Semântica Formal

Carlos A. P. Campani

22 de julho de 2005

Súmula

1. Introdução
2. Abordagens de semântica formal
3. Linguagem exemplo While
4. Sintaxe abstrata

5. Revisão de Prova de Teoremas
   (a) Prova de Implicação
   (b) Prova por Redução ao Absurdo
   (c) Prova por Indução
      i. Indução Sobre os Naturais
      ii. Indução Sobre o Tamanho de uma Seqüência
      iii. Indução Estrutural

6. Semântica de expressões
   (a) Números e Numerais
   (b) Descrevendo Estados
   (c) Expressões aritméticas
   (d) Expressões booleanas
7. Propriedades da Semântica
   (a) Variáveis Livres
   (b) Substituição

8. Semântica Operacional
   (a) Semântica Natural
      i. Propriedades da Semântica
      ii. Função Semântica
   (b) Semântica Operacional Estruturada
      i. Propriedades da Semântica
      ii. Função Semântica
   (c) Um Resultado de Equivalência

9. Semântica Denotacional
   (a) Composicionalidade
   (b) Ponto Fixo
   (c) Definição da Semântica
   (d) Requisitos sobre o Ponto Fixo
   (e) Teoria de Pontos Fixos
   (f) Existência
   (g) Um Resultado de Equivalência

10. Semântica Axiomática
    (a) Provas Diretas de Correção de Programas
        i. Semântica Natural
        ii. Semântica Operacional Estruturada
        iii. Semântica Denotacional
    (b) Asserções para Correção Parcial
    (c) Correção e Completude do Sistema Formal
    (d) Asserções para Correção Total

Bibliografia

- Nielson, H. & Nielson, F. *Semantics with Applications: a formal introduction.* disponível em: `http://www.daimi.au.dk/~bra8130/Wiley_book/wiley.html`;
- lâminas para impressão: `http://www.ufpel.edu.br/~campani/semantica4.ps.gz`.

1 Introdução

Semântica formal preocupa-se em especificar rigorosamente o significado ou comportamento de programas, partes de hardware, etc.

Sintaxe descreve as estruturas de uma linguagem;

Semântica descreve o significado destas estruturas.

A necessidade de uma semântica formal (matemática) para linguagens de programação justifica-se pois:

- pode revelar ambigüidades na definição da linguagem (o que uma descrição não formal não permitiria revelar);
- é uma base para implementação (síntese), análise e verificação formal.

2 Abordagens

Semântica Operacional Significado de uma construção da linguagem é especificado pela computação que ela induz quando executada em uma máquina hipotética (interessa *como* o efeito da computação é produzido);

Semântica Denotacional Significados modelados por objetos matemáticos que representam o efeito de executar uma estrutura (somente o *efeito* interessa, não como é produzido);

Semântica Axiomática Especifica propriedades do efeito da execução das estruturas como *asserções* (alguns aspectos da execução são ignorados).

Tipos de Semântica Operacional:

- Semântica Operacional Estruturada (especifica mais detalhes da execução);
- Semântica Natural (simplifica a notação e esconde detalhes, ao tomar um passo maior).

3 Linguagem Exemplo While

$$n \in \mathbf{Num}$$

$$x \in \mathbf{Var}$$

$$a \in \mathbf{Aexp}$$

$$b \in \mathbf{Bexp}$$

$$S \in \mathbf{Stm}$$

$$a ::= n|x|a_1 + a_2|a_1 * a_2|a_1 - a_2$$

$$b ::= \mathbf{true}|\mathbf{false}|a_1 = a_2|a_1 \leq a_2|\neg b|b_1 \wedge b_2$$

