A *G-machine*

Pedro Vasconcelos

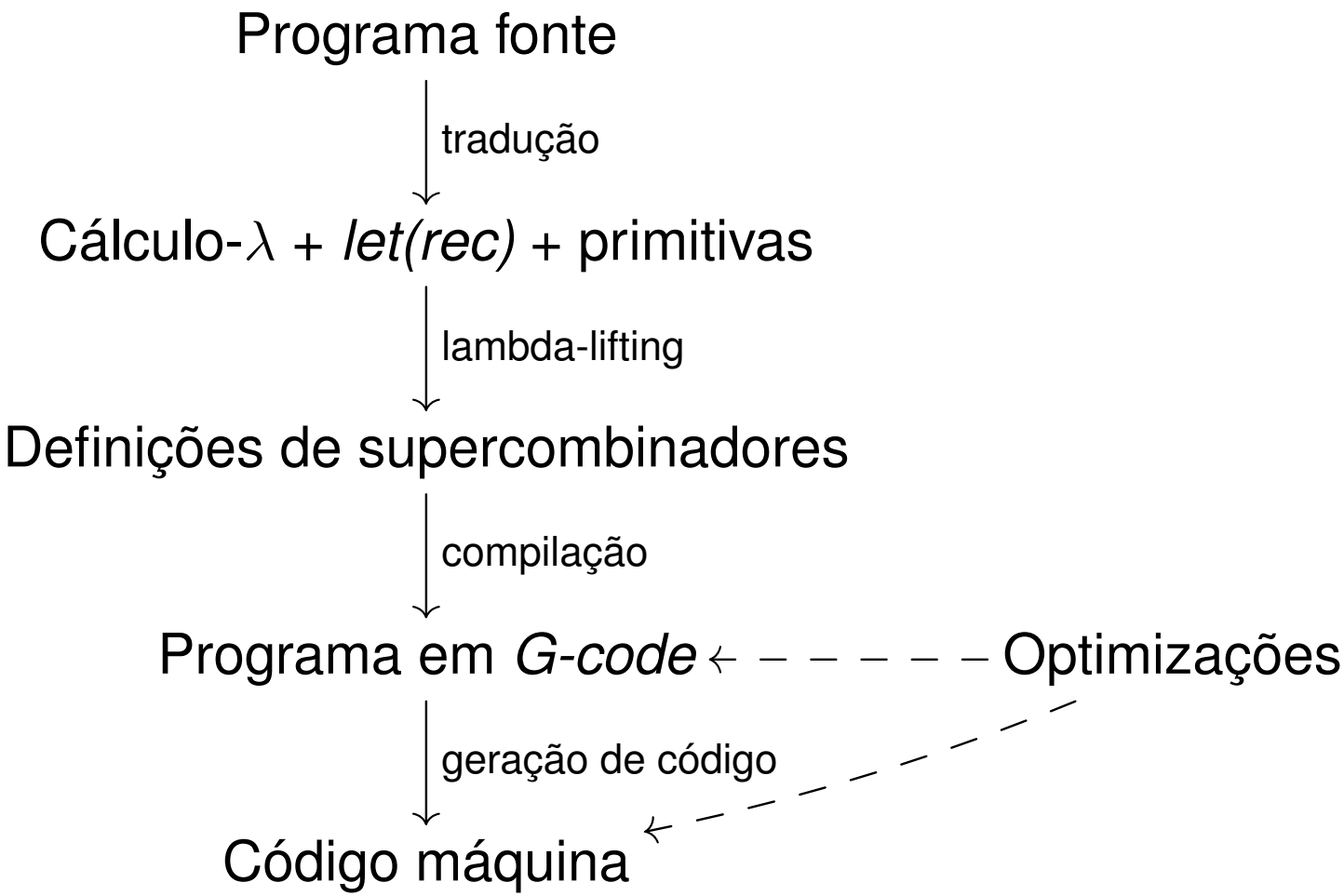
12 de Maio de 2020

G-machine

- Uma máquina abstrata para implementação de linguagens funcionais com lazy evaluation
- Executa um código intermédio (G-code) para implementar redução de grafos com supercombinadores
- Código intermédio pode ser interpretado ou traduzido para código máquina
- Possibilita muitas optimizações do código intermédio

Bibliografia: *The Implementation of Functional Programming Languages*, Simon L. Peyton Jones, Prentice-Hall International, 1987.

