Alguns Problemas de Teoria da Computação

Nelma Moreira
Departamento de Ciência de Computadores
Faculdade de Ciências, Universidade do Porto
email: nam@ncc.up.pt

1997

Conteúdo

Capítulo 1

λ–Calculus Puro

Conceitos: λ-termo; Variáveis ligadas e livres; Substituição; Reduções α,β e η e redexes; Forma normais e teoremas de Church-Rosser; Expressividade do λ-calculus ; Teorema do Ponto Fixo.

Bibliografia

- Sabine Broda, *O Lambda Calculus Puro*, CIUP, 1993.
- H.P Barendregt, *Functional Programming and Lambda Calculus*, em *Handbook of Theoretical Computer Science* Vol B, Elsevier, 1990.

1. Para cada uma das expressões seguintes:

   - determine as que correspondem a um λ-termo.
   - para cada uma que for um λ-termo "parentize"e indique o conjunto das variáveis livres e o conjunto das variáveis ligadas.

   (a) $xyxx$

   (b) $\lambda x.\lambda x.x$

   (c) $y\lambda x.$

   (d) $x\lambda x.\lambda y.xy$

   (e) $xy\lambda x.x\lambda.x.yxx$

   (f) $\lambda x.xy(\lambda y.xy)$

2. Sendo M e N dois λ-termos, $M \equiv N$ se são o mesmo λ-termo ou se podem ser obtidos um do outro por mudança de variáveis ligadas em M ou em N. Por exemplo $(\lambda x.x)z \equiv (\lambda y.y)z$ e $(\lambda x.x)z \not\equiv (\lambda x.y)z$. Prove usando indução na estrutura dum λ-termo M que se $x \not\equiv y$ e se $x \notin FV(L)$ então:

$$M[N/x][L/y] \equiv M[L/y][N[L/y]/x]$$

3. Para cada um dos seguintes λ-termos reduza a forma normal β se possível:

   (a) xy

   (b) $(\lambda x.x)(yz)$

   (c) $(\lambda x.xy)(\lambda z.z)$

   (d) $(\lambda xy.xy)$

   (e) $(\lambda x.x\lambda x.xy)(yy)$

   (f) $(\lambda x.xx)(\lambda x.xx)$

   (g) $(\lambda x.xxx)(\lambda x.xxx)$

   (h) $(\lambda xy.y)((\lambda x.xx)(\lambda x.xx))b$

   (i) $M \equiv AAx$ onde $A \equiv \lambda axz.z(aax)$

4. O que está errado nas seguintes reduções?

   (a) $(\lambda xy.yx)y \rightarrow_\beta \lambda y.yy$

   (b) $(\lambda xx.xx)y \rightarrow_\beta \lambda x.yy$

   (c) $\lambda x.bxbx \rightarrow_\eta bxb$

   (d) $(\lambda x.xx)(\lambda y.ay)k \rightarrow_\beta (\lambda x.xx)ak \rightarrow_\beta akak$

5. Sendo $S \equiv \lambda xyz.xz(yz)$, $K \equiv \lambda xy.x$, $B \equiv \lambda xyz.x(yz)$ e $I \equiv \lambda x.x$ mostre que

   (a) $S(KI) =_{\beta\eta} I$

   (b) $BI =_{\beta\eta} I$

   (c) $SKK =_{\beta\eta} I$

   (d) $S(KS)K =_{\beta\eta} B$

6. Sendo $T1 \equiv (\lambda x.xx)(\lambda x.xx)$ e $T2 \equiv (\lambda xy.yx)(\lambda x.xx)(\lambda x.x)$ diga se $T1$ e $T2$ se podem reduzir a um mesmo λ-termo T.

1.1 Representação de Inteiros em λ-Cálculo: numerais de Church

Um modo de representar o inteiro $n \geq 0$ (os $\underline{n}$ são chamados numerais de Church) no λ-cálculo é através do λ-termo

$$\underline{n} \equiv \lambda f.\lambda x.\overbrace{f(f(f\cdots f(x)\cdots))}^{n\ f\text{'s}}$$

(a representação de n é designada por $\underline{n}$). Temos, por exemplo, $\underline{0} \equiv \lambda f.\lambda x.x$.
A representação de n é o λ-termo que aplica n vezes o seu primeiro argumento (f) ao segundo argumento (x).

1. **Sucessor.** Seja

$$\mathtt{succ} \equiv \lambda n.\lambda x.\lambda y.x(nxy)$$

Verifica que

$$\mathtt{succ}\,\underline{1} \to^*_\beta \underline{2}$$

Verifica que, em geral

$$\mathtt{succ}\,\underline{n} \to^*_\beta \underline{n+1}$$

2. **Soma.** Seja

$$\mathtt{sum} \equiv \lambda m.\lambda n.m\,\mathtt{succ}\,n$$

Verifica que

$$\mathtt{sum}\underline{1}\,\underline{2} \to^*_\beta \underline{3}$$

Verifica que, em geral

$$\mathtt{sum}\underline{m}\,\underline{n} \to^*_\beta \underline{m+n}$$

3. **Valores lógicos.** Sejam os valores lógicos representados por

$$\begin{aligned}\mathtt{true} &\equiv \lambda x.\lambda y.x\\ \mathtt{false} &\equiv \lambda x.\lambda y.y\end{aligned}$$

e sejam os λ-termos

$$\begin{aligned}\mathtt{if} &\equiv \lambda c.\lambda x.\lambda y.cxy\\ \mathtt{zerop} &\equiv \lambda n.n(\lambda x.\mathtt{false})\,\mathtt{true}\end{aligned}$$

   (a) Mostra que, para todos os λ-termos C, M e N, se verifica

$$\mathtt{if}\,CMN \to^*_\beta \begin{cases} M & \mathtt{if}\ C \to^*_\beta \mathtt{true}\\ N & \mathtt{if}\ C \to^*_\beta \mathtt{false}\end{cases}$$

   Indica uma caracterização análoga de `zerop`.

   (b) Indique λ-termos correspondentes aos operadores lógicos "não", "e" e "ou"

4. **Pares.** Considera

$$\begin{aligned}\mathtt{pair} &\equiv \lambda x.\lambda y.\lambda z.(zxy)\\ \mathtt{fst} &\equiv \lambda x.\lambda y.x\\ \mathtt{snd} &\equiv \lambda x.\lambda y.y\end{aligned}$$

Mostra que para todos os λ-termos M e N é

$$\begin{aligned}(\texttt{pair}\ M\ N)\texttt{fst} &\to_\beta^* M\\ (\texttt{pair}\ M\ N)\texttt{snd} &\to_\beta^* N\end{aligned}$$

5. $*$ **Predecessor.** Determina um λ-termo `pred` tal que

$$\texttt{pred}\ \underline{n} \to_\beta^* \begin{cases} \underline{0} & \texttt{if } n = 0\\ \underline{n-1} & \texttt{if } n \geq 1\end{cases}$$

6. $*$ **Multiplicação.** Determina um λ-termo $Mult$ tal que

$$\texttt{mult}\ \underline{m}\,\underline{n} \to_\beta^* \underline{mn}$$

7. Sendo $\texttt{true} \equiv \lambda xy.x$, $\texttt{false} \equiv \lambda xy.y$ e $[M,N] \equiv \lambda z.zMN$

   (a) verifique que $[M,N]\texttt{true} \equiv M$ e $[M,N]\texttt{false} \equiv N$

   (b) sendo $\overline{0} \equiv I$, $\overline{n+1} \equiv [\texttt{false}, \overline{n}]$. Determine λ-termos `suc`, `pred` e `zero` tal que

$$\texttt{suc}\,\overline{n} =_\beta \overline{n+1}$$

$$\texttt{pred}\,\overline{n+1} =_\beta \overline{n}$$

$$\texttt{zero}\,\overline{0} =_\beta \texttt{true}$$

$$\texttt{zero}\,\overline{n+1} =_\beta \texttt{false}$$

   (c) Sendo $\underline{n} \equiv \lambda fx.f^n(x)$ os numerais de Church prove que existem λ-termos T e TI tal que $T\underline{n} =_\beta \overline{n}$ e $TI\overline{n} =_\beta \underline{n}$.

1.2 Algumas notas sobre a resolução dos exercícios

1. **Sucessor.** O λ-termo $\underline{n}$ aplica n vezes o primeiro argumento ao segundo. Após 2 reduções β temos $\underline{n}fx \rightarrow_\beta^* f^n x$. Para obter $f^{n+1}x$ aplicamos f àquele termo. Para termos $\underline{n+1}$ fazemos abstrações em x e f e, finalmente, para obter uma função aplicável a $\underline{n}$ fazemos nove abstração. Chegamos à forma de `succ` indicada e temos, como dissemos

$$\begin{array}{rcl}(\lambda n.\lambda f.\lambda x.f(nfx))\underline{n} & \rightarrow_\beta & \lambda f.\lambda x.f(\underline{n}fx)) \\ & \rightarrow_\beta & \lambda f.\lambda x.f(f^n x) \\ & = & \lambda f.\lambda x.f^{n+1}x \\ & = & \underline{n+1}\end{array}$$

   Exemplo pedido:

$$\begin{array}{rcl}(\lambda n.\lambda f.\lambda x.f(nfx))\underline{1} & \rightarrow_\beta & \lambda f.\lambda x.f((\lambda f.\lambda x.f(x))fx) \\ & \rightarrow_\beta & \lambda f.\lambda x.f(f(x)) \\ & = & \underline{2}\end{array}$$

2. **Soma.** A redução de m `succ` aplicada a n aplica m vezes a função `succ` — isto é, soma m — ao argumento n. Para obter `sum` temos que abstrair em m e n.

3. **Valores lógicos.**

   (a) Verificações simples. Por exemplo, suponhamos que $C \rightarrow_\beta^*$ `true`. Temos

$$\begin{array}{rcl}\texttt{if}\, CMN & \rightarrow_\beta^* & (\lambda c.\lambda x.\lambda y.cxy)(\lambda x.\lambda y.x)MN \\ & \rightarrow_\beta^* & (\lambda x.\lambda y.x)MN \\ & \rightarrow_\beta^* & M\end{array}$$

   (b) Considere-se $Nao \equiv (\lambda w.w(\texttt{false})(\texttt{true}))$, $E \equiv \lambda wz.wz\texttt{false}$ e $Ou \equiv \lambda wz.w(\texttt{true})z$.

4. **Pares.** Considerando, por exemplo, o primeiro caso temos

$$\begin{array}{rcl}(\texttt{pair}\, M\, N)\texttt{fst} & = & ((\lambda x.\lambda y.\lambda z.(zxy))M\, N)(\lambda x.\lambda y.x) \\ & \rightarrow_\beta^* & (\lambda z.(zMN))(\lambda x.\lambda y.x) \\ & \rightarrow_\beta & (\lambda x.\lambda y.x)M\, N \\ & \rightarrow_\beta & (\lambda y.M)N \\ & \rightarrow_\beta & M\end{array}$$

5. $*$ **Predecessor.** Seja

$$\texttt{pred} = \lambda n.(\overbrace{n}^{(a)}(\lambda p.\overbrace{\texttt{pair}(\texttt{succ}(p\,\texttt{fst}))(p\,\texttt{fst})}^{(b)})\overbrace{(\texttt{pair}\,\underline{0}\,\underline{0})}^{(c)})\overbrace{\texttt{snd}}^{(d)}$$

onde utilizamos notações de exercícios anteriores. O λ-termo `pred` corresponde à função que aplica n vezes (a) a função que efectua a transformação (b)

$$(p_1, p_2) \rightarrow (p_1 + 1, p_1)$$

ao par inicial $(\underline{0}, \underline{0})$ (c) e toma a segunda parte (d) do par resultante.

Obtemos assim (supondo $n \geq 1$)

$$(\underline{0}, \underline{0}) \rightarrow (\underline{1}, \underline{0}) \rightarrow (\underline{2}, \underline{1}) \rightarrow \cdots \rightarrow (\underline{n}, \underline{n-1}) \ (n \text{ reduções})$$

e $(\underline{0}, \underline{0})$ se $n = 0$. Ao tomar a segunda parte temos o resultado pretendido: $n - 1$ se $n \geq 1$ e 0 se $n = 0$.

6. $*$ **Multiplicação.** Indicamos apenas uma solução

$$\texttt{mult} = \lambda n.\lambda m.\lambda x.(n(mx))$$

7. **Numerais $\overline{n}$ do exercício 10**

(a)

$$\texttt{suc} \equiv \lambda x.[\texttt{false}, x] \quad \texttt{pred} \equiv \lambda x.x\texttt{false} \quad \texttt{zero} \equiv \lambda x.x\texttt{true}$$

Note que $\overline{n} \equiv \overbrace{\texttt{sucsuc}\ldots\texttt{suc}}^{n}\overline{0}$

(b)

$$T \equiv \lambda x.x\,\texttt{suc}\,\overline{0}$$

$$T I \overline{0} \equiv c_0 \quad T I \overline{n} \equiv \texttt{succ}(T I\ \texttt{pred}\,\overline{n})$$

$$\text{donde} \quad T I \equiv \lambda x.\texttt{if}(\texttt{zero}\ x)c_0(\texttt{succ}(T I\ (\texttt{pred}\,x))$$

1.3 Recursividade

Teorema 1 Teorema do Ponto Fixo

1. *$\forall F \exists X\ FX =_\beta X$*
2. *Existe um λ-termo fechado $Y \equiv \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ tal que*

$$\forall F\ F(YF) =_\beta YF$$

Y diz-se um operador de ponto fixo.

Demonstração 1 *Seja*

1. *Defina $W \equiv \lambda x.F(xx)$ e $X \equiv WW$. Então $X \equiv WW \equiv (\lambda x.F(xx))W \equiv F(WW) \equiv FX$.*
2. *Pela demostração de (1).*

Exercício 1 *Sendo $S \equiv \lambda xyz.xz(yz)$ mostre que $\exists F\ \forall X\ FX =_\beta SFX$.*

Qualquer definição recursiva pode ser transformada numa equação do tipo $x = F(x)$ em que x não ocorre livre em F. A solução desta equação é dada por $Y(F)$ (ponto fixo de F).

Exercício 2 *Mostre que para qualquer λ-termo F existe um λ-termo X tal que $X =_\beta FX$.*

Exercício 3 *Usando a representação de números inteiros como λ-termos, λ-termos definidos nos exercí cios anteriores e o operador de ponto fixo Y, defina λ-termos correspondentes às seguintes funções definidas nos inteiros, eliminando a recursividade:*

1. *A função factorial*

$$\begin{aligned} fact(0) &= 1 \\ fact(n) &= n \times fact(n-1) \quad \forall n\ n \geq 1 \end{aligned}$$

Sugestão: Considere

$$fact \equiv Y(\lambda f.(\lambda n.\mathtt{if}(\mathtt{zerop}\ n)\underline{1}(\mathtt{mult}\ n(f(\mathtt{pred}\ n)))))$$

Trata-se do ponto fixo do funcional F cujo valor é

$$F(f,n) \equiv \begin{cases} \underline{1} & \text{se } n \text{ representa } 0 \\ \mathtt{mult}\ n(f(\mathtt{pred}\ n)) & \text{se } n \text{ representa um inteiro} \geq 1 \end{cases}$$

2. *A adição pode ser definida recursivamente do seguinte modo, para n e m inteiros:*

$$\begin{aligned} soma(n,0) &= n \\ soma(n,m+1) &= soma(n,m)+1 \end{aligned}$$

3. *A função de Ackermann*

$$\begin{aligned} Ack(0,n) &= n+1 \\ Ack(m+1,0) &= Ack(m,1) \\ Ack(m+1,n+1) &= Ack(m,Ack(m+1,n)) \end{aligned}$$

Nota: O operador Y transforma qualquer definição recursiva em iterativa.

