SEMWER zadanie 2, am418402

December 28, 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

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

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

$$\mathrm{Cont} = \mathrm{State} \rightharpoonup \mathrm{State}$$

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

$$\mathrm{Cont} = \mathrm{State} \rightharpoonup \mathrm{Ans}$$

Dalej, dla kontynuacji wyrażeń arytmetycznych:

$$\mathrm{Cont}_E = \mathrm{Num} \longrightarrow \mathrm{State} \rightharpoonup \mathrm{Ans}$$

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

$$\mathrm{Cont}_B = \mathrm{Bool} \longrightarrow \mathrm{State} \rightharpoonup \mathrm{Ans}$$

a dla deklaracji:

$$\mathrm{Cont}_D = \mathrm{FEnv} \rightharpoonup \mathrm{Ans}$$

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

Środowisko funkcji

Środowisko funkcji ma postać

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

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

$$\text{Fun} = (\text{Cont} \longrightarrow \text{Cont}_E \longrightarrow \text{State} \rightharpoonup \text{Ans}) \times \text{Expr}$$

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.

Typy funkcji semantycznych

Dla instrukcji:

$$\mathcal{I}[\![]\!] : \text{Instr} \longrightarrow \text{FEnv} \longrightarrow \text{Cont} \longrightarrow \text{Cont}_E \longrightarrow \text{State} \rightharpoonup \text{Ans}$$

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

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

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

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

Dla deklaracji:

$$\mathcal{D}[\![]\!] : \text{FDecl} \longrightarrow \text{FEnv} \longrightarrow \text{Cont}_D \longrightarrow \text{State} \rightharpoonup \text{Ans}$$

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.

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

gdzie

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

Innymi słowy, wyciągamy ze środowiska funkcji krotkę oznaczającą ciało funkcji i jej wyrażenie domyślne, 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.

todo sprawdzić - na slajdach jest chyba błąd, kappa$_E zamiastkappa_B$

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

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

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

Dla $e_1 = e_2$ analogicznie.

wzmianka że denotacja mniejszości wyrażeń działa tak samo

Denotacje instrukcji

denotacja przypisania

denotacja średnika

denotacja ifa

denotacja while'a

denotacja bloku

denotacja returna?

Denotacje deklaracji

widoczność funkcji jest statyczna - tzn jeśli mamy funkcję f która wywołuje funkcję g, i w momencie deklaracji f jest widoczna g, a potem w jakimś bloku mamy przesłonioną definicję g i wywołamy f, to f skorzysta z tego g, które zobaczyła w momencie deklaracji, a nie wywołania. innymi słowy funkcje pamiętają środowisko funkcji z momentu deklaracji (wywołanie nie przyjmuje środowiska funkcji jako argumentu - no w każdym razie, na slajdach było jak zrobić widoczność statyczną)

funkcje są rekurencyjne, więc definicja deklaracji funkcji musi być stałopunktowa.

denotacja deklaracji funkcji

denotacja złożenia deklaracji funkcji też