Visão global

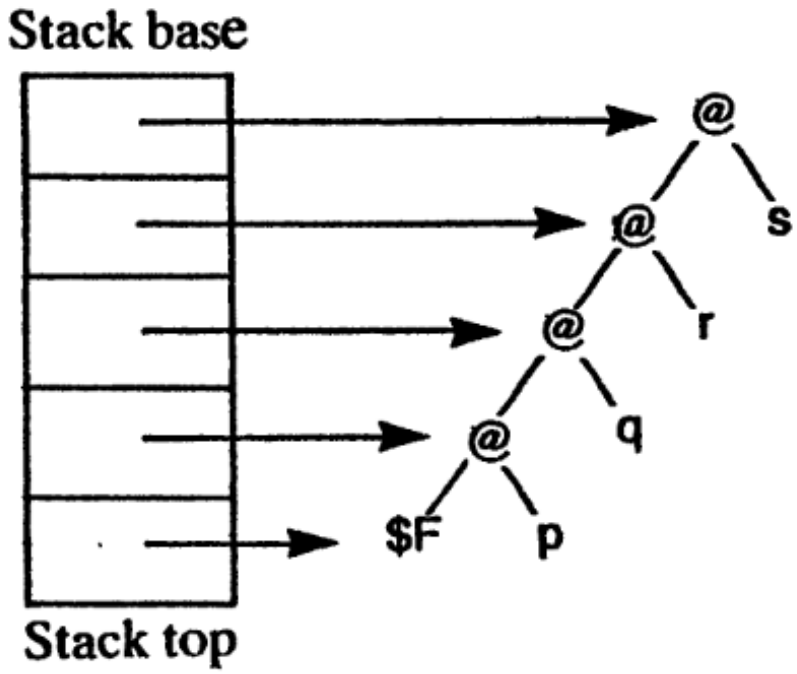


Pilha e contextos I

Tal como a SECD, a G-machine usa uma pilha para aceder aos argumentos e valores temporários.

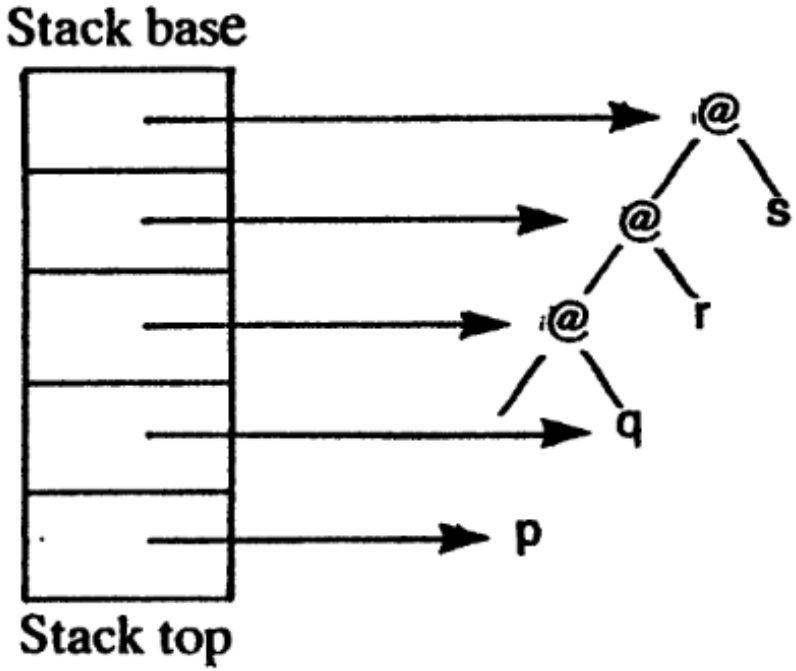
Exemplo: calcular ($*F p q r s*) em que $*F* é um supercombinador com 2 argumentos.