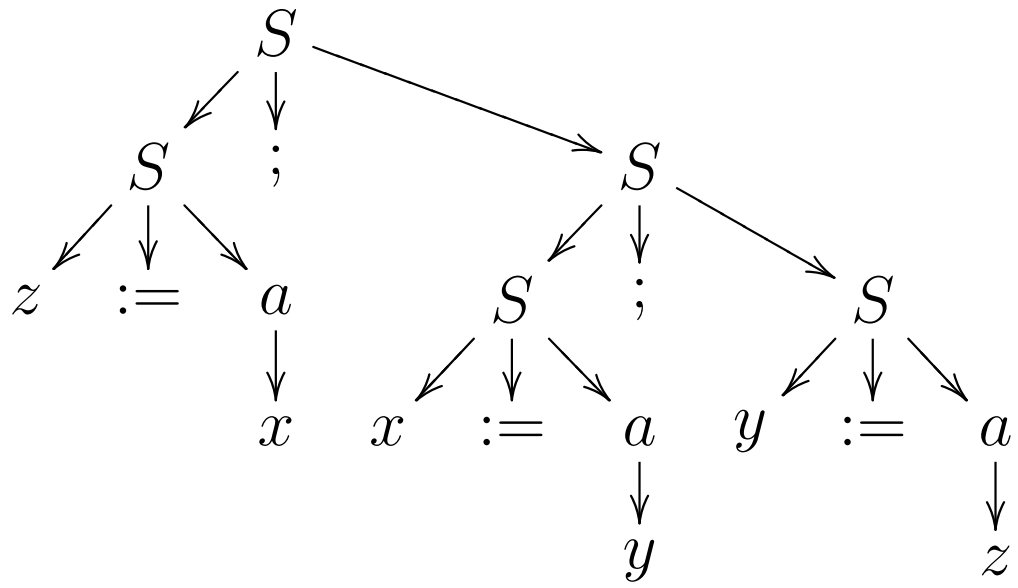
$$S ::= x{:=}a|\mathbf{skip}|S_1; S_2|\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2|\mathbf{while}\ b\ \mathbf{do}\ S$$

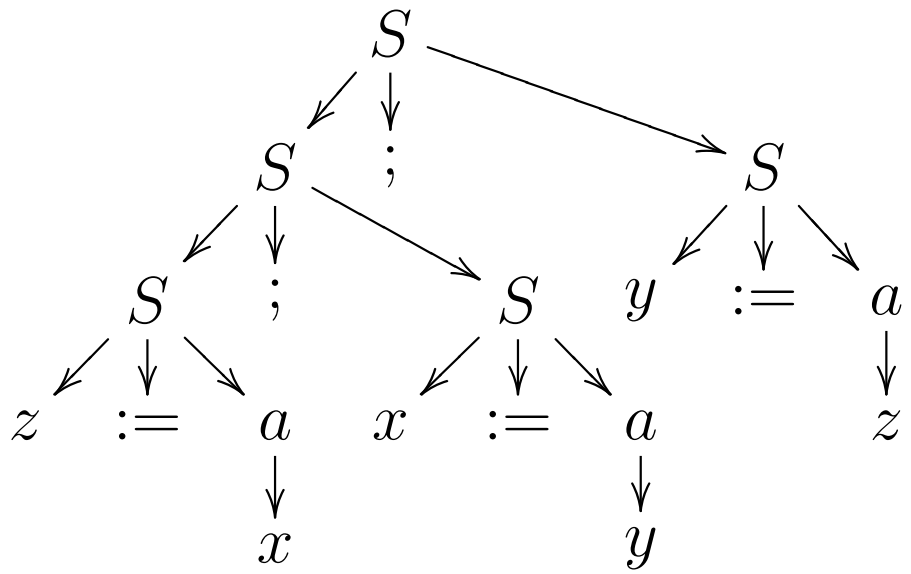
ex1: $z{:=}x; x{:=}y; y{:=}z$

ex2: $y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1)$

4 Sintaxe Abstrata

$$z{:=}x; x{:=}y; y{:=}z$$




S
S
;
S
S
;
S
y
:=
a
z
:=
a
x
:=
a
z
x
y

Usando notação linear:

- $z{:=}x; (x{:=}y; y{:=}z)$ (mais à direita);
- $(z{:=}x; x{:=}y); y{:=}z$ (mais à esquerda).

Exercício: Construa a árvore sintática de
$y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1)$.

5 Revisão de Prova de Teoremas

5.1 Prova de Implicação

Para provar $A \rightarrow B$, assuma A (hipótese) e deduza B.

$$\frac{\begin{array}{l} A \text{ (hipótese)} \\ \ldots \\ \ldots \\ \vdots \\ B \end{array}}{A \rightarrow B}$$

ex: Provar que se um número inteiro é divisível por 6 então também o é por 2.

Seja $n \in \mathbf{Z}$, tal que n é divisível por 6:

1. n é divisível por 6 (hipótese);
2. $n = 6.k$, onde $k \in \mathbf{N}$;
3. $n = 2.3.k$;
4. $n = 2.k'$, onde $k' = 3.k$ e $k' \in \mathbf{N}$
5. n é divisível por 2;
6. Logo, vale o enunciado.

