F (\lambda n_2. \kappa_B(n_1 < n_2))) \ s$$

Dla $e_1 = e_2$ analogicznie. Drobną uwagę: być może użyty tu zapis $n_1 < n_2$ nie jest do końca formalny. Oczywiście chodzi tu o wyrażenie boolowskie które przyjmuje wartość tt , gdy $n_1 < n_2$, a ff w przeciwnym wypadku.

Denotacje instrukcji

$$\mathcal{J}[\![x := e]\!] \rho_F \kappa \kappa_E s = \mathcal{E}[\![e]\!] \rho_F (\lambda n. \lambda s'. \kappa \ s'[x \mapsto n]) \ s$$

$$\mathcal{J}[\![I_1; I_2]\!] \rho_F \kappa \kappa_E s = \mathcal{J}[\![I_1]\!] \rho_F (\mathcal{J}[\![I_2]\!] \rho_F \kappa \kappa_E) \ \kappa_E \ s$$

$$\mathcal{J}[\![\text{if } b \text{ then } I_1 \text{ else } I_2]\!] \rho_F \kappa \kappa_E s = \mathcal{B}[\![b]\!] \rho_F (\lambda d. \text{ifte}(d, \mathcal{J}[\![I_1]\!] \rho_F \kappa \kappa_E, \mathcal{J}[\![I_2]\!] \rho_F \kappa \kappa_E)) \ s$$

”Robocza”, niestałopunktowa denotacja instrukcji **while** wyglądałaby tak:

$$\mathcal{J}[\![\text{while } b \text{ do } I]\!] \rho_F \kappa \kappa_E = \mathcal{B}[\![b]\!] \rho_F (\lambda d. \text{ifte}(d, \mathcal{J}[\![I]\!] \rho_F (\mathcal{J}[\![\text{while } b \text{ do } I]\!] \rho_F \kappa \kappa_E) \ \kappa_E, \kappa))$$

Czyli $\mathcal{I}[\text{while } b \text{ do } I] \rho_F \kappa \kappa_E = fix(\Phi)$ dla

$$\Phi(F) = \mathcal{B}[b] \rho_F (\lambda d. ifte(d, \mathcal{I}[I] \rho_F F \kappa_E, \kappa))$$

$$\mathcal{I}[\text{begin } d_F \text{ } I \text{ end}] \rho_F \kappa \kappa_E s = \mathcal{D}[d_F] \rho_F (\lambda \rho. \mathcal{I}[I] \rho \kappa \kappa_E s)$$

$$\mathcal{I}[\text{return } e] \rho_F \kappa \kappa_E s = \mathcal{E}[e] \rho_F (\lambda n. \kappa_E(n)) s$$

Denotacje deklaracji

Tak jak przy instrukcji **while**, możemy roboczo skonstruować ”definicję” deklaracji funkcji (potencjalnie rekurencyjnej), która nie jest kompozycyjna:

$$\mathcal{D}[\text{fun } f \text{ result } e \text{ do } (I)] \rho_F \kappa_D = \kappa_D (\rho_F[f \mapsto F])$$

gdzie

$$F = \langle \mathcal{I}[I] \rho_F[f \mapsto F], \mathcal{E}[e] \rangle$$

I teraz możemy przepisać to na poprawną definicję stałopunktową:

$$\mathcal{D}[\text{fun } f \text{ result } e \text{ do } (I)] \rho_F \kappa_D = \kappa_D (\rho_F[f \mapsto fix(\Phi)])$$

gdzie

$$\Phi(F) = \langle \mathcal{I}[I] \rho_F[f \mapsto F], \mathcal{E}[e] \rangle$$

Dalej dla złożenia deklaracji:

$$\mathcal{D}[d_{F1}; d_{F2}] \rho_F \kappa_D = \mathcal{D}[d_{F1}] \rho_F (\lambda \rho. \mathcal{D}[d_{F2}] \rho \kappa_D)$$