Pilha e contextos II



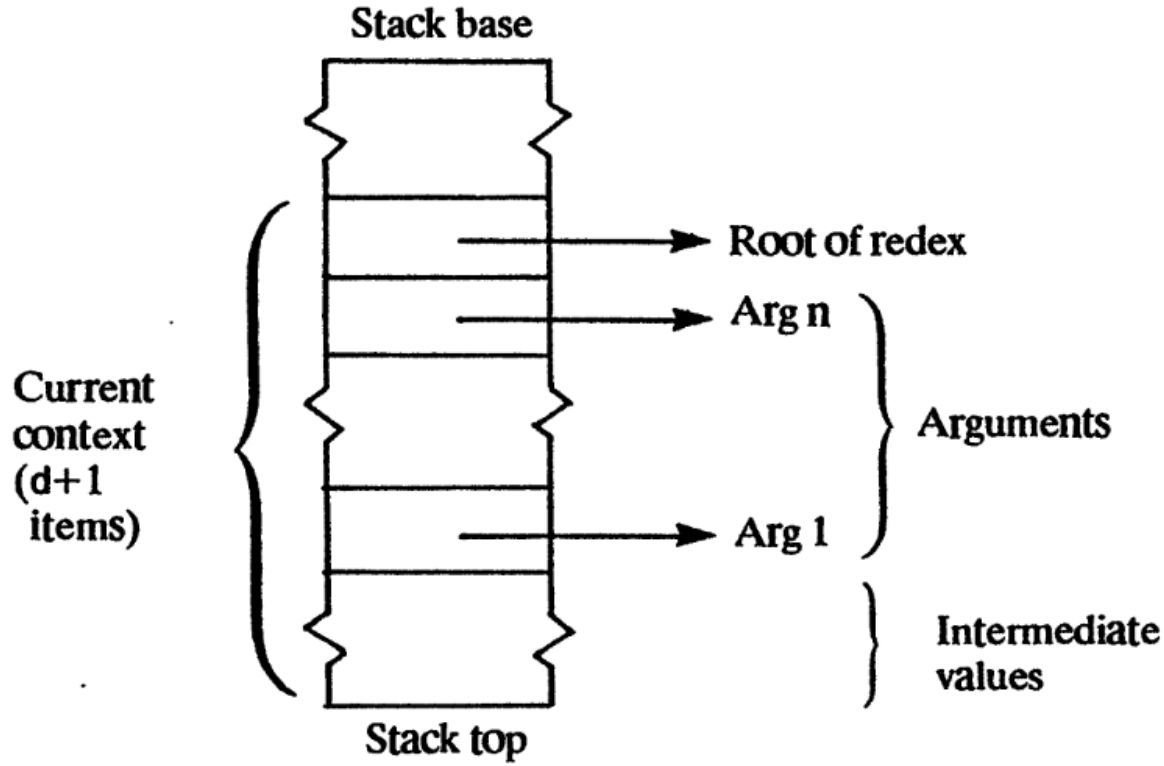
Efectuamos o *unwind* do grafo, i.e. empilhamos apontadores para a *espinha*.

Pilha e contextos III



Antes de executar $\$F$, re-organizamos os apontadores para os argumentos do supercombinador; a entrada anterior aponta a raiz do *redex* (que irá ser actualizada).

Pilha e contextos IV



Em geral: o contexto de execução de um supercombinador com *n* argumentos.