5.2 Prova por Redução ao Absurdo

Para provar A, suponha $\neg A$ e mostre que isto resulta em uma contradição (absurdo).

ex: Provar que se um número somado a si mesmo resulta o número original, então este número é zero.

Suponha que este número é x e que:

$$x + x = x \wedge x \neq 0$$

então

$$2x = x \wedge x \neq 0$$

Logo, $2 = 1$ (absurdo).

5.3 Prova por Indução

5.3.1 Indução Sobre os Naturais

$$n \in \mathbf{N} \qquad \frac{P(0) \qquad P(n) \rightarrow P(n+1)}{\forall n P(n)}$$

Base $P(0)$;

Passo $P(n) \rightarrow P(n+1)$;

- $P(n)$ é a *hipótese indutiva*;
- Então, tenta-se provar $P(n+1)$;

Conclusão $\forall n P(n)$.

ex: Provar que o quadrado de um número natural sempre é maior ou igual ao número.

Base $0 \leq 0^2$;

Passo Assumir $n \leq n^2$. Então:

$$n + 1 \leq n^2 + 1 \leq n^2 + 1 + 2n = (n + 1)^2$$

Conclusão $\forall n \in \mathbf{N}\ n \leq n^2$.

Exercício: Prove que para todo $n \in \mathbf{N}$, $2^n > n$.

Exercício: Prove que $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$.

5.3.2 Indução Sobre o Tamanho de uma Seqüência

1. Prove que a propriedade vale para todas as seqüências de comprimento 0 (base);
2. Prove que a propriedade vale para todas as outras seqüências: Assuma que a propriedade vale para todas as seqüências de tamanho menor ou igual a k (hipótese indutiva) e mostre que vale para seqüências de tamanho $k + 1$.

5.3.3 Indução Estrutural

Aplica-se sobre as estruturas de uma linguagem:

1. Prove que a propriedade vale para todas as estruturas atômicas (ex: $P(x{:=}a)$ e $P(\mathbf{skip})$);
2. Prove que a propriedade vale para qualquer programa: Assuma que vale para as estruturas que formam um comando, e mostre que vale para o comando composto (ex: $P(S_1) \wedge P(S_2) \rightarrow P(S_1; S_2)$).

6 Semântica de Expressões

6.1 Números e Numerais

$$\mathcal{N} : \mathbf{Num} \to \mathbf{Z}$$

Se $n \in \mathbf{Num}$ então escrevemos $\mathcal{N}[\![n]\!]$, e usamos $[\![$ e $]\!]$ para enfatizar que $\mathcal{N}$ é uma *função semântica*.

$$
\begin{aligned}
\mathcal{N}[\![0]\!] &= \mathbf{0} \\
\mathcal{N}[\![1]\!] &= \mathbf{1} \\
\mathcal{N}[\![n\,0]\!] &= \mathbf{2}{*}\mathcal{N}[\![n]\!] \\
\mathcal{N}[\![n\,1]\!] &= \mathbf{2}{*}\mathcal{N}[\![n]\!]{+}\mathbf{1}
\end{aligned}
$$

Observação: 0 é o numeral ($0 \in \mathbf{Num}$) e $\mathbf{0}$ é o número ($\mathbf{0} \in \mathbf{Z}$).

6.2 Descrevendo Estados

Estado representa uma associação entre variáveis e seus valores.

$$\mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}$$

ex: $[x \mapsto 5, y \mapsto 7, z \mapsto 0]$.

6.3 Expressões aritméticas

$$\mathcal{A} : \mathbf{Aexp} \to (\mathbf{State} \to \mathbf{Z})$$

$$\begin{aligned}
\mathcal{A}[\![n]\!]s &= \mathcal{N}[\![n]\!] \\
\mathcal{A}[\![x]\!]s &= s\,x \\
\mathcal{A}[\![a_1 + a_2]\!]s &= \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 * a_2]\!]s &= \mathcal{A}[\![a_1]\!]s * \mathcal{A}[\![a_2]\!]s \\
\mathcal{A}[\![a_1 - a_2]\!]s &= \mathcal{A}[\![a_1]\!]s - \mathcal{A}[\![a_2]\!]s
\end{aligned}$$

Observação: s é um estado.

ex: seja $s\,x = \mathbf{3}$:

$$\begin{aligned}
\mathcal{A}[\![x+1]\!]s &= \mathcal{A}[\![x]\!]s + \mathcal{A}[\![1]\!]s \\
&= (s\,x) + \mathcal{N}[\![1]\!] \\
&= \mathbf{3} + \mathbf{1} \\
&= \mathbf{4}
\end{aligned}$$

Observação: na primeira linha, + dentro do $[\![.]\!]$ pertence à linguagem, + à direita é a operação soma. 1 é o numeral ($1 \in \mathbf{Num}$) e $\mathbf{1}$ é o número ($\mathbf{1} \in \mathbf{Z}$).