Exercício 4 *Encontre λ-termos fechados F tal que:*

1. $Fx =_\beta xI$
2. $Fxy =_\beta xIy$
3. $Fx =_\beta F$
4. $Fx =_\beta xF$

Exercício 5 *Seja $f \circ g \equiv \lambda x.f(gx)$ e $H \equiv \lambda x.xx$.*

1. *Mostre que $H(f \circ H) =_\beta f(H(f \circ H))$.*
2. *Sendo $Y_1 \equiv \lambda f.H(f \circ H)$ e $G \equiv \lambda yf.f(yf)$ mostre que*
   (a) $Y_1x =_\beta x(Y_1x)$
   (b) $GY_1 =_\beta Y_1$
3. *Sendo $Y_{n+1} \equiv Y_nG$ prove por indução sobre n que, para $n \geq 1$,*
   (a) $Y_nx =_\beta x(Y_nx)$
   (b) $GY_n =_\beta Y_n$

Exercício 6 *Mostre que existe um λ-termo fechado M tal que*

$$\forall N \; MN =_\beta MM$$

Sendo $G \equiv \lambda xy.y(xy)$ então

$$\forall M \; \forall F \; M =_{\beta\eta} GM \iff MF =_{\beta\eta} F(MF)$$

Diga qual o significado do segundo membro desta equivalência.

Exercício 7 *Obtenha um λ-termo* `bot` *que*

$$\forall N \; \texttt{bot}\, N =_\beta \texttt{bot}$$

1.4 Funções Computáveis

Definição 1 *1. Uma função numérica é uma aplicação $f : N^p \longrightarrow N$ para algum $p > 0$.*

2. Uma função f com p argumentos é λ-definível se existe um λ-termo fechado F tal que

$$F\underline{n_1}\ldots\underline{n_p} =_\beta \underline{f(n_1,\ldots,n_p)}$$

para todo $n_1 \ldots n_p \in N$.

Definição 2 *1. As* funções iniciais *são as funções numéricas U_n^i, S^+ e Z definidas por:*

$$U_n^i(x_1\ldots,x_n) = x_i$$
$$S^+(n) = n+1$$
$$Z(n) = 0$$

2. Seja $P(n)$ uma relação numérica. $\mu m[P(m)]$ denota o menor inteiro m tal que $P(m)$ se verifica; caso contrário não está definida.

Definição 3 *Seja $\mathcal{A}$ uma classe de funções numéricas.*

1. $\mathcal{A}$ é fechada para a composição se para todo o f definido por

$$f(\bar{n}) = g(h_1(\bar{n}),\ldots,h_m(\bar{n})) \text{ com } g,h_1,\ldots h_m \in \mathcal{A}$$

se tem que $f \in \mathcal{A}$.

2. $\mathcal{A}$ é fechada para a recursão primitiva se, para todo f definido por

$$f(0,\bar{n}) = g(\bar{n}) \quad f(k+1,\bar{n}) = h(f(k,\bar{n}),k,\bar{n}) \text{ com } g,h \in \mathcal{A}$$

se tem que $f \in \mathcal{A}$.

3. $\mathcal{A}$ é fechada para a minimalização se, para todo f definido por

$$f(\bar{n}) = \mu m[g(\bar{n},m) = 0] \text{ com } g \in \mathcal{A} \text{ tal que } \forall\bar{n}\exists m g(\bar{n},m) = 0$$

se tem que $f \in \mathcal{A}$.

Definição 4 *A classe $\mathcal{R}$ das funções recursivas é a menor classe das funções numéricas que contem todas as funções iniciais e é fechada para composição, recursão primitiva e minimalização.*

Lema 1 *As funções iniciais são λ-definíveis.*

Lema 2 *1. As funções λ-definíveis são fechadas para a composição.*

2. As funções λ-definíveis são fechadas para a recursão primitiva.

3. As funções λ-definíveis são fechadas para a minimalização.

Teorema 2 *Toda a função recursiva é λ-definível.*

1. Demonstra o teorema anterior.

Demonstração 2 *Basta demonstrar os lemas 3 e 4. O lema 3 é imediato. Para o lema 4 temos:*

1. *Sejam G e H_i os λ-termos para g e h_i e considerar*

$$F \equiv \lambda\bar{x}.G(H_1\bar{x})\ldots G(H_m\bar{x})$$

2. *Sejam g e h λ-definíveis por G e H é fácil ver que F verifica:*

$$Fx\bar{y} \equiv \texttt{if } (\texttt{zerop}x) \textit{ then } G\bar{y} \textit{ else } H(F(\texttt{pred}x)\bar{y})(\texttt{pred}x)\bar{y}$$

Tal F pode ser obtido pela aplicação do teorema do ponto fixo.

3. *Mais uma vez pelo teorema do ponto fixo existe H tal que*

$$H\bar{x}y \equiv \texttt{if } (\texttt{zerop}(G\bar{x}\ y)) \textit{ then } y \textit{ else } H\bar{x}(\texttt{succ}\ y)$$

Seja $F \equiv \lambda\bar{x}.H\bar{x}\underline{0}$.

1.5 Representação de listas

Usando os λ-termos`pair`, `fst`, `snd` e `bot` podemos implementar as operações sobre listas. Para representar a lista vazia considere

$$\texttt{nil} \equiv \texttt{pair true bot}$$

a lista

$$[M] \equiv \texttt{pair false } (\texttt{pair } M \texttt{ nil})$$

e

$$[M1, M2] \equiv \texttt{pair false } (\texttt{pair } M1\ [M2])$$

...

Exercício 8 *1. Verifique que sendo* `null` $\equiv$ *true*

$$null\ M$$

Defina λ-termos para as funções `head`, `tail`, `cons`, `append` *e* `reverse`.

Capítulo 2

Combinadores

Vamos permitir o uso de constantes nos λ-termos. Seja $\mathcal{C} = \{c, c', \ldots\}$ um conjunto de constantes.

Definição 5 *Dado $\mathcal{V}$ um conjunto de variáveis, o conjunto de λ-termos com constantes, $\Lambda(\mathcal{C})$, é definido por:*

$$\begin{aligned} c \in \mathcal{C} &\Rightarrow c \in \Lambda(\mathcal{C}) \\ x \in \mathcal{V} &\Rightarrow x \in \Lambda(\mathcal{C}) \\ M, N \in \Lambda(\mathcal{C}) &\Rightarrow (MN) \in \Lambda(\mathcal{C}) \\ M \in \Lambda(\mathcal{C}), x \in \mathcal{V} &\Rightarrow (\lambda x.M) \in \Lambda(\mathcal{C}) \end{aligned}$$

Definição 6 *O conjunto de* termos combinatórios *com constantes em $\mathcal{C}$, $\mathcal{CL}(\mathcal{C})$ é definido por:*

$$\begin{aligned} c \in \mathcal{C} &\Rightarrow c \in \mathcal{CL}(\mathcal{C}) \\ x \in \mathcal{V} &\Rightarrow x \in \mathcal{CL}(\mathcal{C}) \\ P, Q \in \mathcal{CL}(\mathcal{C}) &\Rightarrow (PQ) \in \mathcal{CL}(\mathcal{C}) \end{aligned}$$

Vamos estar interessados em constantes, *combinadores*, que correspondam a λ-termos fechados. A cada *combinador* associa-se uma regra de redução $\mapsto_w$.

Definição 7 Sistema de combinadores de Curry *Considerem-se três constantes I,K e S. No conjunto $\mathcal{CL}(I, K, S)$ define-se um a relação de redução $\mapsto_w^\star$ pelas seguintes regras de contracção:*

$$\begin{aligned} IP &\mapsto_w P \\ KPQ &\mapsto_w P \\ SPQR &\mapsto_w PR(QR) \end{aligned}$$

$\mapsto_w^\star$ *é o fecho reflexivo e transitivo de* $\mapsto_w$ *e* $=_w$ *é o fecho simétrico de* $\mapsto_w^\star$.

Por exemplo,

$$\begin{aligned} SKKP &\mapsto_w^\star P \\ SIIP &\mapsto_w PP \\ KIxy &=_w SKxy\ (=_w\ y) \end{aligned}$$

Existe uma estreita analogia entre $\mapsto_w$ e a redução β para os λ-termos.

Exercício 9 Sejam $I = \lambda x.x$, $K = \lambda xy.x$, $S = \lambda xyz.xz(yz)$, $B = \lambda xyz.x(yz)$ mostre que:

1. Mostre que para todo o λ-termo M, $IM =_\beta M$, $KMN =_\beta M$ e $SMNL =_\beta ML(NL)$
2. Mostre que:$S(KI) =_{\beta\eta} I$, $BI =_{\beta\eta} I$, $SKK =_{\beta\eta} I$ e $S(KS)K =_{\beta\eta} B$.

As noções de w-redex e forma normal w podem ser definidas analogamente. O teorema de Church-Rosser é igualmente válido.

Exercício 10 Define as noções referidas no parágrafo anterior para as reduções $\mapsto_w$.

Exercício 11 Prove que se $P =_w P'$ e $Q =_w Q'$ então $P[Q/x] =_w P'[Q'/x]$.

No sistema $\mathcal{CL}(I,K,S)$pode-se "imitar" abstração.

Definição 8 Algoritmo de abstração *Define-se* $\lambda^\star x.P \in \mathcal{CL}(I,K,S)$*por indução na estrutura de* $P \in \mathcal{CL}(I,K,S)$

$$\begin{aligned} \lambda^\star x.x &\equiv I \\ \lambda^\star x.P &\equiv KP \text{ se } x \notin FV(P) \\ \lambda^\star x.PQ &\equiv S(\lambda^\star x.P)(\lambda^\star x.Q) \text{ caso contrário} \end{aligned}$$

Proposição 1 *1.* $FV(\lambda^\star x.P) = FV(P) - \{x\}$

2. $(\lambda^\star x.P)x =_w P$

3. $(\lambda^\star x.P)Q =_w P[Q/x]$, *isto é,* $=_w$ *comporta-se como* $=_\beta$

Exercício 12 Demonstre a proposição anterior. (Use indução na estrutura de P.)

Definição 9 *Considere as aplicações:*

$$\mathcal{CL} : \lambda\text{-termos} \longrightarrow \mathcal{CL}(I,K,S) \quad e \quad \lambda : \mathcal{CL}(I,K,S)] ; \longrightarrow \lambda\text{-termos}$$

$$\begin{array}{lcl} \mathcal{CL}(x) & = & x \\ \mathcal{CL}(MN) & = & \mathcal{CL}(M)\mathcal{CL}(N) \\ \mathcal{CL}(\lambda x.M) & = & \lambda^\star x.\mathcal{CL}(M) \end{array} \qquad \begin{array}{lcl} \lambda(x) & = & x \\ \lambda(MN) & = & \lambda(M)\lambda(N) \\ \lambda(I) & = & I \\ \lambda(K) & = & K \\ \lambda(S) & = & S \end{array}$$

Considerando finalmente a noção de extensionalidade podemos fazer uma correspondência mais precisa deste sistema com a sistema dos λ-termos com $=_{\beta\eta}$. Por exemplo, $SK \neq_w KI$ mas $SK =_\beta KI$.

Definição 10 *A regra da extensionalidade (***t***) é*

$$Ax = Bx \quad \Rightarrow \quad A = B \ se \ x \notin FV(AB)$$

Exercício 13 (**)Mostre que :

1. Se $P \mapsto_w Q$ então $\lambda(P) \mapsto_w \lambda(Q)$
2. Se M é um λ-termo então $\lambda(\mathcal{CL}(M)) =_{\beta\eta} M$
3. Se $P \ \in \mathcal{CL}(I,K,S)$então $\mathcal{CL}(\lambda P)) =_{wt} P$
4. Se $M =_{\beta\eta} N$ então $\mathcal{CL}(M) =_{wt} \mathcal{CL}(N)$

Outros combinadores

Os combinadores B e C podem ser adicionados ao sistema $\mathcal{CL}(I,K,S)$, diminuindo o tamanho dos termos gerados por o algoritmo da abstração.
As regras de redução destes combinadores são:

$$\begin{array}{lcl} CPQR & \mapsto_w & PRQ \\ BPQR & \mapsto_w & P(QR) \end{array}$$

A C chama-se inversor e a B compositor. O sistema $\mathcal{CL}(I,K,B,C,S)$pode ser usado como uma optimização do sistema $\mathcal{CL}(I,K,S)$. Em particular o algoritmo de abstração pode incluir mais estas duas regras:

$$\begin{array}{lcl} \lambda^\star x.PQ & \equiv & C(\lambda^\star x.P)Q \text{ se } x \in P \text{ e } x \notin Q \\ \lambda^\star x.PQ & \equiv & BP(\lambda^\star x.Q) \text{ se } x \notin P \text{ e } x \in Q \end{array}$$

Sendo a regra para S só usada se x ocorre em P e Q.

Exercício 14 Mostre que $B \equiv \lambda xyz.x(yz)$ e $C \equiv \lambda xyz.xzy$ são λ-termos que correspondem aos combinadores atrás definidos, isto é $BMNL =_\beta M(NL)$ e $CMNL =_\beta MLN$.

Exercício 15 Usando os algoritmos anteriores escreva termos combinatórios $P \in \mathcal{CL}(I, K, B, C, S)$ (compare com o correspondente no sistema $\mathcal{CL}(I, K, S)$) equivalente a:

1. $\lambda xy.x$
2. $\lambda xy.y$
3. $\lambda xyz.y(xyz)$
4. $\lambda xy.u(z(xv)(yv))$

O combinador Y com regra de redução $YP \mapsto_w P(YP)$ pode ser também usado para definições recursivas.

Capítulo 3

Ordens Parciais Completas

Definição 11 (PO) *Uma ordem parcial (PO) é um par* $(D,\sqsubseteq)$ *onde* D *é um conjunto (domínio) e* $\sqsubseteq$ *é uma relação binária sobre* D *que é reflexiva, antissimétrica e transitiva.*

Definição 12 *Seja* $(D,\sqsubseteq)$ *uma ordem parcial e* $X \subseteq D$*, um elemento* $d \in D$ *é:*

1. *um* majorante *de* X *sse* $\forall\ x\ \in\ X,\ x\ \sqsubseteq\ d$.
2. *o* supremo *(*lub*) de* X *(e representa-se por* $\bigsqcup X$*) sse* d *é um majorante de* X *e* $\forall\ d'$ *majorantes de* X*,* $d\ \sqsubseteq\ d'$*. Se existir, é único. Se* $X = \{x,y\}$ *o seu supremo se existir representa-se por* $x \sqcup y$*. O supremo de* D *designa-se por* $\top_D$.
3. *Dualmente se definem* minorante *e* ínfimo *(*$\bigsqcap X$*). Se* $X = \{x,y\}$ *o seu ínfimo se existir representa-se por* $x \sqcap y$*. O ínfimo de* D *se existir designa-se por* $\bot_D$ *(elemento mínimo).*

Definição 13

1. *Um subconjunto* $X \subseteq D$ *é* directo *se todo o subconjunto finito de* X *tem supremo em* X.
2. *Um PO* $(D,\sqsubseteq)$ *é um reticulado se* $\bigsqcup X$ *e* $\bigsqcap X$ *existem para todo subconjunto finito não vazio* $X \subseteq D$.
3. *Um reticulado é* completo *se* $\bigsqcup X$ *e* $\bigsqcap X$ *existem para todo subconjunto* $X \subseteq D$.

