Implementação de Semântica Vaga em Notus

Autor: Felipe Silva Loredo
Orientador: Roberto da Silva Bigonha

Definições Informais

A definição informal da semântica de uma linguagem de programação é propensa a fornecer uma especificação ambígua.

Definições Formais

Utilizam o rigor matemático e sua notação formal para tratar os conceitos semânticos com maior precisão.

Semântica Denotacional

A semântica é dada pelo mapeamento das estruturas da linguagem em objetos matemáticos chamados denotações. Argumentos do mapeamento incluem informações do contexto e o resultado é a influência da construção no comportamento global do programa.

Linguagem Notus

Baseando-se na semântica denotacional, definiu-se a linguagem Notus. Trata-se de uma linguagem de domínio específico cujo propósito é prover um ambiente orientado para a definição modular de linguagens de programação.

Definições incrementais

Durante a definição denotacional de uma linguagem, é possível que algumas partes da definição já realizada possam precisar de ser redefinidas devido a necessidades de novas construções.

Transformações e Notus

Na linguagem Notus é possível realizar essas extensões de forma automatizada por meio do uso de um recurso conhecido como transformação de módulos.

Definição do comando *while*:

$$\mathcal{C}[\![\textbf{while}\ E\ C]\!]\ r = \text{FIX}\ \lambda f c.\mathcal{E}[\![E]\!]\ r;\ \lambda v.\textbf{if}\ v\ \textbf{then}\ \mathcal{C}[\![C]\!]\ r\ (f\ c)\ \textbf{else}\ c$$

Abordagem tradicional: reescrever todas as equações da função *C* e adicionar o caso do comando *break.*

Abordagem com transformadores: além da equação do *break* seria adicionada:

$$\textbf{tranformation}\ include\_break\_a$$
$$\textbf{signature}\ \mathcal{C}:\ Com \to Env \to Cc \to Cc$$
$$\textbf{to}\ Com \to Env \to (b : Cc) \to Cc \to Cc$$
$$\textbf{default}\ b\ =\ \lambda s.error$$

Transformadores existentes

Inclusão de parâmetro

$$\textbf{signature}\ f_1 : T_1\ \textbf{to}\ T_1'$$
$$\textbf{default}\ l_1 = c_1$$

Decoração de função

$$\textbf{replace}\ f_1\ p_{11}\ \cdots\ p_{1k_1}\ \textbf{by}\ e_1$$

Redefinição de função

$$\textbf{redefine}\ f_1\ p_{11}\ \cdots\ p_{1k_1}\ =\ e_1$$

Objetivo

Desenvolvimento e a adição das construções para transformação ao compilador da linguagem Notus já implementado de modo que a especificação incremental modular via transformação de módulos seja possível.

$$\mathcal{C}[\![\textbf{while } E\ C]\!]\ r = \text{FIX } \lambda f c.\mathcal{E}[\![E]\!]\ r;\ \lambda v.\textbf{if } v \textbf{ then } \mathcal{C}[\![C]\!]\ r\ (f\ c) \textbf{ else } c$$

tranformation *include_break_a*

signature $\mathcal{C}$: $Com \to Env \to Cc \to Cc$ **to** $Com \to Env \to (b : Cc) \to Cc \to Cc$

default $b = \lambda s.error$

tranformation *transformation-name*

signature $f_1 : T_1$ **to** $T_1', f_2 : T_2$ **to** $T_2', \cdots, f_m : T_m$ **to** T_m'

default $l_1 = c_1, l_2 = c_2, \cdots, l_n = c_n$

replace $f_1\ p_{11}\ \cdots\ p_{1k_1}$ **by** $e_1, \cdots, f_n\ p_{n1}\ \cdots\ p_{nk_n}$ **by** e_n

redefine $f_1\ p_{11}\ \cdots\ p_{1k_1} = e_1, \cdots, f_n\ p_{n1}\ \cdots\ p_{nk_n} = e_n$