6.4 Expressões booleanas

$$\mathcal{B} : \mathbf{Bexp} \to (\mathbf{State} \to \mathbf{T})$$

$$\mathbf{T} = \{\mathbf{tt}, \mathbf{ff}\}$$

$$
\begin{aligned}
\mathcal{B}[\![\text{true}]\!]s &= \mathbf{tt} \\
\mathcal{B}[\![\text{false}]\!]s &= \mathbf{ff} \\
\mathcal{B}[\![a_1 = a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{se } \mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{se } \mathcal{A}[\![a_1]\!]s \neq \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![a_1 \leq a_2]\!]s &= \begin{cases} \mathbf{tt} & \text{se } \mathcal{A}[\![a_1]\!]s \leq \mathcal{A}[\![a_2]\!]s \\ \mathbf{ff} & \text{se } \mathcal{A}[\![a_1]\!]s > \mathcal{A}[\![a_2]\!]s \end{cases} \\
\mathcal{B}[\![\neg b]\!]s &= \begin{cases} \mathbf{tt} & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{ff} \\ \mathbf{ff} & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{tt} \end{cases}
\end{aligned}
$$

$$\mathcal{B}[\![b_1 \wedge b_2]\!]s = \begin{cases} \mathbf{tt} & \text{se } \mathcal{B}[\![b_1]\!]s = \mathbf{tt} \text{ e } \mathcal{B}[\![b_2]\!]s = \mathbf{tt} \\ \mathbf{ff} & \text{se } \mathcal{B}[\![b_1]\!]s = \mathbf{ff} \text{ ou } \mathcal{B}[\![b_2]\!]s = \mathbf{ff} \end{cases}$$

7 Propriedades da Semântica

7.1 Variáveis Livres

As *variáveis livres* de uma expressão a é definida como o conjunto de variáveis que ocorrem em a.

$$
\begin{aligned}
\mathrm{FV}(n) &= \emptyset \\
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(a_1 + a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 * a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2) \\
\mathrm{FV}(a_1 - a_2) &= \mathrm{FV}(a_1) \cup \mathrm{FV}(a_2)
\end{aligned}
$$

Teorema 1 *Sejam s e s' dois estados satisfazendo que $s\,x = s'\,x$ para todo x em $\mathrm{FV}(a)$. Então, $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$.*

Prova: (por indução sobre as expressões aritméticas)

1. *Base:*

   caso n *Sabemos que $\mathcal{A}[\![a]\!]s = \mathcal{N}[\![n]\!]$ e $\mathcal{A}[\![a]\!]s' = \mathcal{N}[\![n]\!]$ e assim $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$;*

   caso x *nós temos $\mathcal{A}[\![x]\!]s = s\,x$ e $\mathcal{A}[\![x]\!]s' = s'\,x$ e, pela hipótese, $s\,x = s'\,x$, já que $x \in \mathrm{FV}(x)$. Logo, $\mathcal{A}[\![a]\!]s = \mathcal{A}[\![a]\!]s'$;*

2. Passo: (supomos que o enunciado vale para a_1 *e* a_2*)*

caso $a_1 + a_2$ *Sabemos que*

$\mathcal{A}[\![a_1 + a_2]\!]s = \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s$ *e*

$\mathcal{A}[\![a_1 + a_2]\!]s' = \mathcal{A}[\![a_1]\!]s' + \mathcal{A}[\![a_2]\!]s'$*. Além disto,*

$\mathrm{FV}(a1) \subseteq \mathrm{FV}(a_1 + a_2)$ *e* $\mathrm{FV}(a2) \subseteq \mathrm{FV}(a_1 + a_2)$*.*

Usando a hipótese indutiva, $\mathcal{A}[\![a_1]\!]s = \mathcal{A}[\![a_1]\!]s'$ *e*

$\mathcal{A}[\![a_2]\!]s = \mathcal{A}[\![a_2]\!]s'$*, e é fácil ver que o enunciado vale para* $a_1 + a_2$*;*

casos $a_1 * a_2$ **e** $a_1 - a_2$ *similares ao anterior.*

Exercício: Definir o conjunto $\mathrm{FV}(b)$ das variáveis livres em uma expressão booleana b.