Definição 14 (Cadeia) *Para uma ordem parcial* $(D,\sqsubseteq)$*, um conjunto* $X \subseteq D$ *é uma* cadeia*se* $X = \{\ x_i \mid i \geq 0\}$ *e* $x_1\ \sqsubseteq\ x_2 \sqsubseteq \ldots$*. Representaremos a cadeia* X *por* $< x_i >$.

Exercício 16 *Mostre que uma cadeia é um conjunto directo.*

Definição 15 (CPO) *Uma ordem parcial* $(D,\sqsubseteq)$ *é uma ordem parcial completa (CPO) se*

1. *Tem um elemento mínimo, que se representa por $\bot_D$, ou apenas por $\bot$ se não houver ambiguidade.*

2. *Toda a cadeia $X \subseteq D$, tem um supremo em D, que se representa por $\bigsqcup X$ ou por $\bigsqcup < x_i >$.*

Teorema 3 *Uma PO (D,$\sqsubseteq$) com elemeto mínimo é um CPO se e só se todo o subconjunto directo de D tiver supremo em D.*

Exercício 17 1. *Dê exemplos de PO que para alguns dos seus subconjuntos:*

   (a) *não tenham magorantes.*

   (b) *tenham majorantes mas não supremos.*

   (c) *tenham ínfimos mas não supremos.*

   (d) *1a e 1b.*

2. *Mostre que (D,$\sqsubseteq$) é um reticulado se e só se $x \sqcup y$ e $x \sqcap y$ existem para todo o $x, y \in D$.*

3. *Mostre que toda a cadeia não vazia é um reticulado. Dê exemplo dum reticulado que não é uma cadeia.*

4. *Mostre que todo o reticulado finito é completo.*

5. *Mostre que todo o PO finito com um elemento mínimo é um CPO.*

Notação

- $\mathbf{N} = \{0, 1, 2, \cdots\}$
- $\mathbf{T} = \{t, f\}$, conjunto dos valores lógicos.
- $A_\bot = A \cup \{\bot_A\}$
- $(A_\bot, \sqsubseteq)$: CPO.
- $A \longrightarrow B$: conjunto das funções de A em B.
- $[A \longrightarrow B]$: conjunto das funções contínuas de A em B.
- A função f diz-se *naturalmente estendida* se $f(a_1, \cdots, a_n) = \bot$ se pelo menos um dos a_i é $\bot$.

Exercício 18 *Para cada caso (e quando aplicável) pergunta-se*

- *Trata-se de uma PO (relação de ordem parcial)?*
- *Trata-se de um CPO?*

- *Como são as cadeias de elementos?*
- *Como se pode determinar o supremo das cadeias?*

1. $(\mathbf{N}, \leq)$
2. $(\mathbf{N}, \geq)$ *onde, obviamente,*

$$\geq = \{(a,b) \mid a \in \mathbf{N}, b \in \mathbf{N}, a \geq b\}$$

3. $(\mathcal{P}(A), \subseteq)$ *onde A é um conjunto, $\mathcal{P}(A)$ o conjunto dos seus subconjuntos e $\subseteq$ representa a relação de inclusão entre conjuntos.*

Exercício 19 *Supõe que $(A, \leq_A)$ e $(B, \leq_B)$ são ordens parciais.*

1. *Mostre que $(A \times B, \leq)$ é ainda uma ordem parcial se definirmos*

$$(a,b) \leq (a',b') \text{ sse } a \leq_A a' \text{ e } b \leq_B b'$$

2. *Mostre que se s é o supremo da cadeia $<a_i>$ e se s' é o supremo da cadeia $<b_i>$ então (s,s') é o supremo da cadeia $<(a_i,b_i)>$ em $(A \times B, \leq)$. Como consequência, se $(A, \leq_A)$ e $(B, \leq_B)$ forem CPO's então $(A \times B, \leq)$ também o é.*
3. *Desenhe o diagrama de Hasse para*

$$(T_\perp \times A_\perp, \sqsubseteq)$$

onde $A = \{x, y, z\}$ e $\sqsubseteq$ é a odem discreta, definida no Exercício 1(d)i.

Exercício 20 *Prove que os seguintes conjuntos são* CPO*s:*

1. *($\mathbf{N}_\perp$,$\sqsubseteq$), onde $\sqsubseteq$ é definida por : $x \sqsubseteq y$ sse $(x = \perp) \vee (x = y)$ (a esta relação chama-se* ordem discreta*).*
2. *($\mathcal{FN}_\perp$,$\sqsubseteq$) onde $\mathcal{FN}_\perp$ é o espaço das funções de $\mathbf{N}_\perp$ em $\mathbf{N}_\perp$ e $\sqsubseteq$ é definida por:*

$$\forall\, f,g \in \mathcal{FN}_\perp,\ f \sqsubseteq g \Longleftrightarrow \forall\, n \in \mathbf{N}_\perp,\ f(n) \sqsubseteq_{N_\perp} g(n),$$

3. *($X \longrightarrow D$,$\sqsubseteq$) com X e D* CPO*s e a relação de ordem:*

$$\forall\, f,g \in X \longrightarrow D,\ f \sqsubseteq g \Longleftrightarrow \forall\, x \in X,\ f(x) \sqsubseteq_D g(x).$$

4. *($\mathbf{T}_\perp$,$\sqsubseteq$)*

3.1 Funções Contínuas

Definição 16 (Função Contínua) *Sejam D e D' dois CPOs, $f{:}D \longrightarrow D'$ é contínua se e só se:*

1. *f é monótona, i.e, $\forall\ d_1, d_2 \in D, d_1 \sqsubseteq_D d_2 \implies f(d_1) \sqsubseteq_{D'} f(d_2)$*
2. *$\forall\ < x_i > \in\ D, f(\bigsqcup < x_i >) = \bigsqcup < f(x_i) >$*

Exercício 21 ‘

Mostre que as seguintes funções são monótonas

1. *A função identidade,*
$$id : (D_\perp)^n \longrightarrow (D_\perp)^n$$
2. *A função totalmente indefinida, $und(x) = \perp$ para todo o $x \in D_\perp$*
$$und : D_\perp \longrightarrow D_\perp$$
3. *A função quociente*
$$/ : \mathbf{N}_\perp \times \mathbf{N}_\perp \longrightarrow \mathbf{N}_\perp$$
naturalmente estendida, definindo-se $a/0 = \perp$ para todo o $a \in \mathbf{N}$.
4. *A função ite (`if...then...else...`)*
$$ite : \mathbf{T}_\perp \times (\mathbf{N}_\perp)^2 \longrightarrow \mathbf{N}_\perp$$
definida por (onde a e b são quaisquer elementos de $\mathbf{N}_\perp$)
$$\begin{aligned} ite(t, a, b) &= a \\ ite(f, a, b) &= b \\ ite(\perp, a, b) &= \perp \end{aligned}$$
Mostre que esta função não é uma extensão natural de `if...then...else...` (embora seja monótona).

Exercício 22

Considera o conjunto de funções
$$F = \mathbf{T}_\perp \longrightarrow \{a, \perp\}$$
(definindo-se $f \sqsubseteq g$ se para todo o $a \in \mathbf{T}_\perp$ é $f(a) \sqsubseteq g(a)$.

1. *Caracteriza (por uma tabela) todas as funções de F.*

2. *Qual o subconjunto $F' \subset F$ de funções monótonas.*

3. *Mostre que $(F', \sqsubseteq)$ é PO.*

4. *Mostre que $(F', \sqsubseteq)$ é CPO.*

Exercício 23 *Mostre que a composta de duas funções contínuas entre* CPO*s é contínua.*

Proposição 2 *Sendo D e E dois CPOs então $[D \longrightarrow E]$ é um CPO com a seguinte relação de ordem*

$$\forall\, f, g \in [D \longrightarrow E]\ f \sqsubseteq g \Longleftrightarrow \forall\, d \in D,\ f(d) \sqsubseteq g(d).$$

Exercício 24 *Demonstre a Proposição 2.*

Teorema 4 *Para qualquer CPO D, toda a função $f \in [D \longrightarrow D]$ tem um ponto fixo mínimo, i.e,*

- *$Fix\, f \in D$ e $f(Fix\, f) = Fix\, f$*
- *$\forall\, d \in D, f(d) = d \implies Fix f \sqsubseteq d$.*

Dem: S=$< f^i(\perp_D) >$ é uma cadeia em D porque f é monótona (verifique!). Defina Fix f=$\bigsqcup$S.

Exercício 25 *Termine a demonstração do Teorema 6.*

Exercício 26 *Determine os pontos fixos mínimos das seguintes funções:*

1.

$$\begin{array}{lcll} F : [N_\perp \longrightarrow N_\perp] & \longrightarrow & [N_\perp \longrightarrow N_\perp] \\ g & \longmapsto & f : N_\perp \longrightarrow N_\perp \\ & & x \mapsto g(x+1) \end{array}$$

2.

$$\begin{array}{lcll} F : [N_\perp \longrightarrow N_\perp] & \longrightarrow & [N_\perp \longrightarrow N_\perp] \\ g & \longmapsto & f : N_\perp \longrightarrow N_\perp \\ & & 0 \mapsto 1 \\ & & x \mapsto g(x+1) \end{array}$$

3.

$$\begin{array}{lcll} F : [N_\perp \longrightarrow N_\perp] & \longrightarrow & [N_\perp \longrightarrow N_\perp] \\ g & \longmapsto & f : N_\perp \longrightarrow N_\perp \\ & & 0 \mapsto 1 \\ & & x \mapsto g(x-1) \end{array}$$

Proposição 3 *Sejam (D,$\sqsubseteq$) um CPO e FIX: [D $\longrightarrow$ D] $\longrightarrow$ D a função que a cada $f \in$ [D $\longrightarrow$ D] associa o menor ponto fixo de f, então FIX é contínua, i.e, FIX $\in$ [[D $\longrightarrow$ D] $\longrightarrow$ D].*

Exercício 27 *Demonstre a Proposição 3.*

Exercício 28 *Seja a definição de* $fact : [\mathbf{N} \longrightarrow \mathbf{N}]_\perp$

$$fact(n) = \begin{cases} 1 & se\ n = 0 \\ n \times fact(n-1) & se\ n \geq 1 \end{cases}$$

1. *Defina o funcional correspondente* $F : [\mathbf{N} \longrightarrow \mathbf{N}]_\perp \longrightarrow [\mathbf{N} \longrightarrow \mathbf{N}]_\perp$. *Note que a definição de F é não recursiva.*

2. *Mostre que o funcional F é monótono e contínuo.*

3. *Caracterize a função* $F^n(\perp) \in [\mathbf{N} \longrightarrow \mathbf{N}]_\perp$ *para cada valor de n, onde* $\perp$ *representa a função totalmente indefinida,* $\lambda n.\perp$. *Considere em detalhe as funções* $F^n(\perp)$ *para n = 0, 1, 2 e 3.*

4. *Mostre que o ponto fixo mínimo de F é*

$$f = \bigsqcup < F^n(\perp) >$$

e que coincide com a função x!.

Exercício 29 *(i) Dado (D,$\sqsubseteq$) e* $f, g \in [D \longrightarrow D]$ *tal que* $f \circ g = g \circ f$ *e* $f(\perp) = g(\perp)$ *Mostre que: Fix f = Fix g.*

(ii) Sendo (D,$\sqsubseteq$) e (E,$\sqsubseteq$) dois CPOS, $h : D \longrightarrow D$ *e* $g : D \longrightarrow D$ *funções contínuas e* $f : D \longrightarrow E$ *contínua e estrita, tal que* $g \circ f = f \circ h$, *mostre que* $Fix\ g = f(Fix\ h)$.

(iii) Sendo (D,$\sqsubseteq$) e (E,$\sqsubseteq$) CPOS, $f : D \longrightarrow E$ *e* $g : E \longrightarrow D$ *funções contínuas, mostre que* $Fix(g \circ f) = g(Fix(f \circ g))$

3.2 Indução do Ponto Fixo

Para funções definidas recursivamente, o Princípio de Indução do Ponto Fixo permite provar que uma "propriedade" é válida se o for para todas as suas subfunções finitas. Uma "propriedade" será definida como um predicado inclusivo:

Definição 17 *Sendo D um CPO, um predicado* $P : D \longrightarrow \mathbf{T}$ *é* inclusivo *sse para toda a cadeia* $X \subset D$, *se para todo* $x_i \in X$, $P(x_i) = t$ *então* $P(\bigsqcup X) = t$.

Definição 18 Indução do ponto fixo *Seja H uma função contínua num CPO D e $\mathcal{P}(x)$ um predicado inclusivo em D. Se*

1. *$\mathcal{P}(\bot)$ se verifica.*
2. *se $\mathcal{P}(x)$ se verifica então $\mathcal{P}(H(x))$ também se verifica*

então podemos inferir que $\mathcal{P}(Fix\ H)$ também se verifica.

Exercício 30 *Repita o Exercício 35 usando indução do ponto fixo, supondo que as propriedades em questão são predicados inclusivos.*

Capítulo 4

Domínios Semânticos

Esta secção caracteriza operadores sobre domínios que serão utilizados nos domínios semânticos.

4.1 Produtos

Proposição 4 (Produto Cartesiano) *Dados D e E CPOs, o produto*

$$D \times E \;=\; \{< x, y > \mid x \;\in\; D,\; y \;\in\; E\}$$

com a relação de ordem $\sqsubseteq$:

$$< x, y > \sqsubseteq < x', y' > \iff x \;\sqsubseteq_D\; x' \; e \; y \;\sqsubseteq_E\; y'$$

é um CPO.

Exercício 31 *Demonstre a Proposição (4).*

Definição 19 *Definam-se as seguintes funções:*

1. *Sendo D um* CPO, *cond: $T \times D \times D \longrightarrow D$ dada por:*

$$cond(b, x, y) = \begin{cases} \bot_D & se\ b = \bot_T \\ x & se\ b = t \\ y & se\ b = f \end{cases}$$

2. *Sendo D e E* CPOs *prod:$D \longrightarrow E \longrightarrow D \times E$, fst:$D \times E \longrightarrow D$ e snd:$D \times E \longrightarrow E$ são definidas, para todo $x \in D$ e $y \in E$, por:*

$$\begin{array}{lcl} prod(x)(y) & = & < x, y > \\ fst(< x, y >) & = & x \\ snd(< x, y >) & = & y \end{array}$$

Exercício 32 *Mostre que cond, prod, fst e snd são funções contínuas entre CPOs.*

Notação: A função $cond(b, x, y)$ representa-se habitualmente por $b \longrightarrow x, y$.

