

Ficha 3

Semântica das Linguagens de Programação

2021/22

1. Recorde a semântica de transições *small-step* para expressões aritméticas e booleanas que definiu na Ficha 2.

Queremos agora estender a linguagem de expressões aritméticas com a operação de divisão, o que levanta o problema da divisão por zero.

- (a) Adapte o sistema de transições que definiu para esta nova operação de forma a que a avaliação expressões que envolvam divisões por zero conduzam a configurações bloqueadas.
 - (b) Seguindo os mesmos princípios, defina agora o sistema de transições para os comandos.
2. Como vimos, usando a semântica natural para a linguagem **While** não conseguimos distinguir entre a situação de um programa terminar abruptamente (por exemplo pela ocorrência de um erro) da situação do programa entrar em ciclo. Uma forma de contornar este problema consiste em considerar um *estado de erro* e modelar a terminação abrupta com uma transição para esse estado de erro (incluído nas configurações terminais).

Considere, de novo, a extensão da linguagem com a operação de divisão. Apresente uma semântica natural para a interpretação das expressões e dos comandos. Neste caso, as expressões aritméticas são interpretadas em $\mathbf{Z}_\perp = \mathbf{Z} \cup \{\perp\}$, as booleanas em $\mathbf{T}_\perp = \mathbf{T} \cup \{\perp\}$ e os comandos em $\mathbf{State}_\perp = \mathbf{State} \cup \{\perp\}$.

3. Considere que acrescentamos à classe sintática das expressões aritméticas a divisão inteira

$$\mathbf{Aexp} \ni a ::= \dots \mid a_1/a_2$$

Estas novas expressões acarretam a possibilidade da ocorrência de erros de execução na avaliação das expressões aritméticas e booleanas.

- (a) Para lidar com a divisão inteira e a possibilidade de erros de execução, considere que acrescentamos à máquina abstracta **AM** a instrução DIV para o cálculo da divisão inteira.
 - i. Como definiria a semântica da máquina **AM** com mais este comando?
 - ii. Naturalmente terá que enriquecer a tradução das expressões para lidar com divisões. Indique as alterações que achar necessárias.
- (b) Calcule o código gerado pela função de tradução para o seguinte programa **While**:
$$x:=0; \text{ if } x \leq y \text{ then } a:=10/x \text{ else } a:=y/2$$

e simule a sua execução a partir do estado inicial s em que $s x = 3$ e $s y = 2$.

4. Estenda a geração de código **AM** da linguagem **While** para o comando **repeat** S **until** b que definiu na Ficha 1.
5. A máquina **AM** está ainda bastante afastada das arquitecturas mais tradicionais de uma máquina. No sentido de gradualmente tornarmos a máquina abstracta mais próxima de uma arquitectura real, as máquinas **AM₁** e **AM₂** que a seguir se apresentam, em vez de se referirem às variáveis do programa pelo seu nome, referem-se a elas pelos seus *endereços de memória*.

AM₁

- configurações: $\langle c, e, m \rangle \in \mathbf{Code} \times \mathbf{Stack} \times \mathbf{Memory}$, com $\mathbf{Memory} = \mathbf{Z}^*$.
- instruções: PUT- n e GET- n , com n um endereço (um número natural), em vez de STORE- x e FETCH- x .

AM₂

- configurações: $\langle pc, c, e, m \rangle \in \mathbf{N} \times \mathbf{Code} \times \mathbf{Stack} \times \mathbf{Memory}$, onde pc é o *program counter*.
- instruções: LABEL- l , JUMP- l e JUMPFALSE- l , sendo l uma etiqueta (um número natural), em vez de BRANCH(...,...) e LOOP(...,...).

- (a) Defina uma semântica operacional para a máquina **AM₁**.
- (b) Defina uma função de tradução de programas **While** em código **AM₁**. Vai precisar de uma função que associa a cada variável o seu endereço de memória:

$$env : \mathbf{Var} \rightarrow \mathbf{N}$$

- (c) Diga como poderia definir a função semântica \mathcal{S}_{am1} induzida pela máquina abstracta **AM₁**.
- (d) Implemente em Haskell um programa que simule a execução de código assembly da máquina **AM₁** e que implemente também a função de tradução de programas **While** em código **AM₁**.
- (e) Defina uma semântica operacional para a máquina **AM₂**. A ideia é que a instrução a ser executada é a que for apontada pelo *program counter* (pc).
 - LABEL- l apenas faz incrementar o pc .
 - JUMP- l faz com que o pc passe a ser o local onde está a instrução LABEL- l no código.
 - JUMPFALSE- l faz o pc saltar para a LABEL- l se no topo da stack estiver **ff**; se no topo da stack estiver **tt** o pc é incrementado.
- (f) Defina uma função de tradução de programas **While** em código **AM₂**. Para além de uma função que associa a cada variável o seu endereço de memória, vai precisar de um parâmetro que registre “o valor para a próxima *label*” de modo a garantir que cada etiqueta é única.
- (g) Diga como poderia definir a função semântica \mathcal{S}_{am2} induzida pela máquina abstracta **AM₂**.
- (h) Implemente em Haskell um programa que simule a execução de código assembly da máquina **AM₂** e que implemente também a função de tradução de programas **While** em código **AM₂**.