Exercício: Provar que, dados s e s' satisfazendo $s\,x = s'\,x$ para todo $x \in \mathrm{FV}(b)$, $\mathcal{B}[\![b]\!]s = \mathcal{B}[\![b]\!]s'$.

7.2 Substituição

Substituição denota a operação em que se troca cada ocorrência de uma variável y de uma expressão a por uma outra expressão a_0.

Notação: $a[y \mapsto a_0]$.

$$
\begin{aligned}
n[y \mapsto a_0] &= n \\
x[y \mapsto a_0] &= \begin{cases} a_0 & \text{se } x = y \\ x & \text{se } x \neq y \end{cases} \\
(a_1 + a_2)[y \mapsto a_0] &= (a_1[y \mapsto a_0]) + (a_2[y \mapsto a_0]) \\
(a_1 * a_2)[y \mapsto a_0] &= (a_1[y \mapsto a_0]) * (a_2[y \mapsto a_0]) \\
(a_1 - a_2)[y \mapsto a_0] &= (a_1[y \mapsto a_0]) - (a_2[y \mapsto a_0])
\end{aligned}
$$

Exercício: Prove que
$\mathcal{A}[\![a[y \mapsto a_0]]\!]s = \mathcal{A}[\![a]\!](s[y \mapsto \mathcal{A}[\![a_0]\!]s])$, para todo s.

Exercício: Defina a substituição para expressões booleanas.

8 Semântica Operacional

- Expressões aritméticas e booleanas apenas produzem um valor, mas não mudam os estados;
- Comandos de programas While mudam o estado;
- Semântica operacional preocupa-se mais em *como* os programas são executados do que meramente com os resultados destas computações;

- Abordagens:

  Semântica natural descreve como os resultados *gerais* são obtidos;

  Semântica operacional estruturada descreve como *passos individuais* da computação ocorrem;

- Os significados dos comandos da linguagem serão descritos por meio de um *sistema de transições*;
- Existem dois tipos de configurações:

  $\langle S, s\rangle$ significando que o comando S será executado a partir do estado s;

  s representando um estado terminal (ou final).

8.1 Semântica Natural

$$[\text{ass}_{\text{ns}}] \qquad \langle x{:=}a, s\rangle \to s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{ns}}] \qquad \langle \mathbf{skip}, s\rangle \to s$$

$$[\text{comp}_{\text{ns}}] \qquad \frac{\langle S_1,s\rangle \to s', \langle S_2,s'\rangle \to s''}{\langle S_1;S_2,s\rangle \to s''}$$

$$[\text{if}^{\text{tt}}_{\text{ns}}] \qquad \frac{\langle S_1,s\rangle \to s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2,s\rangle \to s'} \text{ se } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{if}^{\text{ff}}_{\text{ns}}] \qquad \frac{\langle S_2,s\rangle \to s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2,s\rangle \to s'} \text{ se } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

$$[\text{while}^{\text{tt}}_{\text{ns}}] \qquad \frac{\langle S,s\rangle \to s', \langle \mathbf{while}\ b\ \mathbf{do}\ S,s'\rangle \to s''}{\langle \mathbf{while}\ b\ \mathbf{do}\ S,s\rangle \to s''} \text{ se } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

$$[\text{while}^{\text{ff}}_{\text{ns}}] \qquad \langle \mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \to s \text{ se } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

esquema de axioma ou axioma como em $[\mathrm{ass}_{\mathrm{ns}}]$;

regra como em $[\mathrm{comp}_{\mathrm{ns}}]$;

árvore de derivação é a aplicação dos axiomas e regras para deduzir uma transição $\langle S, s\rangle \rightarrow s'$.