Compilação de supercombinadores

Cada supercombinador do programa é compilado para uma sequência de instruções *G-code* que implementa a re-escrita do grafo correspondente.

Como não tem variáveis livres, cada supercombinador traduz-se num procedimento global que acede aos seus argumentos apenas pela pilha.

Exemplo I

Calcular uma lista infinita $[1, 2, 3 \ldots]$.

```
from n = n : from (succ n)          --definições
succ n = n+1
-------------------------------------------------
from (succ 0)                       --expressão a calcular
```

Exemplo II

O *lambda-lifiting* é trivial (as definições já são supercombinadores):

```
$from n = CONS n ($from ($succ n))
$succ n = + n 1
$main = $from ($succ 0)
```

```
$main
```

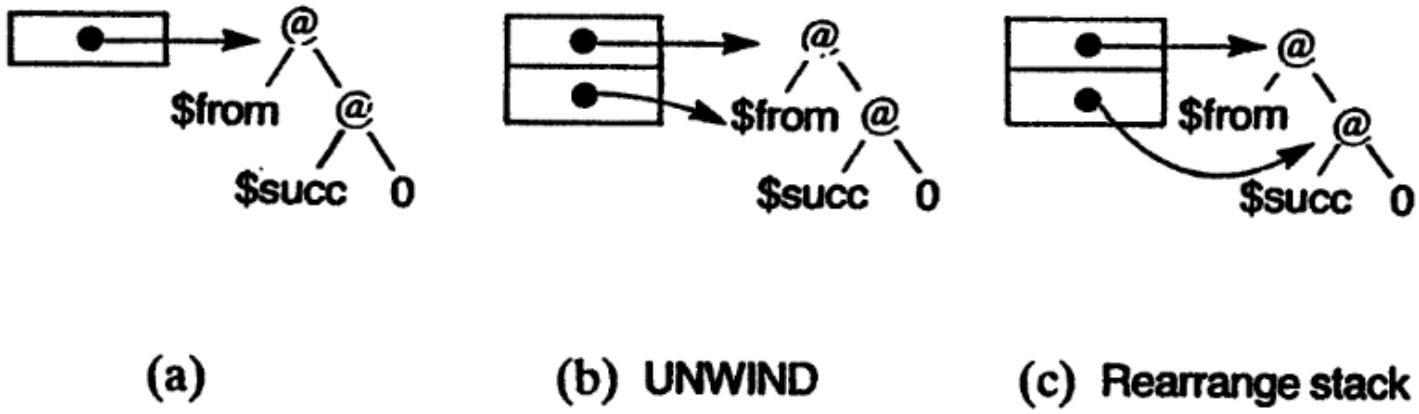
−−grafo inicial

Exemplo III

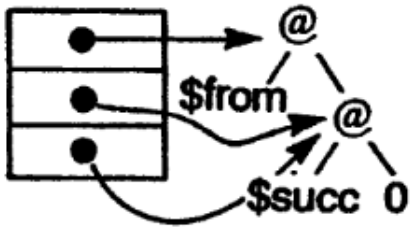
Código para o supercombinador $from:

`PUSH 0;`	empilhar n
`PUSHGLOBAL $succ;`	empilhar função $succ
`MKAP;`	construir ($succ n)
`PUSHGLOBAL $from;`	empilhar função $from
`MKAP;`	construir ($from ($succ n))
`PUSH 1;`	empilhar n
`CONS;`	construir (n : ($from ($succ n)))
`UPDATE 2;`	actualizar raiz do *redex*
`POP 1;`	remover parâmetro n
`UNWIND;`	iniciar próxima redução

