Reaktywny język funkcyjny FElm

Semantyka języków programowania

Filip Pawlak Rafał Łukaszewski

Wrocław, dnia 20 lutego 2014 r.

1. Temat

Celem projektu jest zadanie semantyki oraz implementacja interpretera dla reaktywnego języka funkcyjnego będącego minimalnym podzbiorem języka Elm, opisanym jako FElm w pracy: Asynchronous Functional Reactive Programming for GUIs, na której oparty jest projekt.

2. Podstawowe założenia

- gorliwy
- silnie, ale dynamicznie typowany
- podstawowe konstrukcje dla sygnałów, tj. lift oraz foldp, ale bez async
- podobnie jak w Elmie - brak sygnału sygnałów oraz sygnałów definiowanych rekurencyjnie
- typy danych: liczby całkowite, wartości boolowskie i oczywiście sygnały
- wejście - zdefiniowany z góry zbiór sygnałów wejściowych

3. Składnia

Składnia języka jest całkowicie zgodna z [1, p. 3.1], w związku z czym nie podajemy jej definicji ponownie w tym dokumencie.

4. Semantyka

Zgodnie z [1, p. 3.3] w bazowej wersji systemu proces interpretacji programu jest podzielony na dwa etapy. Wpierw wykonywana jest redukcja wszystkich konstrukcji funkcyjnych do termu w języku pośrednim. W tej postaci pozostają już tylko konstrukcje sygnałów i korzystamy z niej w procesie rozwiązywania zależności pomiędzy konstrukcjami sygnałowymi. Wzorując się na intuicjach zawartych w [1, p. 3.3.2] explicite tworzymy graf sygnału w trakcie etapu drugiego, z którego to grafu ostatecznie korzystamy podczas przetwarzania nadchodzących zdarzeń.

Sygnały definiujemy jako strumienie dyskretnych wartości, oraz przetwarzamy je synchronicznie, tj. kiedy w sygnale wejściowym pojawi się nowa wartość musimy rozpropagować ją w całym grafie zanim będziemy mogli obsłużyć nowe zdarzenie. Przyjmujemy, że zbiór sygnałów wejściowych jest stały i zdefiniowany z góry. Podobnie jak w (mimo, że nie jest to dosłownie wyszczególnione), czas również traktujemy jak zwykły sygnał, tj. abstrahujemy od pojęcia czasu podczas definiowania semantyki.

4.1. Ewaluacja funkcyjna

W podstawowej wersji systemu podczas pierwszego etapu interpretacji korzystamy bezpośrednio z semantyki operacyjnej opisanej w [1, p. 3.3.1], z regułami zdefiniowanymi w [1, fig. 6].

4.2. Budowa grafu

Ten etap ewaluacji opisujemy w formacie semantyki denotacyjnej. Wierzchołki w grafie reprezentujemy następująco:

$$Vertex : N \times Expr \times \{LiftV \times Expr, FoldpV \times Expr, InputV\} \tag{4.1}$$

Pierwszy element krotki to numer wierzchołka, drugi to jego wartość, zaś trzeci określa jego typ (wierzchołki typu LiftV i FoldpV dodatkowo przechowują funkcję przekazaną jako argument w wywołaniu odpowiednio $\texttt{lift}_n$ i `foldp`). Pomijamy definicję funkcji value : Vertex $\rightarrow$ Expr, która dla danego wierzchołka zwraca jego wartość. Wymagamy by krawędzie pomiędzy dwoma wierzchołkami były numerowane (z uwagi na to, że kolejność argumentów sygnałowych jest istotna):

$$Edge : Vertex \times Vertex \times N \tag{4.2}$$

Pomijamy reprezentację grafu, jednak żądamy by istniała funkcja Next : Graph → N, która dla danego grafu podaje kolejny dostępny numer wierzchołka. Poprzez N oznaczamy funkcję typu Expr → Expr, będącą funkcją semantyczną zadaną przez reguły ewaluacji funkcyjnej. Definiujemy również środowisko:

$$Env : Var \to Vertex \tag{4.3}$$

wraz z operacją update typu Var → Vertex → Env → Env. Definiujemy funkcję semantyczną D:

$$D : Expr \to Env \to Graph \to Vertex \times Graph \tag{4.4}$$

$$D[\![\mathrm{x}]\!]\ env\ g = \langle env\ x,\ g\rangle \tag{4.5}$$

$$D[\![\texttt{lift}_n\ \mathrm{f}\ \mathrm{s}_1\ ...\ \mathrm{s}_n]\!] = \langle v,\ g'\rangle \tag{4.6}$$

$$\text{gdzie } \langle v_1,\ g_1\rangle = D[\![\mathrm{s}_1]\!]\ env\ g \tag{4.7}$$

$$\langle v_i,\ g_i\rangle = D[\![\mathrm{s}_i]\!]\ env\ g_{i-1} \quad \text{dla } 2 \le i \le n \tag{4.8}$$

$$defaultV = N[\![\mathrm{f}\ (\mathrm{value}\ \mathrm{v}_1)\ ...\ (\mathrm{value}\ \mathrm{v}_n)]\!] \tag{4.9}$$

$$v = \langle Next(g_n),\ defaultV,\ \langle LiftV,\ f\rangle\rangle \tag{4.10}$$

$$V(g') = V(g_n) \cup \{v\} \tag{4.11}$$

$$E(g') = E(g_n) \cup \{\langle v_i,\ v,\ i\rangle : 1 \le i \le n\} \tag{4.12}$$

$$D[\![\texttt{foldp}\ \mathrm{f}\ \mathrm{d}\ \mathrm{s}]\!] = \langle v,\ g'\rangle \tag{4.13}$$

$$\text{gdzie } \langle v_s,\ g_s\rangle = D[\![\mathrm{s}]\!]\ env\ g \tag{4.14}$$

$$v = \langle Next(g_s),\ d,\ \langle Foldp,\ f\rangle\rangle \tag{4.15}$$

$$V(g') = V(g_s) \cup \{v\} \tag{4.16}$$

$$E(g') = E(g_s) \cup \{\langle v_s,\ v,\ 1\rangle\} \tag{4.17}$$

$$D[\![\texttt{let}\ \mathrm{l} = \mathrm{s}\ \texttt{in}\ \mathrm{r}]\!] = \langle v_r,\ g_r\rangle \tag{4.18}$$

$$\text{gdzie } \langle v_s,\ g_s\rangle = D[\![\mathrm{s}]\!]\ env\ g \tag{4.19}$$

$$env' = update\ l\ v_s\ env \tag{4.20}$$

$$\langle v_r,\ g_r\rangle = D[\![\mathrm{r}]\!]\ env'\ g_s \tag{4.21}$$

4.3. Inne podejście

Powyższa semantyka wiernie bazuje na pracy [1] i klarownie oddziela od siebie ewaluację elementów funkcyjnych i sygnałowych. Wadą takiego podejścia jest nieoczekiwane zatrzymywanie się ewaluacji dla niektórych wyrażeń,

jak np.:

```
let y = (let x = Window.width in \z -> x)
in y ()
```

Jednym z możliwych rozwiązań tej sytuacji jest połączenie obu etapów ewaluacji w jeden i budowanie grafu równolegle z ewaluacją elementów funkcyjnych. Nową relację redukcji oznaczamy w ten sam sposób, lecz teraz jest ona określona na zbiorze Expr $\times$ Graph. Do składni abstrakcyjnej dodajemy pomocnicze wyrażenia `signal` i (i $\in$ N) reprezentujące sygnały (wierzchołki w grafie). Dodajemy je również do kategorii wartości. Konteksty ewaluacyjne definiujemy niemal identycznie, jednak z definicji kontekstu usuwamy produkcję której prawą stroną jest `let` x = s `in` E (zatem nie ewaluujemy ciała konstrukcji `let`). Definiujemy regułę CONTEXT dla nowej relacji:

$$\frac{\langle \mathrm{e},\, \mathrm{g}\rangle \to \langle \mathrm{e}',\, \mathrm{g}'\rangle}{\langle \mathrm{E[e]},\, \mathrm{g}\rangle \to \langle \mathrm{E[e']},\, \mathrm{g}\rangle} \quad \text{CONTEXT}$$