Observação: x é uma meta-variável.

ex: (árvore de derivação)

$$(z{:=}x; x{:=}y); y{:=}z \qquad s_0 = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$$

$$\frac{\dfrac{\langle z{:=}x, s_0\rangle \to s_1 \qquad \langle x{:=}y, s_1\rangle \to s_2}{\langle z{:=}x; x{:=}y, s_0\rangle \to s_2} \qquad \langle y{:=}z, s_2\rangle \to s_3}{\langle (z{:=}x; x{:=}y); y{:=}z, s_0\rangle \to s_3}$$

Observação: $s_1 = s_0[z \mapsto 5]$, $s_2 = s_1[x \mapsto 7]$ e
$s_3 = s_2[y \mapsto 5]$.

Como construir uma árvore de derivação?

Resposta: da raiz para as folhas.

ex: $y{:=}1; \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x{-}1)$ e
$s = [x \mapsto 3, y \mapsto 0]$

Raiz:

$\langle y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s\rangle \rightarrow s_{61}$

$$s_{61} = s[y \mapsto 6][x \mapsto 1]$$

árvore T:

$$\frac{\langle y{:=}1, s\rangle \to s_{13} \qquad T_1}{\langle y{:=}1; \mathbf{while}\ \neg(x=1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x{-}1), s\rangle \to s_{61}}$$

$$s_{13} = s[y \mapsto 1]$$

árvore T_1:

$$\frac{T_2 \qquad T_3}{\langle \mathbf{while}\ \neg(x=1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x-1), s_{13}\rangle \to s_{61}}$$

Pois $\mathcal{B}[\![\neg(x=1)]\!]s_{13} = \mathbf{tt}$

árvore T_2:

$$\frac{\langle y{:=}y{*}x, s_{13}\rangle \to s_{33} \qquad \langle x{:=}x{-}1, s_{33}\rangle \to s_{32}}{\langle y{:=}y{*}x; x{:=}x-1, s_{13}\rangle \to s_{32}}$$

$$s_{33} = s[y \mapsto 3] \qquad s_{32} = s[y \mapsto 3][x \mapsto 2]$$

árvore T_3:

$$\dfrac{\dfrac{\langle y{:=}y{*}x, s_{32}\rangle \to s_{62} \qquad \langle x{:=}x{-}1, s_{62}\rangle \to s_{61}}{\langle y{:=}y{*}x; x{:=}x{-}1, s_{32}\rangle \to s_{61}} \qquad T_4}{\langle \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s_{32}\rangle \to s_{61}}$$

$$s_{62} = s[y \mapsto 6][x \mapsto 2]$$

árvore T_4 (folha):

$$\langle \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s_{61} \rangle \rightarrow s_{61}$$

Pois $\mathcal{B}[\![\neg(x = 1)]\!]s_{61} = \textbf{ff}$

Exercício: Construa a árvore de derivação para o seguinte programa:

$$z{:=}0;\ \textbf{while}\ y \leq x\ \textbf{do}\ (z{:=}z{+}1; x{:=}x{-}y)$$

com $s = [x \mapsto \mathbf{17}, y \mapsto \mathbf{5}]$.

A execução de um comando S no estado s:

- *termina* se e somente se existe um estado s' tal que $\langle S, s\rangle \rightarrow s'$;
- entra em *loop* se e somente se não existe um estado s' tal que $\langle S, s\rangle \rightarrow s'$.

8.1.1 Propriedades da Semântica

S_1 e S_2 são *semanticamente equivalentes* se

$$\langle S_1, s\rangle \to s' \text{ se e somente se } \langle S_2, s\rangle \to s'$$

para todos os estados s e s'.

Provar que **while** b **do** S é semanticamente equivalente a **if** b **then** $(S;$ **while** b **do** $S)$ **else skip**.

Prova: ($\Rightarrow$) Se $\langle$**while** b **do** $S, s\rangle \to s''$ então
$\langle$**if** b **then** $(S;$ **while** b **do** $S)$ **else skip**$, s\rangle \to s''$

$$\langle \textbf{while } b \textbf{ do } S, s\rangle \to s''$$

Possíveis árvores:

$$\frac{T_1 \qquad\qquad T_2}{\langle \textbf{while } b \textbf{ do } S, s\rangle \to s''}$$

Onde T_1 é a árvore com raiz $\langle S, s\rangle \to s'$ e T_2 é a árvore com raiz $\langle$**while** b **do** $S, s'\rangle \to s''$ e $\mathcal{B}[\![b]\!]s = \mathbf{tt}$.