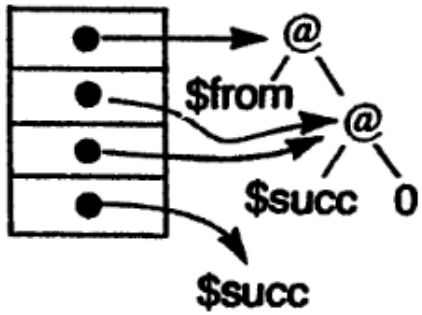
Exemplo de execução



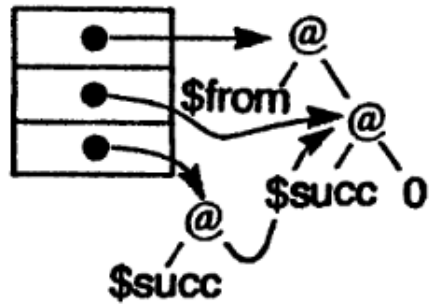
Exemplo de execução



(d) PUSH 0

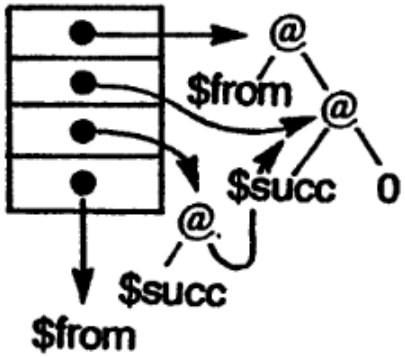


(e) PUSHGLOBAL $succ

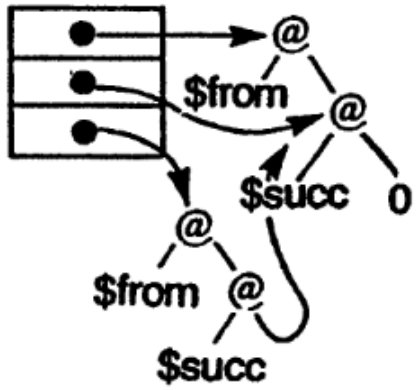


(f) MKAP

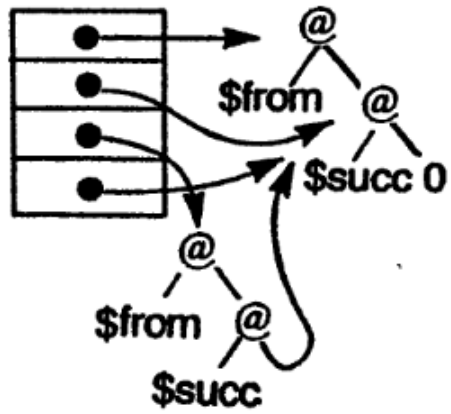
Exemplo de execução



(g) PUSHGLOBAL $from

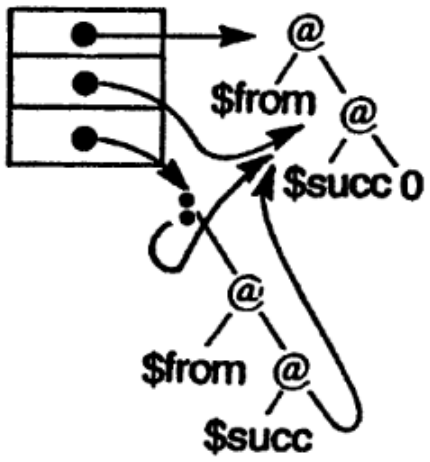