Proposição 5 *Dados D, E e F CPOs e* $f : F \longrightarrow D$ *e* $g : F \longrightarrow E$ *funções contínuas existe uma função contínua* $< f, g >: F \longrightarrow D \times E$ *tal que*

$$fst \circ < f, g > = f$$
$$snd \circ < f, g > = g$$

Exercício 33 *Mostre a Proposição (5).*

Definição 20 *Dadas* $f{:}D \longrightarrow D'$ *e* $g{:}E \longrightarrow E'$ *funções contínuas entre CPOs, definimos* $f \times g : D \times E \longrightarrow D' \times E'$ *como:*

$$f \times g = < f \circ fst, g \circ snd >$$

Exercício 34 *Considerando a Definição (??) mostre que* $id_D \times id_E = id_{D \times E}$ *e que*

$$(f \times g) \circ (f' \times g') = (f \circ f') \times (g \circ g')$$

Proposição 6 (apply e curry) *Os operadores* $\longrightarrow$ *e* $\times$ *podem se relacionar do modo seguinte. Sejam D,E e F CPOs. Define-se a função* **apply**: $[[E \longrightarrow F] \times E] \longrightarrow F$ *como:*

$$\forall\, f : E \longrightarrow F,\ x \in E,\ \ apply(< f, x >) = f(x)$$

e sendo $f \in [D \times E \longrightarrow F]$, *a função* $\mathbf{curry}(f) : D \longrightarrow [E \longrightarrow F]$ *tal que:*

$$\forall x \in D,\ y \in E,\ \ curry(f)(x)(y) = f(x, y)$$

Então,

$$f = apply \circ (curry(f) \times id_D)$$

Exercício 35 *Mostre que* **apply** *e* **curry** *são funções contínuas e que a igualdade da Proposição (??) se verifica. Explicite a relação entre os dois operadores referidos.*

Pode-se generalizar o produto cartesiano a n CPOs:

Definição 21 *Dados* $D_1, \ldots, D_n$ *CPOs define-se*

$$\Pi(D_1) = D_1$$

$$\Pi(D_1, \ldots, D_n) = \Pi(D_1, \ldots, D_{n-1}) \times D_n$$

e as funções

$$proj_i : \Pi(D_1, \ldots, D_n) \longrightarrow D_i$$

tal que

$$prog_i = fst^{n-i} \circ snd$$

4.2 Produtos amalgamados

No produto cartesiano entre CPOs, $D \times E$, os elementos da forma $< x, \perp >$ e $< \perp, y >$ são distintos se $x \neq y$. Em certos casos, podemos querer identificar estes elementos. Por exemplo, em semântica denotacional de linguagens de programação para a *passagem de variáveis por valor*. Assim pode-se definir o chamado *produto amalgamado*.
Recorde que,

Definição 22 (Função estrita) *Uma função $f : D \longrightarrow E$ entre CPOs é estrita sse $f(\perp_D) = \perp_E$. Então representa-se f por $f : D \longrightarrow_\perp E$e o CPO das funções contínuas estritas entre dois CPOs D e E por$[D \longrightarrow_\perp E]$*

Exercício 36 *Verifique que$[D \longrightarrow_\perp E]$ é um sub-CPO de $[D \longrightarrow E]$, isto é, é um subconjunto e tem o mesmo elemento mínimo e os mesmos supremos das cadeias ascendentes.*

Definição 23 *Seja a função contínua $strict : [D \longrightarrow E] \longrightarrow [D \longrightarrow_\perp E]$ tal que*

$$strict(f)(x) = \begin{cases} f(x) & se\ x \neq \perp \\ \perp & se\ x = \perp \end{cases}$$

Definição 24 (Par mergulho/projecção) *Um par de funções contínuas $g : D \longrightarrow E$ e $f : E \longrightarrow D$ diz-se um par mergulho/projecção (g é um mergulho e f uma projecção) se satisfazem*

$$f \circ g = id_D$$
$$g \circ f \sqsubseteq id_E$$

Exercício 37 *Mostre que a função* **strict** *é uma projecção cujo mergulho correspondente é a função inclusão*

$$incl : [D \longrightarrow_\perp E] \longrightarrow [D \longrightarrow E]$$

Definição 25 (Produto amalgamado) *Sendo D e E CPOS, o produto amalgamado $D \otimes E$ é o conjunto*

$$\{< x, y > \in D \times E \mid x \neq \perp e\ y \neq \perp\} \cup \{\perp_{D \otimes E}\}$$

com a relação de ordem $\sqsubseteq$:

$$< x, y > \sqsubseteq < x', y' > \iff x \sqsubseteq_D x' \ e \ y \sqsubseteq_E y'$$

e

$$\forall z \in D \otimes E, \ \perp_{D \otimes E} \sqsubseteq z$$

É evidente que $D \otimes E$ é um CPO.

Proposição 7 *A função contínua* $smash : [D \times E] \longrightarrow [D \otimes E]$ *tal que*

$$smash(x,y) = \begin{cases} (x,y) & se\ x \neq \bot\ e\ y \neq \bot \\ \bot_{D\otimes E} & caso\ contrário \end{cases}$$

é uma projecção cujo mergulho é dado por: $unsmash : [D \otimes E] \longrightarrow [D \times E]$ *tal que*

$$unsmash(z) = \begin{cases} z & se\ z =< x,y > \\ < \bot, \bot > & se\ z = \bot_{D\times E} \end{cases}$$

Exercício 38 *Mostre a Proposição (6).*

Exercício 39 *Sendo* $f : D \longrightarrow D'$ *e* $g : E \longrightarrow E'$ *funções contínuas estritas, mostre que a seguinte função é contínua e estrita*

$$f \otimes g \ = \ smash \circ (f \times g) \circ unsmash$$

Como no caso do produto cartesiano, pode-se relacionar o operador $\otimes$ com o operador $\longrightarrow_\bot$. Tente definir duas funções **strict_apply** e **strict_curry** usando as funções **apply**, **curry**, **strict**, **smash**, **unsmash**, **id** e **incl**.

4.3 Soma

Definição 26 (Soma amalgamada) *Dados* D *e* E *CPOs, a soma*

$$D \oplus E \ = \ \{< x,1 > | \ x \ \in \ D - \{\bot_D\}\} \cup \{< y,2 > | \ y \ \in \ E - \{\bot_E\}\} \cup \{\bot_{D\oplus E}\}$$

com a relação de ordem $\sqsubseteq$*:*

$$< x,m > \sqsubseteq < x',n > \iff \ m = n\ e\ x \ \sqsubseteq \ x'$$

e

$$\forall z \ \in \ D \oplus E, \ \bot_{D\oplus E} \sqsubseteq z$$

Exercício 40 *Demonstre que* $D \oplus E$ *é um CPO se* D *e* E *o forem.*

Definição 27 *Sendo* D_1 *e* D_2 *dois CPOs, definem-se,* $1 \leq i \leq 2$, $in_i : D_i \longrightarrow (D_1 \oplus D_2)$, $out_i : (D_1 \oplus D_2) \longrightarrow D_i$ *e* $is_i : (D_1 \oplus D_2) \longrightarrow T$*:*

$$in_i(x) = \begin{cases} < x,i > & se\ x \neq \bot \\ \bot_{D_1\oplus D_2} & se\ x = \bot \end{cases}$$

$$out_i(d) = \begin{cases} \bot_{D_i} & se\ d = \bot_{D_1\oplus D_2} \\ x & se\ d =< x,i > e\ x \ \in \ D_i \\ \bot & se\ d =< x,i >,\ x \ \in \ D_j\ e\ i \neq j \end{cases}$$

$$is_i(d) = \begin{cases} \bot_T & se\ d = \bot_{D_1\oplus D_2} \\ t & se\ d =< x,i > e\ x \ \in \ D_i \\ f & se\ d =< x,i >,\ x, \in \ D_j\ e\ i \neq j \end{cases}$$

Exercício 41 *Mostre que as funções definidas na definição (29) são contínuas e estritas e que* $out_i \circ in_i = id_{D_i}$ *e*

$$id_D = cond \circ < is_1, in_1 \circ out_1, in_2 \circ out_2 >$$

Notação: Habitualmente sendo $D = D_1 \oplus D_2$ designa-se $in_i(x)$ por x **em** D e $out_i(d)$ por $d \mid D_i$. Como no caso dos produtos,

Proposição 8 *Se* $f : D_1 \longrightarrow_\perp F$ *e* $g : D_2 \longrightarrow_\perp F$ *são funções estritas contínuas entre CPOs, então existe uma única função estrita e contínua* $[f, g] : D_1 \oplus D_2 \longrightarrow F$ *tal que*

$$[f, g] \circ in_1 = f$$

$$[f, g] \circ in_2 = g$$

Exercício 42 *Mostre a Proposição (7).*

Definição 28 *Dadas f:*$D \longrightarrow_\perp D'$ *e g:*$E \longrightarrow_\perp E'$ *funções contínuas estritas entre CPOs, definimos* $f \oplus g : D \oplus E \longrightarrow_\perp D' \oplus E'$ *como sendo:*

$$f \oplus g = [in_1 \circ f, in_2 \circ g]$$

Exercício 43 *Considerando a definição (30) mostre que* $id_D \oplus id_E = id_{D \oplus E}$ *e que*

$$(f \oplus g) \circ (f' \oplus g') = (f \circ f') \oplus (g \circ g')$$

Pode-se generalizar a soma amalgamada a n-CPOs:

Definição 29 *Dados* $D_1, \ldots, D_n$ *CPOs define-se*

$$\sum(D_1) = D_1$$

$$\sum(D_1, \ldots, D_n) = \sum(D_1, \ldots, D_{n-1}) \oplus D_n$$

e as funções

$$in_i : \sum(D_1, \ldots, D_n) \longrightarrow D_i$$

tal que

$$in_i = in_1^{n-i} \circ in_2$$

Definição 30 (Lift) *Dado um CPO D, define-se o* **lift** *de D como o conjunto* $D_\perp = (D \times \{0\} \cup \{\perp\}$, *onde* $\perp$ *é um novo elemento, e a relação de ordem parcial* $\sqsubseteq$*:*

$$< x, 0 > \sqsubseteq < x', 0 > \iff x \sqsubseteq_D x'$$

e

$$\forall z \in D_\perp, \ \perp \sqsubseteq z$$

Exercício 44 *Mostre que $D_\perp$ é CPO se D o é.*

Proposição 9 *A função $down : D_\perp \longrightarrow_\perp D$ dada por*

$$down(z) = \begin{cases} x & se\ z =< x, 0 > \\ \perp_D & caso\ contrário \end{cases}$$

é contínua estrita e a função $up : D \longrightarrow D_\perp$ dada por

$$up(x) =< x, 0 >$$

é contínua e verificam as seguintes relações:

$$down \circ up = id_D$$
$$up \circ down \sqsupseteq id_{D_\perp}$$

Exercício 45 *Mostre a Proposição (8).*

Proposição 10 *Dados dois CPOs D e E e uma função contínua $f : D \longrightarrow E$ existe uma função contínua estrita $g : D_\perp \longrightarrow E$ tal que $f = g \circ up$. Dizemos que g=lift(f).*

Exercício 46 *Mostre a Proposição (??).*

Definição 31 *Dada uma função contínua $f : D \longrightarrow E$ define-se $f_\perp : D_\perp \longrightarrow E_\perp$ por,*

$$f_\perp = strict(up \circ down)$$

Podemos dizer que o *lift* transforma funções contínuas em funções contínuas estritas. Uma aplicação desta operação ocorre quando não se pretende *amalgamar* os elementos mínimos dos somandos duma soma de CPOs. Temos então uma *soma separada.*

Definição 32 (Soma separada) *Dados D e E CPOs, definimos soma separada $D + E$ como sendo o CPO $D_\perp \oplus E_\perp$.*

Temos também,

Proposição 11 *Se $f : D_1 \longrightarrow_\perp F$ e $g : D_2 \longrightarrow_\perp F$ são funções estritas contínuas entre CPOs, então existe uma única função contínua estrita $h : D_1 + D_2 \longrightarrow F$ tal que*

$$h \circ in_1 \circ up = f$$

$$h \circ in_2 \circ up = g$$

Exercício 47 *Mostre a Proposição (??)*

Definição 33 *Dadas f:$D \longrightarrow D'$ e g:$E \longrightarrow E'$ funções contínuas entre CPOs, definimos $f + g : D + E \longrightarrow D' + E'$ como sendo:*

$$f + g = f_\perp \oplus g_\perp$$

Exercício 48 *Mostre que $f+g$ verificam as propriedades de $f \oplus g$ definidas no exercício 46.*

Capítulo 5

Pω: Domínio Reflexivo

Bibliografia J. Stoy, *Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.**Cáp. 7**

Definição 34 *Sejam D e E CPOs e sejam* $i : E \longrightarrow D$ *e* $j : D \longrightarrow E$ *funções contínuas. O par* $< i, j >$ *é um par projeção de E em D, i um* mergulho, *j uma* projeção *e E uma* projeção *(contração) de D se e só se:*

$$j \circ i = id_E \text{ e } i \circ j \sqsubseteq id_D$$

Definição 35 *Um CPO D é reflexivo se* $[D \longrightarrow D]$ *é uma projeção de D.*

Seja ($\mathbf{P}\omega$,$\subseteq$) o conjunto potência de $\mathbf{N}$, $\mathbf{P}\omega$=$\{X \mid X \subseteq \mathbf{N}\}$, ordenado pela inclusão.

Exercício 49 *Mostre que (*$\mathbf{P}\omega$,$\subseteq$*) é um CPO.*

Qualquer elemento de $\mathbf{P}\omega$ pode ser visto como a reunião infinita de conjuntos finitos. Em particular,

Exercício 50 *Mostre que*

$$\forall\, X \in \mathbf{P}\omega, X = \bigsqcup\{a_i \mid a_i \subseteq X \wedge \forall i, a_i \text{ é finito e } a_i \subseteq a_{i+1}\}$$

indicando expressamente a cadeia $< a_i >$.

Vamos caracterizar as funções contínuas em $\mathbf{P}\omega$.

Proposição 12 *Se f:*$\mathbf{P}\omega \longrightarrow \mathbf{P}\omega$ *então,*

$$f \text{ é contínua} \Longleftrightarrow f(X) = \bigsqcup\{f(a_i) \mid a_i \subseteq X \wedge \forall i,\ a_i \text{ finito e } a_i \subseteq a_{i+1}\}$$

Exercício 51 *Demonstre a proposição (9) e conclua que* uma função contínua em $\mathbf{P}\omega$ fica caracterizada pelos seus valores em conjuntos finitos.

Dado que uma função f$\in [\mathbf{P}\omega \longrightarrow \mathbf{P}\omega]$ é determinada pelos seus valores em conjuntos finitos vamos ver que f pode ser codificada como um elemento de $\mathbf{P}\omega$.

Definição 36 (Codificações de $\mathbf{N}^2$ e de conjuntos finitos de N)

1. $\forall\, n, m \ \in \mathbf{N} \ \ (n,m) = \frac{1}{2}(n+m)(n+m+1)+m$

   é uma bijeção entre $\mathbf{N}$ *e* $\mathbf{N}^2$. *Corresponde às "diagonais" de* $\mathbf{N}^2$,

   $$(0,0),(1,0),(0,1),(2,0),(1,1),(0,2),(3,0),(2,1),\ldots$$

2. $\forall\, n \in \mathbf{N}$ *seja* $e_n = \{k_0,\ldots,k_{m-1}\}, k_i < k_{i+1}$ *e* $n = \sum_{i<m} 2^{k_i}$ *é uma bijeção entre* $\mathbf{N}$ *e* $\{X \in \mathbf{P}\omega \mid\ X$ *é finito*$\}$.

Por exemplo, $e_{13} = e_{(1101)_2} = \{0,2,3\}$ e $(1,3) = 13$.

Definição 37

1. $$\forall\, f \in [\mathbf{P}\omega \longrightarrow \mathbf{P}\omega],\ \ \mathbf{gr}(f) = \{(n,m) \mid m \ \in \ f(e_n)\}$$

   onde (n,m) *representa o seu índice em* $\mathbf{N}$.

2. $$\forall\, U \in \mathbf{P}\omega, \quad \mathbf{fun}(U) \in [\mathbf{P}\omega \longrightarrow \mathbf{P}\omega] : \\ \mathbf{fun}(U)(X) = \{m \mid \exists e_n \subseteq X \ \wedge \ (n,m) \ \in \ U\}$$

[1]

Note que **gr** *associa a cada função contínua um elemento de* $\mathbf{P}\omega$ *e* **fun** *a cada elemento de* $\mathbf{P}\omega$ *uma função contínua.*

Proposição 13

1. *gr:*$[\mathbf{P}\omega \longrightarrow \mathbf{P}\omega] \longrightarrow \mathbf{P}\omega$ *é contínua.*
2. *Para todo* $U \ \in \mathbf{P}\omega$ $\mathbf{fun}(U) \ \in \ [\mathbf{P}\omega \longrightarrow \mathbf{P}\omega]$
3. **fun** *é contínua*
4. $\forall\, f \ \in \ [\mathbf{P}\omega \longrightarrow \mathbf{P}\omega]$, $\mathbf{fun}(\mathbf{gr}(f)) = f$
5. *Para todo* $U \ \in \mathbf{P}\omega$ $\mathbf{gr}(\mathbf{fun}(U)) \supseteq U$.