$$[\text{comp}_{\text{ns}}] \qquad \frac{\langle S_1, s\rangle \to s' \qquad\qquad \langle S_2, s'\rangle \to s''}{\langle S_1; S_2, s\rangle \to s''}$$

Usando T_1 e T_2 como premissas da regra $[\text{comp}_{\text{ns}}]$:

$$\frac{T_1 \qquad\qquad T_2}{\langle S; \textbf{while } b \textbf{ do } S, s\rangle \to s''}$$

$$[\mathrm{if}^{\mathrm{tt}}_{\mathrm{ns}}] \qquad \frac{\langle S_1, s\rangle \to s'}{\langle \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2, s\rangle \to s'} \quad \text{Se } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

Obtemos:

$$\frac{\dfrac{T_1 \qquad T_2}{\langle S;\mathbf{while}\ b\ \mathbf{do}\ S, s\rangle \to s''}}{\langle \mathbf{if}\ b\ \mathbf{then}\ (S;\mathbf{while}\ b\ \mathbf{do}\ S)\ \mathbf{else}\ \mathbf{skip}, s\rangle \to s''}$$

Falta provar o caso $\mathcal{B}[\![b]\!]s = \mathbf{ff}$. Neste caso, T é simplesmente:

$$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s'' \rangle \to s''$$

Usando $[\text{skip}_{\text{ns}}]$:

$$\langle \mathbf{skip}, s'' \rangle \to s''$$

Usando $[\text{if}^{\text{ff}}_{\text{ns}}]$:

$$[\text{if}^{\text{ff}}_{\text{ns}}] \qquad \frac{\langle S_2, s\rangle \to s'}{\langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \to s'} \quad \text{Se } \mathcal{B}[\![b]\!]s = \textbf{ff}$$

$$\frac{\langle \textbf{skip}, s''\rangle \to s''}{\langle \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}, s''\rangle \to s''}$$

($\Leftarrow$) Se $\langle \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}, s\rangle \to s''$ então $\langle \textbf{while } b \textbf{ do } S, s\rangle \to s''$

Usando $[\text{if}^{\text{tt}}_{\text{ns}}]$:

$$\frac{\langle S; \textbf{while } b \textbf{ do } S, s\rangle \to s''}{\langle \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}, s\rangle \to s''}$$

e $\mathcal{B}[\![b]\!]s = \textbf{tt}$

Usando $[\text{comp}_{\text{ns}}]$:

$$\frac{\dfrac{\langle S,s\rangle \to s' \qquad \langle \textbf{while } b \textbf{ do } S,s'\rangle \to s''}{\langle S;\textbf{while } b \textbf{ do } S,s\rangle \to s''}}{\langle \textbf{if } b \textbf{ then } (S;\textbf{while } b \textbf{ do } S) \textbf{ else skip}, s\rangle \to s''}$$

Usando $[\text{while}^{\text{tt}}_{\text{ns}}]$:

$$\frac{\langle S, s\rangle \to s' \qquad \langle \textbf{while } b \textbf{ do } S, s'\rangle \to s''}{\langle \textbf{while } b \textbf{ do } S, s\rangle \to s''}$$

No caso $\mathcal{B}[\![b]\!]s = \mathbf{ff}$:

$$\frac{\langle \mathbf{skip}, s\rangle \to s''}{\langle \mathbf{if}\ b\ \mathbf{then}\ (S; \mathbf{while}\ b\ \mathbf{do}\ S)\ \mathbf{else}\ \mathbf{skip}, s\rangle \to s''}$$

e $s = s''$.

Usando $[\text{while}^{\text{ff}}_{\text{ns}}]$:

$$\langle \mathbf{while}\ b\ \mathbf{do}\ S, s''\rangle \to s''$$

Isto completa a prova.

Exercício: Prove que $S_1; (S_2; S_3)$ e $(S_1; S_2); S_3$ são semanticamente equivalentes.

Exercício: Mostre que $S_1; S_2$, no geral, não é semanticamente equivalente a $S_2; S_1$.

Uma semântica é *determinística* se, para todos S, s, s' e s'':

$$\langle S, s\rangle \to s' \text{ e } \langle S, s\rangle \to s'' \text{ implica } s' = s'' .$$

Isto significa que para todo comando S e estado inicial s nós podemos determinar um *único* estado final s' se a execução de S termina.