Do naszej semantyki włączamy reguły APPLICATION, OP, COND-TRUE, COND-FALSE i REDUCE ([1, p. 3.3.1]) wraz z poniższą regułą umożliwiającą ich stosowanie:

$$\frac{\mathrm{e} \to \mathrm{e}'}{\langle \mathrm{e},\, \mathrm{g}\rangle \to \langle \mathrm{e}',\, \mathrm{g}\rangle}$$

Kolejne dwie reguły budują fragment grafu odpowiadający rozpatrywanej konstrukcji sygnałowej oraz zastępują ją wyrażeniem `signal` i, gdzie i to numer wierzchołka odpowiadającego wynikowemu sygnałowi:

$$\frac{}{\langle \texttt{lift}_n \text{ val } (\texttt{signal } \mathrm{i}_1) \text{ ... } (\texttt{signal } \mathrm{i}_n),\, \mathrm{g}\rangle \to \langle \texttt{signal } \mathrm{i},\, \mathrm{g}'\rangle} \quad \text{LIFT}$$

$$\text{gdzie } defaultV = N[\![\text{val (value } \mathrm{v}_{i_1}) \text{ ... (value } \mathrm{v}_{i_n})]\!] \tag{4.22}$$

$$i = Next(g) \tag{4.23}$$

$$v = \langle i,\, defaultV,\, \langle LiftV,\, val\rangle\rangle \tag{4.24}$$

$$V(g') = V(g) \cup \{v\} \tag{4.25}$$

$$E(g') = E(g) \cup \{\langle v_{i_j},\, v,\, j\rangle : 1 \le j \le n\} \tag{4.26}$$

$$\frac{}{\langle \texttt{foldp } \mathrm{val}_1 \ \mathrm{val}_2 \ (\texttt{signal } \mathrm{i}),\, \mathrm{g}\rangle \to \langle \texttt{signal } \mathrm{i}',\, \mathrm{g}'\rangle} \quad \text{FOLDP}$$

(4.27) $$\text{gdzie } v = \langle Next(g),\ val_2,\ \langle FoldpV,\ val_1\rangle\rangle$$

(4.28) $$V(g') = V(g) \cup \{v\}$$

(4.29) $$E(g') = E(g) \cup \{\langle v_i,\ v,\ 1\rangle\}$$

2

4.4. Propagacja zdarzeń w grafie

Aby reprezentować pojawienie się nowej wartości w sygnale lub jej brak dodajemy do krawędzi grafu dodatkową informację typu:

$$type\ Event\ \alpha = NoChange\ \alpha \mid Change\ \alpha$$

Pojawienie się nowej wartości w wierzchołku będziemy propagować przez ustawienie na wychodzących z niego krawędziach etykiety $Change\ v$ (gdzie v oznacza nową wartość znajdującą się w wierzchołku), natomiast gdy nic nowego nie pojawi się w wierzchołku oznaczymy jego krawędzie etykietami $NoChange\ v$ (w tym przypadku v to ostatnia wartość jaka pojawiła się w danym sygnale). W ten sposób unikniemy ponownego obliczania wartości w wierzchołkach, kiedy nie jest to konieczne.

Kiedy posiadamy już poprawnie zbudowany graf propagację nowych wartości pojawiających się w sygnałach wejściowych przeprowadzamy przez:

1. oznaczenie krawędzi wychodzących wierzchołka danego sygnału wejściowego etykietą $Change\ v$
2. oznaczenie krawędzi wychodzących reszty wierzchołków sygnałów wejściowych etykietą $NoChange\ v$
3. odpowiednie przetworzenie każdego wierzchołka (nie reprezentującego sygnału wejściowego) według porządku topologicznego grafu zgodnie z regułami przedstawionymi poniżej:

$$\frac{\langle v_s, v, Change\ val\rangle \in E(g)}{\langle v = \langle d, \langle FoldpV, f\rangle\rangle, g\rangle \Rightarrow g[v/\langle nV, \langle FoldpV, f\rangle\rangle][\langle v, *, \_\rangle/\langle v, *, Change\ nV\rangle]}$$

gdzie $nV = N[\![f\ val\ d]\!]$

5. Uwagi dot. implementacji

Literatura

[1] E. Czaplicki, S. Chong, *Asynchronous Functional Reactive Programming for GUIs*, PLDI'13, 2013.