Exercício 52 *Demonstre a proposição (10) e conclua que* $\mathbf{P}\omega$ *é um CPO reflexivo.*

[1] Equivalentemente $\mathbf{gr}(f) = \{(a_i,b_j) \mid \forall\, X \in \mathbf{P}\omega,\ X = \bigsqcup < a_i > \wedge f(a_i) = \bigsqcup < b_j >\}$ e $\mathbf{fun}(U)(X) = \{b \mid X = \bigsqcup < ai > \ \wedge (a_i,b) \in U\}$

Bibliografia J. Stoy, *Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977.**Cap. 8**

R. Tennent, *The Denotational Semantics of Programming Languages*, CACM August 1976, vol. 19, num. 8

Schmidt D. *Denotational Semantics*, **Cáp. 7.2**

Kooge, P. *The Architecture of Symboic Computers*, McGraw-Hill, 1991, Çap. 4 e 5

Capítulo 6

Semântica Denotacional do λ-Calculus

6.1 Semântica Denotacional do λ-Calculus Puro

O domínio sintáctico é o dos λ-termos, sendo Id o domínio das variáveis.

6.1.1 Domínios Semânticos

$$v \in V \simeq [V \to V]$$

$$\rho \in Amb = [Id \to V]$$

V é o domínio dos valores dos λ-termos e ρ é o ambiente.

Definição 38 *O ambiente $\rho[\delta/I]$ é definido por*

$$\rho[\delta/I][\![I']\!] = (I' = I) \longrightarrow \delta, \rho[\![I']\!]$$

6.1.2 Função Semântica

$\mathcal{E}$:λ-termos $\longrightarrow Amb \longrightarrow V$

1. $\mathcal{E}[\![x]\!]\rho = \rho[\![x]\!]$
2. $\mathcal{E}[\![ee']\!]\rho = \mathcal{E}[\![e]\!]\rho \mid_{[V \to V]} (\mathcal{E}[\![e']\!]\rho)$
3. $\mathcal{E}[\![\lambda x.e]\!]\rho = \lambda v.\mathcal{E}[\![e]\!]\rho[v/x]$ em V

Exercício 53 *Determine:*

(i) $\mathcal{E}[\![(\lambda x.x)]\!]\rho$

(ii) $\mathcal{E}[\![(\lambda xy.x)(\lambda x.x)]\!]\rho$

(iii) $\mathcal{E}[\![(\lambda x.xx)(\lambda x.xx)]\!]\rho$

6.1.3 Semântica Operacional e Semântica Denotacional

Considere-se a semântica operacional para o λ-calculus dada pela relação de equivalência sobre termos, $=_{\beta\eta}$. Dizemos que M e N são *convertíveis* se $M =_{\beta\eta} N$.

Definição 39

$$E_0 \equiv E_1 \Leftrightarrow (\forall \rho\ \mathcal{E}[\![E_0]\!]\rho = \mathcal{E}[\![E_1]\!]\rho)$$

Lema 3 *Se I não ocorre livre em E, então*

$$\forall \rho, \delta\ \mathcal{E}[\![E]\!]\rho[\delta/I] = \mathcal{E}[\![E]\!]\rho$$

Lema 4 $\forall \rho\ \mathcal{E}[\![[E_1/I]E_0]\!]\rho = \mathcal{E}[\![E_0]\!](\rho[\mathcal{E}[\![E_1]\!]\rho/I])$

Teorema 5 $E_0 =_{\beta\eta} E_1 \Rightarrow E_0 \equiv E_1$

Demonstração 3 *Recorde a definição de $\rightarrow_\beta$, $\rightarrow_\alpha$ e $\rightarrow_\eta$:*

(α) $\lambda x.X \rightarrow_\alpha \lambda y.[y/x]X$ *e y não ocorre livre em X*

(β) $(\lambda x.M)N \rightarrow_\beta [N/x]M$

(η) $(\lambda x.Mx) \rightarrow_\eta M$ *e x não ocorre livre em M*

Duas expressões E_1 e E_2 são convertiveis se:

1. *uma pode ser transformada na outra pela aplicação directa duma regra;*
2. *uma pode ser transformada na outra pela aplicação duma regra a uma das suas componentes;*
3. *uma pode ser transformada na outra pela aplicação duma sequência finita de regras de conversão*

A relação $\equiv$, como a $=_{\beta\eta}$, é uma relação de equivalência. Portanto o caso 3) está resolvido. O caso 2) resolve-se por indução estrutural. *Para o caso 1) utilizam-se os lemas dados.* •

Exercício 54 *Demonstre os lemas 5 e 6. Para o lema 6 use indução estrutural sobre E_0 e a definição de substituição. Complete a demonstração do teorema 7.*

Em geral, $E_0 =_{\beta\eta} E_1 \not\Leftarrow E_0 \equiv E_1$. Contudo, a implicação é válida se os λ-termos admitirem forma normal.

Exercício 55 *Discuta estes factos para a implementação do λ-calculus usando reduções $\beta\eta$.*

Exercício 56 *Modifique a semântica dada para o λ-calculus considerando a passagem de argumentos por* **valor**.

6.2 Semântica Denotacional do λ-Calculus com extensão aos inteiros

6.2.1 Domínios Semânticos

$n \in \mathbf{N}$	inteiros
$v \in V \simeq \mathbf{N}_\perp + [V \to V]$	valores
$\rho \in Amb = [Id \to V]$	ambientes
Id	variáveis do λ-Calculus

6.2.2 Função Semântica

$\mathcal{E}$: λ-termos $\longrightarrow Amb \longrightarrow V$

1. $\mathcal{E}[\![\underline{n}]\!]\rho = n$ em V
2. $\mathcal{E}[\![x]\!]\rho = \rho[\![x]\!]$
3. $\mathcal{E}[\![ee']\!]\rho = \mathcal{E}[\![e]\!]\rho \mid_{[V \to V]} (\mathcal{E}[\![e']\!]\rho)$
4. $\mathcal{E}[\![\lambda x.e]\!]\rho = \lambda v.\mathcal{E}[\![e]\!]\rho[v/x]$ em V

Exercício 57 *Calcule* $\mathcal{E}[\![(\lambda xy.xy)(\lambda x.x)\underline{5}]\!]\rho$

1. *Directamente*
2. *Usando o Teorema 7*

Exercício 58 *Calcule* $\mathcal{E}[\![(\lambda f.(\lambda x.f(fx)))s\ \underline{3}]\!]\rho$, *onde* ρ *é definido por*

$$\rho[\![x]\!] = \begin{cases} \perp & \text{se } x \neq s \\ \textbf{dobro} & \text{se } x = s, \text{ onde } \textbf{dobro} \text{ é a função definida em } \textbf{Ndobro}(n)=2n \\ & \text{e estendida naturalmente a } V \end{cases}$$

Exercício 59 *(i) Estenda a linguagem dos λ-termos para incluir funções aritméticas explícitamente ("build-in"), por exemplo, $+$, $\times$ e $-$. Defina uma semântica denotational para o λ-calculus resultante.*

(ii) Determine

(a) $\mathcal{E}[\![(\lambda x.x + \underline{5})\underline{3}]\!]\rho$

(b) $\mathcal{E}[\![(\lambda xyz.x(yz))(\lambda x.x + \underline{4})(\lambda x.x \times x)\underline{7}]\!]\rho$

6.3 Semântica Denotacional do λ-Calculus com extensão aos inteiros com detecção de erros

6.3.1 Domínios Semânticos

$n \in \mathbf{N}$	inteiros
$w \in W = \{\underline{erro}\}_\perp$	valor de erro
$v \in V \simeq \mathbf{N}_\perp + [V \to V] + W$	valores
$\rho \in Amb = Id \to [V + \{\underline{indefinido}\}_\perp]$	ambientes
Id	variáveis do λ-Calculus

6.3.2 Função Semântica

Nota: Para simplificar notação designámos $\underline{erro}$ por $\underline{erro}$ em V. $\mathcal{E}$: λ-termos $\longrightarrow$ $Amb \longrightarrow V$

1. $\mathcal{E}[\![\underline{n}]\!]\rho = n$ em V
2. $\mathcal{E}[\![x]\!]\rho = \rho[\![x]\!] = \underline{indefinido} \longrightarrow \underline{erro}, \rho[\![x]\!]$ em V
3. $\mathcal{E}[\![e_1 e_2]\!]\rho = \begin{cases} isW(v_2) \longrightarrow \underline{erro}, & \\ isW(v_1) \longrightarrow \underline{erro}, & \text{onde } v_i = \mathcal{E}[\![e_i]\!]\rho \\ is[V \to V](v_1) \longrightarrow v_1 \mid_{[V \to V]} v_2, \underline{erro} & \end{cases}$
4. $\mathcal{E}[\![\lambda x.e]\!]\rho = \lambda v.\mathcal{E}[\![e]\!]\rho[v/x]$ em V

Exercício 60 *Repita os exercícios 58 e 60 para esta semântica.*

6.4 Semântica Denotacional do λ-Calculus com funções *built-in*

Considere que a sintaxe dos λ-termos (expressões) e é dada pela seguinte **bnf**:

$$\begin{aligned} e \ ::= \ & x \mid \underline{n} \mid t \mid f \mid ee \mid \lambda x.e \mid \\ & e + e \mid e - e \mid e \times e \mid e = e \mid e > e \mid e < e \mid \ldots \\ & if\ e\ then\ e\ else\ e \mid let\ x\ be\ e\ in\ e \end{aligned}$$

Exercício 61 *(i) Verifique que a linguagem definida é equivalente ao λ-calculus puro.*

(ii) Defina uma semântica denotacional para a sintaxe dos λ-termos acima definida, considerando que

$$let\ x\ be\ e_1\ in\ e_2 \equiv (\lambda x.e_2)e_1$$

(iii) Prove que para todo o ambiente ρ,

$$\mathcal{E}[\![if\ e_1\ then\ e_2\ else\ e_3]\!]\rho = \mathcal{E}[\![(\lambda x\lambda y\lambda z.if\ x\ then\ y\ else\ z)e_1e_2e_3]\!]\rho$$

Exercício 62 *Como se pode modificar a sintaxe da expressão*

$$let\ x\ be\ e\ in\ e$$

para permitir a definição simultanea de várias variáveis, por exemplo,

$$let\ x = \underline{3}, y = x + \underline{1}\ in\ x \times y$$

Defina a função $\mathcal{E}$ para uma expressão desse tipo.

Para cada $n \geq 1$, seja $\mathcal{F}_n = \{f_1,\ f_2, \ldots\}$ um conjunto de símbolos de funções de aridade n. Considere-se $\phi \in FAmb = \prod_{n\geq 1}[\mathcal{F}_n \to [V^n \to V]]$ o conjunto de ambientes de função. Suponha que a sintaxe das expressões e é estendida a

$$e \ ::= let\ f(x_1, \ldots, x_n)\ be\ e\ in\ e \ \mid \ letrec\ f(x_1, \ldots, x_n)\ be\ e\ in\ e$$

para qualquer $n \geq 1$ e $f \in \mathcal{F}_n$.

Definição 40 *Sendo $g \in [V^n \to V]$ e $f_n \in \mathcal{F}_n$, o ambiente de função $\phi[g/f_n]$ é, para cada componente, definido por*

$$(\phi[g/f_n])_m[\![f'_m]\!] = \begin{cases} g & se\ n = m\ e\ f'_m = f_n \\ \phi_m[\![f'_m]\!] & caso\ contrário \end{cases}$$

Notação: Considerando que $\vec{v}$ denota $v_1, \ldots, v_n$ para qualquer v e n, pode-se definir $\rho[\vec{v}/\vec{x}]$ como $\rho[v_1/x_1]\ldots[v_n/x_n]$.

Seja $\mathcal{E}$: λ-termos $\longrightarrow Amb \times FAmb \longrightarrow V$ então a semântica denotacional destas duas expressões é:

1. $\mathcal{E}[\![let\ f(x_1, \ldots, x_n)\ be\ e_1\ in\ e_2]\!]\rho\phi = \mathcal{E}[\![e_2]\!]\rho\phi[\lambda\vec{v}.\mathcal{E}[\![e_1]\!]\rho[\vec{v}/\vec{x}]\phi/f]$
2. $\mathcal{E}[\![letrec\ f(x_1, \ldots, x_n)\ be\ e_1\ in\ e_2]\!]\rho\phi = \mathcal{E}[\![e_2]\!]\rho\phi[Fix(\lambda g.\lambda\vec{v}.\mathcal{E}[\![e_1]\!]\rho[\vec{v}/\vec{x}]\phi[g/f])/f]$

Exercício 63 *Verifique se estas novas expressões se podem definir no λ-calculus puro. Que alterações tem de considerar? Qual a vantagem, em caso afirmativo?*

Exercício 64 *Redefina a semântica denotacional dada no exercício 62 para todas as expressões e.*

Exercício 65 *Determine*

(i) $\mathcal{E}[\![letrec\ fact(x)\ be\ if\ x = \underline{0}\ then\ \underline{1}\ else\ x \times fact(x - \underline{1})\ in\ fact(\underline{4})]\!]\rho\phi$

(ii) $\mathcal{E}[\![letrec\ f_1(x,y)\ be\ if\ x = \underline{0}\ then\ y\ else\ f_1(x - \underline{1}, y + \underline{1})\ in\ f_1(\underline{1},\underline{2})]\!]\rho\phi$

Exercício 66 *Supondo que* $Id = Id \cup \bigcup_{n\geq 1} \mathcal{F}_n$ *e para cada* $\phi \in FAmb$ *e para* $f \in \mathcal{F}_n$, $\phi(f) = \lambda\vec{v}.v' = \lambda v_1 \ldots . \lambda v_n.v'$, *para* $f \in \mathcal{F}_n$ *podiamos apenas ter ambientes* $\rho \in Amb = [Id \rightarrow V]$. *Prove então que:*

$$\mathcal{E}[\![\ let\ f(x_1, \ldots, x_n)\ be\ e_1\ in\ e_2]\!]\rho = \mathcal{E}[\![\ let\ f\ be\ \lambda x_1, \ldots, x_n.e_1\ in\ e_2]\!]\rho$$

Exercício 67 *Defina funções que permitam manipular* **listas** *de inteiros, por exemplo* **member**, **append** *e* **reverse**. *Considere funções primitivas* **hd**, **tl**, **nil** *e* **null**.

Exercício 68 *Modifique a semântica denotacional dada considerando a passagem de argumentos por* **valor**.

Exercício 69 *Defina uma semântica operacional para as expressões e considerando uma relação* $\longrightarrow_E$:

(i) com passagem de argumentos por nome, $\mathcal{O}_n$

(ii) com passagem de argumentos por valor, $\mathcal{O}_v$

Demonstre a equivalência entre cada uma destas semânticas e as respectivas semânticas denotacionais.