(h) MKAP

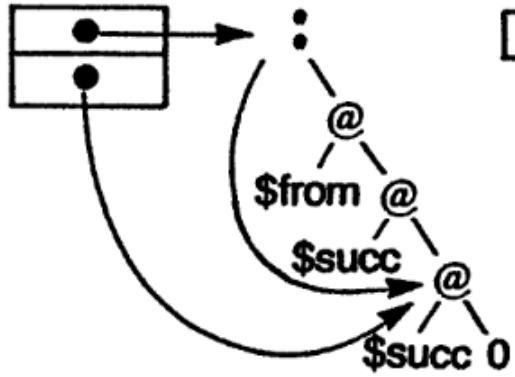


(i) PUSH 1

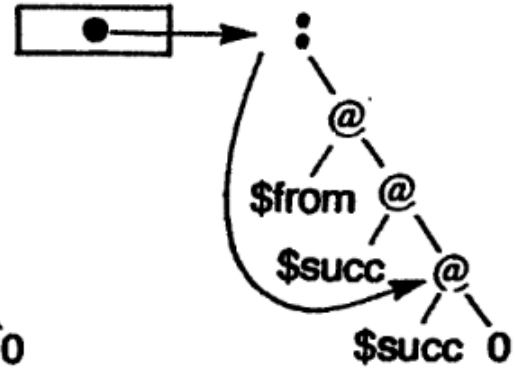
Exemplo de execução



(j) CONS



(k) UPDATE 2



(l) POP 1

Esquemas de compilação I

Três esquemas de compilação:

$F[\![-]\!]$ definição dum supercombinador

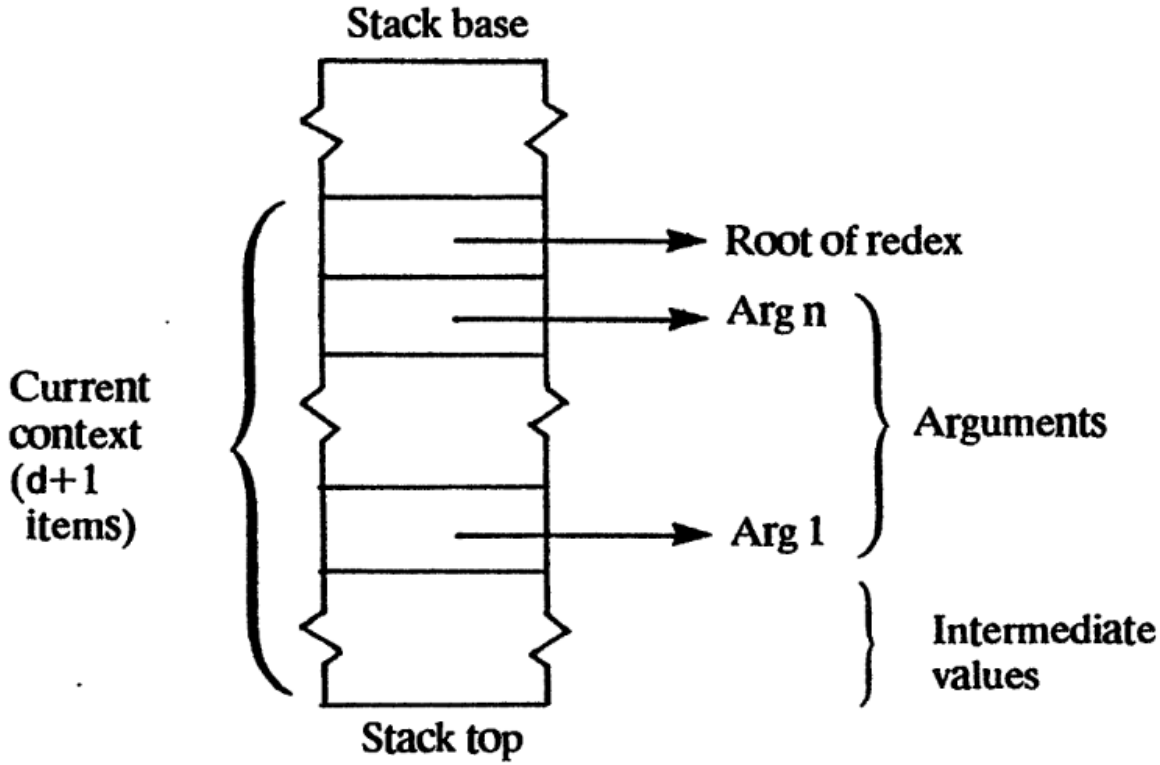
$R[\![-]\!]\ \rho\ d$ aplicação dum supercombinador

