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

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

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

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

Ale dla rozróżnienia wprowadzę typ Ans = State i będę pisał

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

Dalej, dla kontynuacji wyrażeń arytmetycznych:

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

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

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

a dla deklaracji:

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

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

Środowisko funkcji

Środowisko funkcji ma postać

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

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.

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

analogicznie dla fałszu.

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

$$\mathcal{B}[\![\mathrm{not}\ b]\!]\rho_F\ \kappa_B\ s = \mathcal{B}[\![b]\!]\ \rho_F\ (\lambda d.\ \kappa_B\ ifte(d,\ \mathrm{ff},\ \mathrm{tt}))s$$

uruchamiamy wyrażenie b mówiąc mu że funkcją zaaplikowaną do jego wyniku będzie jakaś lambda ifte b kontynuacja fałszu kontynuacja prawdy

denotacja koniunkcji sfer

denotacja równości dwóch wyrażeń

wzmianka że denotacja znaku mniejszości 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