Definição 41 *(***Pontos fixos simultaneos***)*
Sejam $D_0, \ldots, D_{n-1}$ *cpos e* $f_i : [\prod_{i<n} D_i \longrightarrow D_i]$, $i < n$ *funções contínuas. Os elementos* $d_0, \ldots, d_{n-1}$ *de* $D_0, \ldots, D_{n-1}$ *são pontos fixos simultaneos de* $f_0, \ldots, f_{n-1}$ *se e só se:*

$$\begin{aligned} d_0 &= f_0(d_0, \ldots, d_{n-1}) \\ &\vdots \\ d_{n-1} &= f_{n-1}(d_0, \ldots, d_{n-1}) \end{aligned}$$

Exercício 70 *Mostre que se* $d_0, \ldots, d_{n-1}$ *são pontos fixos simultaneos de* $f_0, \ldots, f_{n-1}$ *então* $< d_0, \ldots, d_{n-1} >$ *é ponto fixo de* $< f_0, \ \ldots, \ f_n >$ *e que*

$$Fix(< f_0, \ \ldots, \ f_n >) =< Fix(f_0), \ \ldots, \ Fix(f_n) >$$

Exercício 71 Funções mutuamente recursivas *Estenda a sintaxe e a semântica denotacional das expressões e para permitir a definição de funções mutuamente recursivas, por exemplo:*

$$\begin{aligned} letrec \quad & f_1(x,y)\ be\ if\ x = \underline{0}\ then\ \underline{1}\ else\ f_1(x - \underline{1}, f_2(y)), \\ & f_2(x)\ be\ if\ x = \underline{0}\ then\ \underline{6}\ else\ f_1(x - \underline{1}, x + \underline{1}) \\ & in\ f_1(\underline{1},\underline{2}) \end{aligned}$$

Sugestão: *Para a semântica denotacional use a noção de pontos fixos simultaneos.*

Capítulo 7

A Linguagem TC

Bibliografia J. Stoy, *Denotational Semantics: the Scott-Strachey Approach to Programming Language Theory*, MIT Press, 1977. **Cap. 9-12**

R. Tennent, *The Denotational Semantics of Programming Languages*, CACM August 1976, vol. 19, num. 8

Michael Gordon, *The denotational description of programming languages: An Introduction*, Springer Verlag. 1979

Schmidt D., *Denotational Semantics*, **Cap. 5, 7.1, 9**

A linguagem que consideramos a seguir, essencialmente baseada em (Stoy, 77) é uma linguagem imperativa, onde o conceito de computação corresponde ao de que um *comando* transforma um *estado* da máquina noutro *estado*.

Sintaxe

$x \in Id$ conjunto de identificadores

$o_n \in Op$ conjunto de operadores, p. e, $+$, $\times$, $-$, ...

$n \in \mathbf{N}$ conjunto de constantes (de valores primitivos, p.e, inteiros)

$e \in Exp$ conjunto de expressões

$c \in Cmd$ conjunto de comandos

onde,

$$\begin{array}{rcl} e & ::= & x \mid n \mid true \mid false \mid e = e \mid o_0() \mid o_1(e) \mid o_2(e,e) \mid \cdots \mid \\ & & \text{if e then e else e} \mid \text{let x be e in e} \mid \text{fn x.e} \mid \\ & & e(e) \\ c & ::= & dummy \mid \text{if e then c else c} \mid (c;c) \mid \text{while e do c} \end{array}$$

Domínios Semânticos

Vamos distinguir *ambientes* de *estados*.

- Os ambientes associam a cada identificador o valor da correspondente *posição* (ou endereço) de memória. O significado de um identificador (ou outra *expressão denotável*) traduz-se na posição de memória que lhe corresponde; este significado (e o ambiente) pode mudar, por exemplo, quando uma função é chamada e um identificador passa a designar uma variável local. As expressões denotáveis podem também ser nomes de funções e subrotinas.
- O estado corresponde ao conteúdo da memória, incluindo o valor do "program counter". Um estado pode mudar, por exemplo, numa atribuição. (Nota: para simplificar, na linguagem inicialmente considerada não existe nenhum comando que altere o estado, mas isso será considerado em exercícios posteriores)

$\delta \in D = [N + T + [D \longrightarrow E]]$ conjunto dos valores denotáveis

$\rho \in Amb = [Id \longrightarrow D]$ ambientes

$b \in T$ valores de verdade

$n \in N$ inteiros

$\sigma \in S$ estados

$\gamma \in C = [S \longrightarrow S]$ valores dos comandos (transformações de estados)

$v \in E = D$ valores das expressões

Funções Semânticas e Denotações

$\mathcal{E} : Exp \longrightarrow Amb \longrightarrow S \longrightarrow E$

$\mathcal{O}_n : Op \longrightarrow [E^n \longrightarrow E]$

$\mathcal{C} : Cmd \longrightarrow Amb \longrightarrow C$

- $\mathcal{E}[\![x]\!]\rho\sigma = \rho[\![x]\!]$
- $\mathcal{E}[\![n]\!]\rho\sigma = n\,em\,E$
- $\mathcal{E}[\![true]\!]\rho\sigma = t\,em\,E$
- $\mathcal{E}[\![false]\!]\rho\sigma = f\,em\,E$
- $\mathcal{E}[\![e_1 = e_2]\!]\rho\sigma = (\mathcal{E}[\![e_1]\!]\rho\sigma = \mathcal{E}[\![e_2]\!]\rho\sigma \longrightarrow$ t em E, f em E)
- $\mathcal{E}[\![o_n(e_1, \cdots, e_n)]\!]\rho\sigma = \mathcal{O}_n[\![o_n]\!](\mathcal{E}[\![e_1]\!]\rho\sigma, \ldots, \mathcal{E}[\![e_n]\!]\rho\sigma)$
- $\mathcal{E}[\![$ if e_0 then e_1 else $e_2]\!]\rho\sigma = (\mathcal{E}[\![e_0]\!]\rho\sigma | T \longrightarrow \mathcal{E}[\![e_1]\!]\rho\sigma, \mathcal{E}[\![e_2]\!]\rho\sigma)$
- $\mathcal{E}[\![$ let x be e_1 em $e_2]\!]\rho\sigma = \mathcal{E}[\![e_2]\!]\rho[\mathcal{E}[\![e_1]\!]\rho\sigma/x]\sigma$

- $\mathcal{E}[\![$ fn x.e$]\!]\rho\sigma = (\lambda\delta.\mathcal{E}[\![\ e]\!]\rho[\delta/x]\sigma)$ em D
- $\mathcal{E}[\![\ e_1(e_2)]\!]\rho\sigma = \mathcal{E}[\![\ e_1]\!]\rho\sigma|[D \longrightarrow E](\mathcal{E}[\![\ e_2]\!]\rho\sigma)$

- $\mathcal{C}[\![\ dummy]\!]\rho\sigma = \sigma$
- $\mathcal{C}[\![$ if e then c_1 else $c_2]\!]\rho\sigma = (\mathcal{E}[\![\ e]\!]\rho\sigma|T \longrightarrow \mathcal{C}[\![\ c_1]\!]\rho\sigma, \mathcal{C}[\![\ c_2]\!]\rho\sigma)$
- $\mathcal{C}[\![\ (c_1; c_2)]\!]\rho\sigma = \mathcal{C}[\![\ c_2]\!]\rho(\mathcal{C}[\![\ c_1]\!]\rho\sigma)$
- $\mathcal{C}[\![$ while e do c$]\!]\rho = Fix(\lambda\gamma.\lambda\sigma.(\mathcal{E}[\![\ e]\!]\rho\sigma|T \longrightarrow (\gamma \circ \mathcal{C}[\![\ c]\!]\rho)\sigma, \sigma))$

Exercício 72 1. Defina a semântica para a expressão $\neg e$ e mostre que

$$\mathcal{C}[\![\text{ if e then } c_1 \text{ else } c_2]\!] = \mathcal{C}[\![\text{ if } \neg e \text{ then } c_2 \text{ else } c_1]\!]$$

2. Mostre que $\mathcal{C}[\![$ while e do c$]\!] = \mathcal{C}[\![$ if e then (c; while e do c) else dummy$]\!]$

3. Defina a semântica para o comando until e do c e mostre que

$$\mathcal{C}[\![\text{ while e do c}]\!] = \mathcal{C}[\![\text{ until } \neg e \text{ do c}]\!]$$

4. Defina a semântica para o comando c repeatwhile e e mostre que

$$\begin{aligned} \mathcal{C}[\![\text{ c repeatwhile e}]\!] &= \mathcal{C}[\![\ (c; \text{while e do c})]\!] \\ \mathcal{C}[\![\text{ while e do c}]\!] &= \mathcal{C}[\![\text{ if e then (c repeatwhile e) else dummy}]\!] \end{aligned}$$

5. Mostre que $\mathcal{C}[\![\ (\text{while e do } c_1; \text{while e do } c_2))]\!] = \mathcal{C}[\![\text{ while e then } c_1]\!]$

Exercício 73 [**Funções Estritas**] Reescreva a semântica dada considerando funções estritas. Dê exemplo de alterações no significado de algumas instruções neste caso. Distinga em passagem de parâmetros por *valor* e por *nome*. Nota: Utilize o operador $strict : [A \longrightarrow B] \longrightarrow [A \longrightarrow_\perp B]$, A e B CPOs.

Exercício 74 [**Expressões com "side-effects"**] Distinga entre ambientes e estados; identificadores e endereços; declarações e atribuições. Reescreva a semântica dada (alterando dominios e funções) supondo que as expressões podem alterar o estado e considerando o operador $\star$ tal que se $f \in [E \longrightarrow [S \longrightarrow D]]$ e $g \in [S \longrightarrow [E \times S]]$ então para $s \in S$, $(f \star g)s = (fe)s'$ se $gs = (e, s')$. Neste caso, $\mathcal{E} : Exp \longrightarrow Amb \longrightarrow S \longrightarrow [E \times S]$ e por exemplo,

$$\begin{aligned} \mathcal{E}[\![\ true]\!]\rho &= \lambda\sigma. < t\,em\,E, \sigma > \\ \mathcal{E}[\![\text{ if } e_0 \text{ then } e_1 \text{ else } e_2]\!]\rho &= (\lambda\beta.\beta \longrightarrow \mathcal{E}[\![\ e_1]\!]\rho, \mathcal{E}[\![\ e_2]\!]\rho) \star \mathcal{E}[\![\ e_0]\!]\rho \end{aligned}$$

O domínio E passa a ter mais um somando $[D \longrightarrow W]$, com $W = [S \longrightarrow [E \times S]]$. Em especial, deve considerar as alterações na semântica das funções $(fn\ x.e)$ e na sua aplicação: considere funções *Apply* e *map* tal que:

$$Apply < \epsilon_1, \epsilon_2 >= (\epsilon_1|[D \longrightarrow [S \longrightarrow [E \times S]])(\epsilon_2)$$

e $map < \omega_1, \omega_2 > \sigma =<< \epsilon_1, \epsilon_2 >, \sigma_2 >$ onde $\omega_1\sigma =< \epsilon_1, \sigma_1 >$ e $\omega_2\sigma_1 =< \epsilon_2, \sigma_2 >$ então:

$$\mathcal{E}[\![\ fn\ x.e]\!]\rho\ =\ strict(\lambda\sigma < \lambda\delta.\mathcal{E}[\![\ e]\!]\rho[\delta/x])\ em\ E, \sigma >)$$

$$\mathcal{E}[\![\ e_1(e_2)]\!]\rho\ =\ Apply \star map < \mathcal{E}[\![\ e_1]\!]\rho, \mathcal{E}[\![\ e_2]\!]\rho >$$

Verifique e complete. Considere apenas funções estritas.

Exercício 75 Considerando a semântica definida no exercício 75 com *funções estritas* mostre que $\mathcal{E}[\![\ (fn\ x.e_0)(e_1)]\!] = \mathcal{E}[\![$ let x be e_1 in $e_0]\!]$

Exercício 76 [**Rotinas**] Considerando a semântica definida no exercício 75 estenda a linguagem (e a correspondente semântica) para admitir rotinas (sem nomes nem parâmetros). Considere a expressão $rt\ c$ e o comando $call\ e$.
Nota: Altere convenientemente o domínio dos valores das expressões para admitir "comandos" como valores. Para a chamada da rotina basta definir convenientemente uma função *Run* tal que $\mathcal{C}[\![\ call\ e]\!]\rho\ =\ Run \star \mathcal{E}[\![\ e]\!]\rho$

Exercício 77 Considerando a semântica definida no exercício 77, mostre que

$$\mathcal{C}[\![\ (call(rt\ c))]\!]\ =\ \mathcal{C}[\![\ c]\!]$$

$$\mathcal{C}[\![\ call(fn\ x.rt\ c)(e)]\!]\ =\ \mathcal{E}[\![$ let x be e in $c]\!]$$

Exercício 78 [**Atribuição**] Estenda a liguagem dada considerando o comando $e_1 := e_2$ (atribuição). Modifique os domínios semânticos das expressões para distinguir entre "conteúdos" $\beta \in V$ (*expressões armazenáveis*, pode ser igual a E) e os endereços $\alpha \in L$ (*posições de memória*). O domínio L é um somando de D. Podemos então considerar o domínio dos estados $S = [L \longrightarrow V]$ e considerar as funções auxiliares $cont : L \longrightarrow S \longrightarrow V$ e $update : [L \times V] \longrightarrow S \longrightarrow S$ tal que $cont\,\alpha\sigma = \sigma\alpha$ e $update(\alpha, \beta)\sigma = \sigma[\beta/\alpha]$. Defina a semântica para a atribuição.

Exercício 79 Modifique a semântica da atribuição para o caso de alocação dinâmica de memória. Nota: Considere funções auxiliares para criar ($new : S \longrightarrow L$, $extend : L \longrightarrow S \longrightarrow S$) e libertar ($free : L \longrightarrow S \longrightarrow S$) posições de memória, para testar ($area : L \longrightarrow S \longrightarrow T$) se uma posição está a ser usada num dado estado e para mapear a memória $map \in [L \longrightarrow V]$. Defina as funções *cont* e *update* usando as funções anteriores. Neste caso, $S = [L \longrightarrow v] \times [L \longrightarrow T]$.

Exercício 80 [**Continuações**] Verifique que a semântica dada anteriormente não é adequada para o tratamento de instruções que envolvam transferências de control com eventual mudança do âmbito do ambiente ou estado de avaliação. Considere expressões como *enter e exit* (define um âmbito de avaliação) e *with e escape* que quando avaliada no interior duma expressão como a anterior tranfere o control para o "mais próximo" *exit* e o valor de toda expressão anterior é o de e. Considere comandos como $x{:}c$ (cuja semântica é a de c) e *goto* x, e analise geralmente instruções como $(goto\, x_1; c_1)$. Defina uma semântica de continuações para a linguagem dada estendida com estas novas instruções. Uma continuação especifica uma transformação entre um estado corrente e um estado final do programa. Para uma expressão uma continuação envolve um resultado e um estado $\kappa \in K = [E \longrightarrow [S \longrightarrow S]] = [E \longrightarrow C]$, para um comando envolve uma transformação de funções de estados em funções de estados. Sendo $\theta \in C = [S \longrightarrow S]$ temos para um comando c $\mathcal{C}[\![\ c]\!]\rho\theta \in C$. Por exemplo, defina $\mathcal{C}[\![\ (c_1; c_2)]\!]\rho\theta = \mathcal{C}[\![\ c_1]\!]\rho\{\mathcal{C}[\![\ c_2]\!]\rho\theta\}$ e $\mathcal{E}[\![$ if e_0 then e_1 else $e_2]\!]\rho\kappa = \mathcal{E}[\![\ e_0]\!]\rho\{Cond(\mathcal{E}[\![\ e_1]\!]\rho\kappa, \mathcal{E}[\![\ e_2]\!]\rho\kappa)\})$ onde $Cond : [C \times C] \longrightarrow K$ é definida por $Cond(\theta_1, \theta_2) = \lambda v.isT(v) \longrightarrow (v|T) \longrightarrow \theta_1, \theta_2), wrong$ e $wrong \in C$ é a continuação associada a uma situação de erro.