$C[\![-]\!]\ \rho\ d$ construir o grafo duma expressão

onde:

- ρ associa *identificadores* a *offsets* na pilha;
- $d + 1$ é a *profundidade* do contexto actual (ver diagrama seguinte)

Esquemas de compilação II



Esquemas F e R

$$F[\![f\ x_1\ x_2\ \ldots\ x_n = E]\!] =$$
$$\text{GLOBSTART}\ f, n; R[\![E]\!]\ [x_1 \mapsto n, x_2 \mapsto n-1, \ldots, x_n \mapsto 1]\ n$$

$$R[\![E]\!]\ \rho\ d = C[\![E]\!]\ \rho\ d; \text{UPDATE}\ (d+1); \text{POP}\ d; \text{UNWIND}$$

Esquema C: constantes e identificadores

$$C[\![i]\!]\ \rho\ d = \text{PUSHINT}\ i$$

$$C[\![f]\!]\ \rho\ d = \text{PUSHGLOBAL}\ f$$

$$C[\![x]\!]\ \rho\ d = \text{PUSH}\ (d - \rho_x)$$

Esquema C: aplicações

$$C[\![E_1\ E_2]\!]\ \rho\ d = C[\![E_2]\!]\ \rho\ d; C[\![E_1]\!]\ \rho\ (d+1); \mathsf{MKAP}$$

Esquema C: expressões *let*

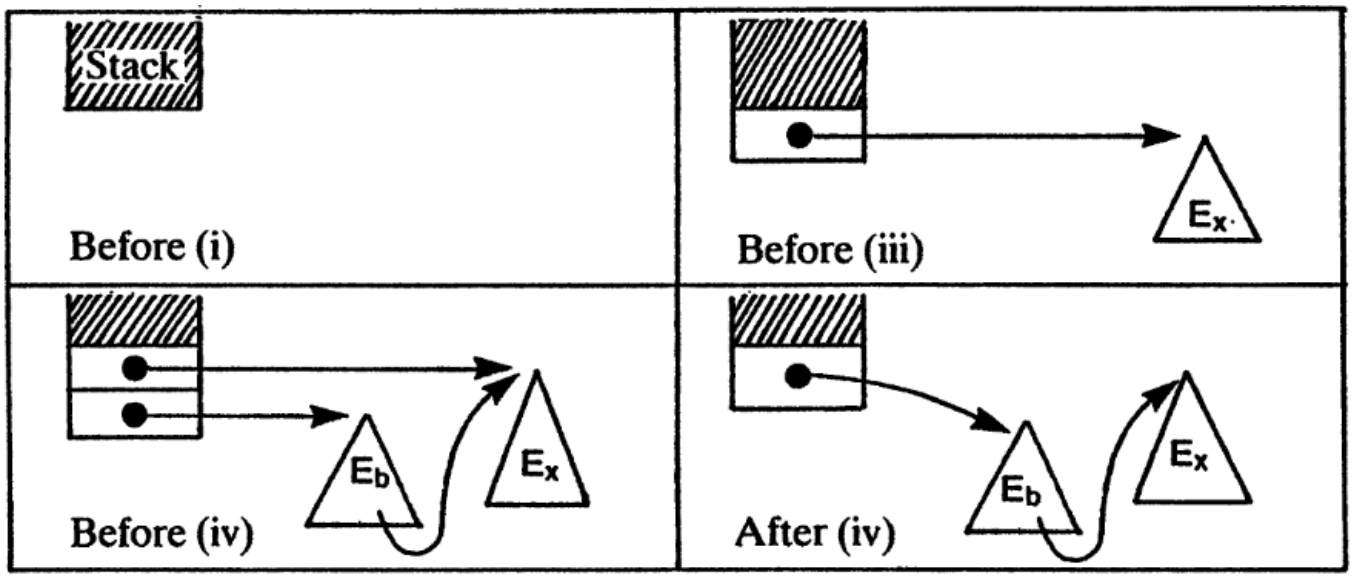
$$C[\![\text{let } x = E_x \text{ in} E_b]\!]\ \rho\ d =$$
$$C[\![E_x]\!]\ \rho\ d;\ C[\![E_b]\!]\ \rho[x \mapsto d+1]\ (d+1);\ \text{SLIDE 1}$$

