Pixy formal semantics

A. Finn Hackett, Reed Mullanix

February 5, 2018

1 Introduction

The evaluation rules of Pixy are split into two steps: construction and evaluation.

First, any preprocessing is performed such as determining and allocating queue sizes or scanning for free variables.

Then the evaluation rules are applied to the result of this step in order to execute the progam.

2 Utilities

$$\begin{aligned} \text{apply}(E, nil) &= \exists v \in freevariables(E), \text{apply}(E[v/nil], nil) \\ \text{apply}(E, <>) &= E \\ \text{apply}(E, << n, v >, R... >) &= apply(E[n/v], R) \end{aligned}$$

3 If

If has some quite interesting semantics - unlike in many languages it does not completely skip the evaluation of the subexpression it does not select. Instead, it always executes both subexpressions except that when a subexpression is not selected the inputs are replaced by nil. This has the effect of synchronising time between both branches regardless of which if chosen, while avoiding the catastrophically bad performance of actually providing data for both branches to process.

3.1 Evaluation

$$\frac{C => nil, \text{apply}(T, nil \dots) => \_, \text{apply}(F, nil \dots) => \_}{\text{if}(C, < T, T_{args} >, < F, F_{args} >) => nil} \texttt{Eval - if - nil}$$

$$\frac{C => true, \text{apply}(T, T_{args}) => T_{val}, \text{apply}(F, nil) => \_}{\text{if}(C, < T, T_{args} >, < F, F_{args} >) => T_{val}} \texttt{Eval - if - true}$$

$$\frac{C => false, \text{apply}(T, nil) => \_, \text{apply}(F, F_{args}) => F_{val}}{\text{if}(C, < T, T_{args} >, < F, F_{args} >) => F_{val}} \texttt{Eval-if-false}$$

3.2 Construction

$$\frac{\begin{array}{c}\Gamma| - S|C => \Gamma| - S_1|C_{expr}, C_{vars} \\ \Gamma| - S_1|Tsrc => \Gamma| - S_2|T_{expr}, T_{vars} \\ \Gamma| - S_2|Fsrc => \Gamma| - S_3|F_{expr}, F_{vars}\end{array}}{\Gamma| - S \left| \begin{array}{c}\text{if } C \text{ then } T \\ \text{else } F\end{array} \right. => \begin{array}{c}\text{if}(C_{expr}, < T_{expr}, T_{vars} >, < F_{expr}, F_{vars} >), \\ C_{vars} \cup T_{vars} \cup F_{vars}\end{array}} \texttt{Construct-if}$$

4 fby

4.1 Evaluation

$$\frac{S => false, L => nil, R => nil}{\text{fby}(L, R, S, Q) => nil} \texttt{Eval-fby-1}$$

$$\frac{S => false, L => nil, R => R_{val}, R_{val} \neq nil, push(Q, R_{val})}{\text{fby}(L, R, S, Q) => nil} \texttt{Eval-fby-2}$$

$$\frac{S => false, L => L_{val}, L_{val} \neq nil, R => R_{val}, R_{val} \neq nil, push(Q, R_{val}), set(S, true)}{\text{fby}(L, R, S, Q) => L_{val}} \texttt{Eval-fby-3}$$

$$\frac{S => true, R => R_{val}, R_{val} \neq nil, \neg empty(Q), push(Q, R_{val})}{\text{fby}(L, R, S, Q) => pop(Q)} \texttt{Eval-fby-4}$$

$$\frac{S => true, R => R_{val}, R_{val} \neq nil, empty(Q)}{\text{fby}(L, R, S, Q) => R_{val}} \texttt{Eval-fby-5}$$

$$\frac{S => true, R => nil, empty(Q)}{\text{fby}(L, R, S, Q) => nil} \texttt{Eval-fby-6}$$

$$\frac{S => true, R => nil, \neg empty(Q)}{\text{fby}(L, R, S, Q) => pop(Q)} \texttt{Eval-fby-7}$$

4.2 Construction

$$\frac{\begin{array}{c}\Gamma| - S|L => \Gamma| - S_1|L_{expr}, L_{vars} \\ \Gamma| - S_1|R => \Gamma| - S_2|R_{expr}, R_{vars} \\ d = maxdistance(L_{expr}, R_{expr}) \\ < Q_f, \Gamma' >= fresh(Q, \Gamma) \\ < P_f, \Gamma'' >= fresh(P, \Gamma') \\ S_3 = alloc(d, Q_f, S_2) \\ S_4 = alloc(P_f, S_3)\end{array}}{\Gamma| - S|L \text{ fby } R => \Gamma''| - S_4| \text{fby}(L_{expr}, R_{expr}, P_f, Q_f), L_{vars} \cup R_{vars}} \texttt{Construct-fby}$$

5 check

5.1 Evaluation

$$\frac{E => nil}{\text{check}(E) => false}\texttt{Eval} - \texttt{check} - \texttt{nil}$$

$$\frac{E => v, v \neq nil}{\text{check}(E) => true}\texttt{Eval} - \texttt{check} - \texttt{other}$$

5.2 Construction

$$\frac{\Gamma| - S|E => \Gamma| - S_1|E_{expr}, E_{vars}}{\Gamma| - S|?E => \text{check}(E_{expr}), E_{vars}}\texttt{Construct} - \texttt{check}$$

6 where

6.1 Evaluation

$$\frac{e_i => v_i..., set(n_i, v_i), E => V}{\text{where}(E, < n_i, e_i > ...) => V}\texttt{Eval} - \texttt{where}$$

6.2 Construction

$$\frac{\begin{array}{c} < nf_i, \Gamma' >= fresh(n_i, \Gamma)... \\ S' = alloc(nf_i, S)... \\ E_s = E[n_i/nf_i...] \\ \Gamma'| - S'|E_s => \Gamma'| - S'_0|E_{expr}, E_{vars} \\ es_i = e_i[n_i/nf_i...]... \\ \Gamma'| - S'_{i-1}|es_i => \Gamma'| - S'_i|e_{i,expr}, e_{i,vars}... \end{array}}{\Gamma| - S \left| \begin{array}{l} E \text{ where} \\ n_i = e_i; ... \\ end \end{array} \right. => \Gamma'| - S'_n \left| \begin{array}{c} \text{where}(E_{expr}, < nf_i, e_{i,expr} > ...), \\ E_{vars} \cup e_{i,vars}... \setminus \{nf_i...\} \end{array} \right.}\texttt{Construct} - \texttt{where}$$

7 hold

TODO: how to achieve nested iteration; current theory: specify a set of streams to sample from and hold constant while the nested iteration finishes. ?