Exercício 81 Considerando a semântica de continuações dada no exercício anterior defina a semântica para o comando *stop* e mostre que $(c_1; stop; c_2)$ é equivalente a $(c_1; stop)$.

Exercício 82 Considerando a semântica de continuações dada nos exercícios anteriores defina a semântica para a atribuição $e_1 := e_2$. Redefina as funções *cont*, *update*, $free$ e *extend*.

Exercício 83 Modifique o domínio dos estados para considerar comandos de entrada e saída, como por exemplo *read* x e *write* e. Defina uma semântica para estes novos comandos, considerando deteção de erros.

Algumas notas sobre a resolução dos exercícios

Exercicio 73

1. Seja $\mathcal{E}[\![\ \neg e]\!]\rho\sigma\ =\ (\mathcal{E}[\![\ e]\!]\rho\sigma|\ T\ \longrightarrow$ f em E, t em E)

   $$\begin{aligned}&\mathcal{C}[\![\text{ if } \neg e \text{ then } c_2 \text{ else } c_1]\!]\\ &=\ \mathcal{E}[\![\ \neg e]\!]\rho\sigma|T\ \longrightarrow\ \mathcal{C}[\![\ c_2]\!]\rho\sigma, \mathcal{C}[\![\ c_1]\!]\rho\sigma\\ &=\ (\mathcal{E}[\![\ e]\!]\rho\sigma|\ T\ \longrightarrow \text{ f em E, t em E})|T\ \longrightarrow\ \mathcal{C}[\![\ c_2]\!]\rho\sigma, \mathcal{C}[\![\ c_1]\!]\rho\sigma\end{aligned}$$

   Então, se $\mathcal{E}[\![\ e]\!]\rho\sigma =$ t em E, o resultado é $\mathcal{C}[\![\ c_1]\!]\rho\sigma$ e caso contrário $\mathcal{C}[\![\ c_2]\!]\rho\sigma$, como se pretendia.

2. Basta verificar que a semântica de while e do c foi definida para que essa igualdade se verificasse. Sendo $H = \lambda\gamma.\lambda\sigma.(\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ (\gamma \circ \mathcal{C}[\![\ c]\!]\rho)\sigma, \sigma)$ e $\mathcal{C}[\![$ while e do c$]\!]\rho = \gamma'$ então

   $$\begin{aligned}&\gamma' = H(\gamma') =\\ &\quad(\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ (\gamma' \circ \mathcal{C}[\![\ c]\!]\rho)\sigma, \sigma) =\\ &\quad\mathcal{C}[\![\text{ if } e \text{ then (c; while } e \text{ do } c) \text{ else dummy}]\!]\rho\end{aligned}$$

3. Pretende-se que $\mathcal{C}[\![$ until e do c$]\!] = \mathcal{C}[\![$ if e then dummy else (c; until e do c)$]\!]$, então $\mathcal{C}[\![$ until e do c$]\!]\rho = Fix(\lambda\gamma.\lambda\sigma.(\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ \sigma, (\gamma \circ \mathcal{C}[\![\ c]\!]\rho)\sigma))$

4. Pretende-se que $\mathcal{C}[\![$ c repeatwhile e$]\!] = \mathcal{C}[\![$ c; if e then c repeatwhile e else dummy$]\!]$ então $\mathcal{C}[\![$ c repeatwhile e$]\!]\rho = Fix(\lambda\gamma.(\lambda\sigma.\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ \gamma\sigma, \sigma) \circ \mathcal{C}[\![\ c]\!]\rho)$

   Para mostrar que

   $$\mathcal{C}[\![\text{ c repeatwhile e}]\!]\rho\ =\ \mathcal{C}[\![\ (c; \text{while e do c})]\!]\rho \tag{7.1}$$

   $$\mathcal{C}[\![\text{ while e do c}]\!]\rho\ =\ \mathcal{C}[\![\text{ if e then (c repeatwhile e) else dummy}]\!]\rho \tag{7.2}$$

   Sejam

   $$\begin{aligned}H_0\ &=\ \lambda\gamma.\lambda\sigma.(\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ (\gamma \circ \mathcal{C}[\![\ c]\!]\rho)\sigma, \sigma)\\ H_1\ &=\ \lambda\gamma.(\lambda\sigma.\mathcal{E}[\![\ e]\!]\rho\sigma|T\ \longrightarrow\ \gamma\sigma, \sigma) \circ \mathcal{C}[\![\ c]\!]\rho\end{aligned}$$

   $\mathcal{C}[\![$ c repeatwhile e$]\!]\rho = Fix(H_1) = \gamma_1$ e $\mathcal{C}[\![$ while e do c$]\!]\rho = Fix(H_0) = \gamma_0$. Se $\gamma' = \mathcal{C}[\![\ c]\!]\rho$, $\omega = \mathcal{E}[\![\ e]\!]\rho$ e $\gamma_2 = \lambda\sigma.\,\omega\sigma|T\ \longrightarrow\ \gamma_1\sigma, \sigma$ as igualdades acima ficam:

   $$\gamma_1 = \gamma_0 \circ \gamma' \tag{7.3}$$

   $$\gamma_0 = \gamma_2 \tag{7.4}$$

Se $\gamma_i = Fix(H_i)$ e se se mostar que $H_i(\gamma_j) = \gamma_j$ $(i \neq j)$ tem-se que $\gamma_i \sqsubseteq \gamma_j$ e se o recíproco também se verificar concluí-se que $\gamma_i = \gamma_j$.

$$\begin{aligned}\gamma_1 &= H_1(\gamma_1)\\ &= \lambda\sigma.\omega\sigma \longrightarrow \gamma_1\sigma, \sigma) \circ \gamma'\\ &= \gamma_2 \circ \gamma'\end{aligned}$$

e

$$\begin{aligned}H_0(\gamma_2) &= \lambda\sigma.\omega\sigma \longrightarrow (\gamma_2 \circ \gamma')\sigma, \sigma\\ &= \lambda\sigma.\omega\sigma \longrightarrow \gamma_1\sigma, \sigma\\ &= \gamma_2\end{aligned}$$

Logo, $\gamma_0 \sqsubseteq \gamma_2$.

$$\begin{aligned}H_1(\gamma_0 \circ \gamma') &= (\lambda\sigma.\omega\sigma \longrightarrow (\gamma_0 \circ \gamma')\sigma, \sigma) \circ \gamma'\\ &= H_0(\gamma_0) \circ \gamma'\\ &= \gamma_0 \circ \gamma'\end{aligned}$$

Logo $\gamma_1 \sqsubseteq \gamma_0 \circ \gamma'$. Temos provado uma das partes de cada igualdade. Agora,

$$\gamma_2 \sqsubseteq \lambda\sigma.\, \omega\sigma|T \longrightarrow (\gamma_0 \circ \gamma')\sigma, \sigma = H_0(\gamma_0) = \gamma_0$$

e, então $\gamma_1 = \gamma_2 \circ \gamma' = \gamma_0 \circ \gamma'$.

Exercício 77 Considere-se

$$\mathcal{E}[\![\ rt\ c]\!]\rho\ =\ strict(\lambda\sigma.\ <\mathcal{C}[\![\ c]\!]\rho\ em\ E, \sigma>)$$

$$\mathcal{C}[\![\ call\ e]\!]\rho\ =\ Run \star \mathcal{E}[\![\ e]\!]\rho$$

onde $Run(\epsilon) = strict(\epsilon \mid C)$

Exercício 79

$$\mathcal{C}[\![\ e_1 := e_2]\!]\rho\sigma = update(\alpha, \beta)\sigma$$

onde $\alpha = \mathcal{E}[\![\ e_1]\!]\rho\sigma \mid L$ e $\beta = \mathcal{E}[\![\ e_2]\!]\rho\sigma \mid V$

Exercício 80 Define-se $map(\sigma) = \pi_1(\sigma)$ e $area(\sigma) = \pi_2(\sigma)$. Temos para $\alpha \in L$ e $\sigma \in S$

$$\begin{aligned}
cont(\alpha)(\sigma) &= area(\sigma)(\alpha) \longrightarrow map(\sigma)(\alpha), \underline{undef}\\
update(\alpha, \beta)\sigma &= \sigma[\beta/\alpha] \text{ onde } map(\sigma[\beta/\alpha]) = \lambda\alpha'.(\alpha = \alpha') \longrightarrow \beta, map(\sigma)\alpha \text{ e } area(\sigma[\beta/\alpha]) = area(\sigma)\\
area(\sigma)(new(\sigma)) &= false\\
map \circ extend(\alpha) &= map\\
area(extend(\alpha)\sigma)(\alpha') &= (\alpha = \alpha') \longrightarrow t\,em\,V, area(\sigma)\alpha'\\
map \circ free(\alpha) &= map\\
area(free(\alpha)\sigma)(\alpha') &= (\alpha = \alpha') \longrightarrow f\,em\,V, area(\sigma)\alpha'
\end{aligned}$$

Exercício 81 Apresentamos a semântica de continuações para o seguinte sub-conjunto:

$$\begin{aligned} e &::= x \mid n \mid true \mid false \mid o_0() \mid o_1(e) \mid o_2(e,e) \mid \cdots \mid \\ &\quad \text{if e then e else e} \mid \text{let x be e in e} \mid \text{fn x.e} \mid \\ &\quad e(e) \\ c &::= dummy \mid \text{if e then c else c} \mid (c;c) \mid \text{while e do c} \\ &\quad \mid \text{goto e} \mid \text{begin } x_1 : c_1 \ldots x_n : c_n \text{ end} \end{aligned}$$

$A = S$

$F = [D \longrightarrow K \longrightarrow C]$

$\theta \in C = [S \longrightarrow A]$

$\epsilon \in E = [T + N + C + F]$

$E = D$

$\kappa \in K = [E \longrightarrow C]$

$\rho \in Amb = [Id \longrightarrow D]$

$\mathcal{E}' : Exp \longrightarrow Amb \longrightarrow K \longrightarrow C$

$\mathcal{O}'_n : Op \longrightarrow [E^n \longrightarrow [K \longrightarrow C]$

$\mathcal{C}' : Cmd \longrightarrow Amb \longrightarrow C \longrightarrow C$

$wrong \in C$

$Cond : [C \times C] \longrightarrow K$ é definida por

$$Cond(\theta_1, \theta_2) = \lambda v.(v \text{ em } T \longrightarrow (v|T) \longrightarrow \theta_1, \theta_2), wrong$$

- $\mathcal{E}'[\![x]\!]\rho\kappa = \rho[\![x]\!] = \underline{undef} \longrightarrow wrong, \kappa(\rho[\![x]\!])$
- $\mathcal{E}'[\![n]\!]\rho\kappa = \kappa(n\, em\, E)$
- $\mathcal{E}'[\![true]\!]\rho\kappa = \kappa(t\, em\, E)$
- $\mathcal{E}'[\![false]\!]\rho\kappa = \kappa(f\, em\, E)$
- $\mathcal{E}'[\![o_n(e_1, \cdots, e_n)]\!]\rho\kappa = \mathcal{E}'[\![e_1]\!]\rho\{\lambda\epsilon_1.\mathcal{E}'[\![e_2]\!]\rho\{\ldots \lambda\epsilon_{n-1}\mathcal{E}'[\![e_n]\!]\rho\{\lambda\epsilon_n.O'_{\mathrm{n}}(o_n) < \epsilon_1, \ldots, \epsilon_n > \kappa\}\ldots\}$ onde

$$O'_{\mathrm{n}}(o_n) < \epsilon_1, \ldots, \epsilon_n > \kappa = \eta\, em\, N \longrightarrow \kappa(\eta\, em\, E), wrong$$
$$\text{e } \eta = O_{\mathrm{n}}(o_n)(\epsilon_1|N, \ldots, \epsilon_n|N)$$

- $\mathcal{E}'[\![\text{if } e_0 \text{ then } e_1 \text{ else } e_2]\!]\rho\kappa = \mathcal{E}'[\![e_0]\!]\rho\{Cond(\mathcal{E}'[\![e_1]\!]\rho\kappa, \mathcal{E}'[\![e_2]\!]\rho\kappa)\}$

- $\mathcal{E}'[\![$ let x be e_1 em $e_2]\!]\rho\kappa = \mathcal{E}'[\![e_1]\!]\rho\{\lambda\delta.\mathcal{E}'[\![e_1]\!](\rho[\delta/x])\kappa\}$
- $\mathcal{E}'[\![$ fn x.e$]\!]\rho\kappa = \kappa(\lambda v.\lambda\kappa'.\mathcal{E}'[\![e]\!]\rho[v/x]\{\kappa'\}\, em\, E)$
- $\mathcal{E}'[\![e_1(e_2)]\!]\rho\kappa = \mathcal{E}[\![e_1]\!]\rho\kappa_1$
  onde $\kappa_1(\epsilon_1) = \mathcal{E}[\![e_2]\!]\rho\kappa_2$ e $\kappa_2(\epsilon_2) = (\epsilon_2 \mid F)(\epsilon_1 \mid D)\{\kappa\}$
- $\mathcal{C}'[\![dummy]\!]\rho\theta = \mathbf{I}$, onde $\mathbf{I} = \lambda\theta.\theta$
- $\mathcal{C}'[\![$ if e then c_1 else $c_2]\!]\rho\theta = \mathcal{E}'[\![e]\!]\rho\{Cond(\mathcal{C}'[\![c_1]\!]\rho\theta, \mathcal{C}'[\![c_2]\!]\rho\theta)\}$
- $\mathcal{C}'[\![(c_1; c_2)]\!]\rho\theta = \mathcal{C}'[\![c_1]\!]\rho\{\mathcal{C}'[\![c_2]\!]\rho\theta\}$
- $\mathcal{C}'[\![$ while e do c$]\!]\rho\theta = Fix(\lambda\theta'.\mathcal{E}'[\![e]\!]\rho\{Cond(\mathcal{C}'[\![c]\!]\rho\theta', \theta)\})$
- $\mathcal{C}'[\![$ goto e$]\!]\rho\theta = \mathcal{E}'[\![e]\!]\rho\{Jump\}$
  onde $Jump(\epsilon) = \epsilon\, em\, C \longrightarrow \epsilon \mid C, wrong$
- $\mathcal{C}'[\![$ begin $x_1 : c_1 \ldots x_n : c_n$ end$]\!]\rho\theta =$
  $(Fix(\lambda < \theta_1, \ldots, \theta_n > . < \mathcal{C}'[\![c_1]\!]\rho'\theta_2, \ldots, \mathcal{C}'[\![c_n]\!]\rho'\theta >)) \downarrow 1$ onde,

$$\rho' = \rho[\theta_1\, em\, D/x_1] \ldots [\theta_n\, em\, D/x_n]$$

Usando a semântica anterior, mostre que

$$\begin{array}{lll} begin & x_1 : c_{11}; & \\ & & \text{if e then goto } x_3 \text{ else dummy;} \\ & & \text{goto } x_4; \\ & x_3 : c_{13}; & \\ & & \text{goto } x_5; \\ & x_4 : c_{14}; & \\ & x_5 : c_{15}; & \\ end & & \end{array}$$

e

$$c_{11}; (\text{if e then } c_{13} \text{ else } c_{14}); c_{15}$$

têm a mesma semântica.