(i) Construir E_x

(ii) Extender p com $x \mapsto d+1$

(iii) Construir E_b

(iv) Deslizar o valor no topo da pilha uma posição



Esquema C: expressões *letrec* I

$$\text{letrec } x_1 = E_1; x_2 = E_2; \ldots; x_n = E_n \text{ in } E_b$$

(i) Alocar n nós vazios (um por definição)

(ii) Extender ρ com $x_i \mapsto d + i$

(iii) Para cada definição:

  (a) construir E_i;

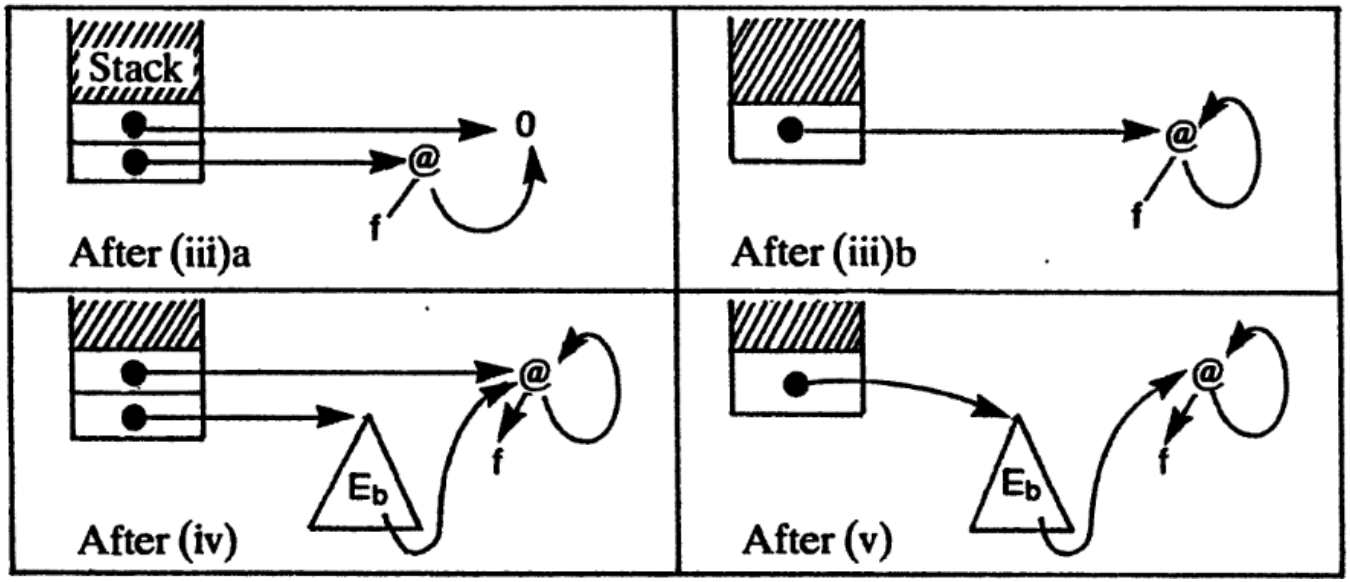
  (b) actualizar o nó vazio correspondente (UPDATE)

(iv) Construir E_b

(v) Deslizar o topo da pilha n argumentos

(Regra de compilação: ver a bibliografia.)

Esquema C: expressões *letrec* II



Exemplo: execução de letrec $x = f\ x$ in E_b.