Exercício 83 Considerem-se as funções:

$$\begin{array}{lcl} Cont\,\alpha\kappa\sigma & = & area(\sigma)\alpha \longrightarrow \kappa(map(\sigma)\alpha), wrong\sigma \\ Update\,\alpha\beta\theta & = & (isL(\alpha) \longrightarrow isV(\beta) \longrightarrow \theta \circ update(\alpha, \beta), wrong), wrong \\ Extend\,\kappa\sigma & = & new(\sigma) = \underline{undef} \longrightarrow wrong\sigma, \kappa(new(\sigma)(extend(new\sigma)\sigma) \\ Free\,\alpha\theta\sigma & = & area(\sigma)\alpha \longrightarrow \theta(free\,\alpha\sigma), wrong\sigma \end{array}$$

Definem-se as funções semânticas para cada $x \in Id$ tal que $\rho[\![\ x]\!] = (\alpha\, em\, D$ para $\alpha \in L$,

$$\mathcal{L}[\![\ x]\!]\rho\kappa = k\alpha \text{ se } \rho[\![\ x]\!] = \alpha\, em\, D \text{ e } \mathcal{E}[\![\ x]\!]\rho\kappa = \kappa(\alpha\, em\, E)$$

$$\mathcal{R}[\![\ x]\!]\rho\kappa = Cont\alpha\kappa$$

Se $x' \in Id$ for tal que, por exemplo, $\rho[\![\ x']\!] = n\, em\, D$ para $n \in \mathbf{N}$,

$$\mathcal{R}[\![\ x']\!]\rho\kappa = \kappa\, n \text{ e } \mathcal{E}[\![\ x']\!]\rho\kappa = \kappa(n\, em\, E)$$

$$\mathcal{L}[\![\ x']\!]\rho\kappa = wrong$$

ou interpretar como encontrar uma posição de memória e colocar lá n.

$$\mathcal{L}[\![\ x']\!]\rho\kappa = \mathcal{E}[\![\ x']\!]\rho\{\lambda\epsilon.Extend\{\lambda\alpha.Update\,\alpha(\epsilon \mid V)\{\kappa\alpha\}\}\}$$

Para definir $\mathcal{L}$ e $\mathcal{R}$ à custa de $\mathcal{E}$, considerem-se

$$lv : [L \longrightarrow C] \longrightarrow [E \longrightarrow C]$$

$$rv : [V \longrightarrow C] \longrightarrow [E \longrightarrow C]$$

e

$$\mathcal{L}[\![\ e]\!]\rho\kappa = \mathcal{E}[\![\ e]\!]\rho\{lv\,\kappa\} \quad \mathcal{R}[\![\ e]\!]\rho\kappa = \mathcal{E}[\![\ e]\!]\rho\{rv\,\kappa\}$$

com

$$\begin{aligned} lv = \lambda\kappa\lambda\epsilon.isC(\epsilon) &\longrightarrow \kappa(\epsilon \mid L),\\ isV(\epsilon) &\longrightarrow Extend\{\lambda\alpha.Update\,\alpha(\epsilon \mid V)\{\kappa\alpha\}\}, wrong\\ rv = \lambda\kappa\lambda\epsilon.isL(\epsilon) &\longrightarrow Cont(\epsilon \mid L)\kappa,\\ isV(\epsilon) &\longrightarrow \kappa(\epsilon \mid V), wrong \end{aligned}$$

Vem,

$$\mathcal{C}[\![\ e_1 := e_2]\!]\rho\theta = \mathcal{L}[\![\ e_1]\!]\rho\{\lambda\alpha.\mathcal{R}[\![\ e_2]\!]\rho\{\lambda\beta.\,Update\,\alpha\beta\theta\}\}$$

Capítulo 8

Tipos

8.1 λ-Cálculo Tipificado

No λ-Cálculo tipificado todos os λ-termos têm um tipo. Fixa-se um conjunto de tipos básicos a partir dos quais outros tipos são obtidos por construtores cuja forma depende não só das operações existentes para formar λ-termos mas também da complexidade do sistema de tipos que se pretende obter. Consideremos um conjunto de tipos:

Definição 42 *O conjunto dos tipos T é definido indutivamente por:*

1. *Os tipos básicos estão em T (constantes)*
2. *$\alpha_1, \alpha_2, \ldots \in T$ são variáveis de tipo*
3. *Se τ e $\sigma \in T$ então $\tau \longrightarrow \sigma \in T$*

Os tipos básicos podem ser os inteiros, os reais ou os valores booleanos; estes tipos são representados respectivamente pelas constantes **int**, **real** e **boolean**. Supomos que $\longrightarrow$ é associativa à direita.

Definição 43 *$M : \sigma$ onde $\sigma \in T$ e M um λ-termo indica que M tem tipo σ*

Definição 44 *Regras de inferência de tipos*

1. *Se $x : \tau$ e $N : \sigma$ então $\lambda x.N : \tau \longrightarrow \sigma$*
2. *Se $M : \tau \longrightarrow \sigma$ e $N : \tau$ então $MN : \sigma$*

Estas regras permitem determinar o tipo dum λ-termo. Por exemplo se no termo $\lambda x.suc(x)$ soubermos que suc:**int** $\longrightarrow$ **int**, então o tipo de x pode ser inferido como sendo **int**.

Relativamente ao λ-Cálculo sem tipos, as noções de variável livre e ligada, renomeação e substituição mantêm-se no λ-Cálculo tipificado. Para redução β

$$(\lambda x.P)Q \longrightarrow [Q/x]P$$

é necessário que Q e x tenham o mesmo tipo.

Lema 5 *Na redução β o β-redex $(\lambda x.P)Q$ tem o mesmo tipo que a sua contração $[Q/x]P$*

Exercício 84 *Demonstre o lema anterior por indução na estrutura de P*

Teorema 6 *Se $E : \tau$ e N é a forma normal de E então $N : \tau$*

Note-se que neste sistema de tipos nem todos os λ-termos são válidos. Por exemplo os operadores de ponto fixo, como Y, não estão definidos uma vez que

$$\forall \tau \in T \; \tau \neq \tau \longrightarrow \tau$$

Isto leva a que as definições recursivas tenham de ser tratadas de outros modos. Contudo, podemos concluir que todos os λ-termos tipificados em T têm forma normal.

Exercício 85 *Mostre que o λ-termo $\lambda x.xx$ não admite tipo em T.*

Exercício 86 *Determine o tipo dos seguintes λ-termos em T:*

1. $\lambda xy.x$
2. $\lambda x.y$ *se* $y : \tau$
3. $\lambda xy.xyy$

Exercício 87 *Mostre que em T:*

1. **B**:$(\tau \longrightarrow \rho) \longrightarrow (\sigma \longrightarrow \tau) \longrightarrow \sigma \longrightarrow \rho$
2. **S**: $((\sigma \longrightarrow (\tau \longrightarrow \rho)) \longrightarrow (\sigma \longrightarrow \tau) \longrightarrow \sigma \longrightarrow \rho)$

8.1.1 Semântica denotational de tipos

Consideremos um conjunto de tipos T análogo ao anterior, sendo Tv o conjunto de variáveis de tipo α e Bt o conjunto dos tipos básicos ι, então a sintaxe dos tipos é dada por:

$$\tau ::= \alpha \mid \iota \mid \tau \longrightarrow \tau'$$

Definição 45 *Uma substituição S de tipos para variáveis de tipo é uma função de Tv em T. Sendo $\alpha_1, \ldots, \alpha_n \in Tv$ e $\tau_1, \ldots, \tau_n \in T$*

$$[\tau_1/\alpha_1, \ldots, \tau_n/\alpha_n]$$

denota a substituição que a cada α_i faz corresponder τ_i, e que para as outras variáveis faz corresponder elas próprias.
A composição de duas substituições R e S é definida por

$$(SR)\alpha = S(R\alpha)$$

e é associativa.
Uma susbstituição S estende-se a tipos substituindo cada ocorrência duma variável α num tipo por $S\alpha$.
Um tipo τ' é uma instância dum tipo τ se existir uma substituição S tal que $\tau' = S\tau$. Se τ e τ' são instâncias um do outro diz-se que τ' é uma instância trivial de τ

Definição 46 (Ideal dum cpo) *Um subconjunto não vazio I dum cpo V é um ideal se e só se*

1. *Se $v \in I$ e $v' \sqsubseteq v$ então $v' \in I$*
2. *O supremo de qualquer cadeia ascendente em I está em I*

Os ideais de domínios são um bom modelo para tipos. Seja $\overline{V}$ o conjunto de todos os ideais dum cpo V, ordenados com a relação

$$I \sqsubseteq I' \equiv I' \text{ é um subconjunto de } I$$

Dados I e I' em $\overline{V}$ o conjunto

$$I \longrightarrow I' = \{ v \in V \mid is[V \longrightarrow V](v)\, e\, \forall v' \in I,\ (v\ |_{[V \longrightarrow V]})v' \in I' \}$$

está em $\overline{V}$
Uma validação de variáveis de tipo é uma função ϕ que atribui um elemento de $\overline{V}$ a cada variável de tipo. Seja $Tval$ o domínio das validações

$$Tval = Tv \longrightarrow \overline{V}$$

e dado $\xi : Bt \longrightarrow \overline{V}$, define-se a função semântica para tipos:

$$T : Tv \longrightarrow Tval \longrightarrow \overline{V}$$

por:

$$\begin{aligned} T[\![\iota]\!]\phi &= \xi[\![\iota]\!] \\ T[\![\alpha]\!]\phi &= \phi[\![\alpha]\!] \\ T[\![\tau \longrightarrow \tau']\!]\phi &= (T[\![\tau]\!]\phi) \longrightarrow (T[\![\tau']\!]\phi) \end{aligned}$$

Definição 47 *1.*

$$v :_\phi \tau \equiv v \in T[\![\tau]\!]\phi$$

2. *Dado um conjunto de asserções da forma $x : \tau$, podemos estender a relação anterior a ambientes e conjuntos de asserções:*

$$\rho :_\phi A \equiv \forall x : \tau \in A \; \rho[\![x]\!] :_\phi \tau$$

3. *O significado de asserções da forma $A \models e : \tau$ é dado por:*

$$A \models e : \tau \equiv \forall \phi \in Tval \, \forall \rho \in Amb \; \rho :_\phi A \implies \mathcal{E}[\![e]\!]\rho :_\phi \tau$$

Vamos considerar uma linguagem de expressões que é o λ-Cálculo estendido com expressões da forma:

$$let\, x = e\, in\, e'$$

Definição 48 (Inferência de tipos) *Sendo A um conjunto de asserções diz-se que se pode inferir o tipo τ para e de A, e escreve-se $A \vdash e : \tau$ sse isso pode ser derivado das regras de inferência seguintes, onde A_x denota o resultado de excluir de A qualquer asserção acerca de x:*

TAUT:

$$A \vdash x : \tau \quad (x : \tau \, in \, A)$$

COMB:

$$\frac{A \vdash e : \tau \longrightarrow \tau' \; A \vdash e' : \tau}{A \vdash ee' : \tau'}$$

ABS:

$$\frac{A_x \cup x : \tau' \vdash e : \tau}{A \vdash \lambda x.e : \tau' \longrightarrow \tau}$$

LET:

$$\frac{A \vdash e : \tau_1, \ldots, A \vdash e : \tau_n, \; A_x \cup x : \tau_1, ..., x : \tau_n \vdash e' : \tau}{A \vdash \; let \; x{=}e \; in \; e' : \tau}$$

Exercício 88 *Mostre que*

$$\vdash \textit{(let } i{=}\lambda\ x.x \textit{ in } ii) : \alpha \longrightarrow \alpha$$

Teorema 7 *Para qualquer expressão e, tipo τ e asserções A se*

$$A \vdash e : \tau$$

então

$$A \models e : \tau$$

Exercício 89 *Demonstre o teorema anterior por indução estrutural na árvore de derivação de $A \vdash e : \tau$*

8.1.2 Algoritmo de inferência de tipos

Seja U o unificador de dois tipos τ e τ' (substituição tal que $U\tau = U\tau'$) . Dada uma expressão algoritmo seguinte retorna, quando sucede, um tipo e um conjunto de asserções.

Definição 49 *$T(e) = (A, \tau)$, onde*

1. *Se e é x então $A = \{x : \beta\}$ e $\tau = \beta$, onde β é uma nova variável de tipo*
2. *Se e é $(e_1 e_2)$, seja $(A_i, \tau_i) = T(e_i)$ e U o unificador de τ_1 e $\tau_2 \longrightarrow \beta$ onde β é uma variável nova, então, $A = U(A_1 \cup A_2)$ e $\tau = U\,\beta$*
3. *Se e é $\lambda x.e'$ seja $(A', \tau') = T(e')$, então:*
   - *se não existem asserções sobre x em A' então: $A = A'$ e $\tau = \beta \longrightarrow \tau'$, onde β é nova.*
   - *se existe uma só asserção $x : \nu$ em A', então: $A = A'_x$ e $\tau = \nu \longrightarrow \tau'$.*
   - *se existem asserções $x : \nu_i$ em A', seja U o unificador de ν_i então, $A = U\,A'_x$ e $\tau = U(\nu_1 \longrightarrow \tau')$.*
4. *Se e é let $x{=}e_1$ in e_2, seja $(A_2, \tau_2) = T(e_2)$, então:*
   - *se não existem asserções sobre x em A_2 então: $A = A_2$ e $\tau = \tau_2$.*
   - *se existem asserções $x : \nu_i$ em A_2, seja $(A_1, \tau_1) = T(e_1)$ e $(A_1^1, \tau_1^1), \ldots, (A_1^n, \tau_1^n)$, variantes triviais de $(A_1, \tau_1) = T(e_1)$ e U o unificador de ν_i e τ_{1_i}, então, $A = U\,(A_{2_x} \cup A_1^1 \ldots \cup A_1^n)$ e $\tau = U\tau_2$.*

Note que T falha se não existir unificador U.

Exercício 90 *Use o algoritmo anterior para determinar o tipo de*

(let $i{=}\lambda x.fx$ in ii)

Definição 50 (Esquema de tipos) *Um esquema de tipos η ou é um tipo τ ou um termo da forma:*

$$\forall\alpha\,\eta$$

onde τ é um tipo e α é uma variável de tipo, que será chamada variável genérica de η

Definição 51 (Instância genérica) *ν é uma instância genérica dum esquema de tipo η, e escreve-se $\eta > \nu$, se η é ν ou se η é da forma $\forall\alpha\,\eta'$ e existe um tipo τ tal que $\nu = [\tau/\alpha]\eta'$*

Note-se que se pode abreviar

$$\forall\alpha_1 \ldots \forall\alpha_n\,\tau$$

para

$$\forall\alpha_1 \ldots \alpha_n\,\tau$$

Definição 52 (Inferência de esquemas de tipos) *Pode-se estender o conjunto de regras de inferência :*

TAUT:

$$A \vdash x : \eta \quad (x : \eta\, in\, A)$$

INST:

$$\frac{A \vdash e : \eta}{A \vdash e : \eta'}\ (\eta > \eta')$$

GEN:

$$\frac{A \vdash e : \eta}{A \vdash e : \forall\alpha\,\eta}\ (\alpha \textit{não ocorre livre em} A)$$

COMB:

$$\frac{A \vdash e : \tau \longrightarrow \tau'\ A \vdash e' : \tau}{A \vdash ee' : \tau'}$$

ABS:

$$\frac{A_x \cup x : \tau' \vdash e : \tau}{A \vdash \lambda x.e : \tau \longrightarrow \tau}$$

LET:

$$\frac{A \vdash e : \eta,\ A_x \cup x : \eta \vdash e' : \tau}{A \vdash \textit{let } x{=}e \textit{ in } e' : \tau}$$

O algoritmo de inferência de tipos pode ser estendido para esquemas de tipos.