Teorema 2 *A semântica natural é determinística.*

8.1.2 Função Semântica

$$\mathcal{S}_{\text{ns}} : \mathbf{Smt} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

Observação: usa-se $\hookrightarrow$ para funções parciais. Ou seja:

$$\mathcal{S}_{\text{ns}}[\![S]\!] \in \mathbf{State} \hookrightarrow \mathbf{State}$$

dado por:

$$\mathcal{S}_{\text{ns}}[\![S]\!]s = \begin{cases} s' & \text{Se } \langle S, s\rangle \to s' \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

Observação: $\mathcal{S}_{\text{ns}}[\![\mathbf{while\ true\ do\ skip}]\!]s = \underline{\text{undef}}$ para todo s (função parcial).

8.2 Semântica Operacional Estruturada

- Ênfase nos passos individuais da computação;
- Relação de transição: $\langle S, s\rangle \Rightarrow \gamma$ (a transição expressa o *primeiro* passo da computação), com dois possíveis resultados:
  - $\gamma = \langle S', s'\rangle$ significando que a execução de S *não* está completa e $\langle S', s'\rangle$ é uma configuração intermediária;
  - $\gamma = s'$ significando que a execução de S terminou e o estado final é s';
- Se diz que $\langle S, s\rangle$ *emperrou* se não existe γ tal que $\langle S, s\rangle \Rightarrow \gamma$.

$$[\text{ass}_{\text{SOS}}] \qquad \langle x{:=}a, s\rangle \Rightarrow s[x \mapsto \mathcal{A}[\![a]\!]s]$$

$$[\text{skip}_{\text{SOS}}] \qquad \langle \textbf{skip}, s\rangle \Rightarrow s$$

$$[\text{comp}^1_{\text{SOS}}] \qquad \frac{\langle S_1,s\rangle \Rightarrow \langle S_1',s'\rangle}{\langle S_1;S_2,s\rangle \Rightarrow \langle S_1';S_2,s'\rangle}$$

$$[\text{comp}^2_{\text{SOS}}] \qquad \frac{\langle S_1,s\rangle \Rightarrow s'}{\langle S_1;S_2,s\rangle \Rightarrow \langle S_2,s'\rangle}$$

$$[\text{if}^{\text{tt}}_{\text{SOS}}] \qquad \langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_1, s\rangle$$

$$\text{Se } \mathcal{B}[\![b]\!]s = \textbf{tt}$$

$$[\text{if}^{\text{ff}}_{\text{SOS}}] \qquad \langle \textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$$

$$\text{Se } \mathcal{B}[\![b]\!]s = \textbf{ff}$$

$$[\text{while}_{\text{SOS}}] \qquad \langle \textbf{while } b \textbf{ do } S, s\rangle \Rightarrow$$

$$\langle \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}, s\rangle$$

Uma *seqüência de derivação* de um comando S iniciando no estado s é:

- uma seqüência finita

$$\gamma_0, \gamma_1, \gamma_2, \ldots, \gamma_k$$

de configurações satisfazendo $\gamma_0 = \langle S, s \rangle$, $\gamma_i \Rightarrow \gamma_{i+1}$ para $0 \leq i < k$, e $k \geq 0$, e ou γ_k é uma configuração terminal ou é uma configuração emperrada; ou

- uma seqüência infinita

$$\gamma_0, \gamma_1, \gamma_2, \ldots$$

de configurações satisfazendo $\gamma_0 = \langle S, s \rangle$ e $\gamma_i \Rightarrow \gamma_{i+1}$ para $i \geq 0$.

Notação: Usamos $\gamma_0 \Rightarrow^i \gamma_i$ para indicar i passos na seqüência. Usamos $\gamma_0 \Rightarrow^* \gamma_i$ para indicar um número finito de passos.

ex: $(z{:=}x; x{:=}y); y{:=}z$ e $s_0 = [x \mapsto 5, y \mapsto 7, z \mapsto 0]$.

Seqüência de derivação:

$$\begin{aligned}
\langle (z{:=}x; x{:=}y); y{:=}z, s_0 \rangle &\Rightarrow \langle x{:=}y; y{:=}z, s_0[z \mapsto 5] \rangle \\
&\Rightarrow \langle y{:=}z, (s_0[z \mapsto 5])[x \mapsto 7] \rangle \\
&\Rightarrow ((s_0[z \mapsto 5])[x \mapsto 7])[y \mapsto 5]
\end{aligned}$$

Cada passo da derivação corresponde a uma *árvore de derivação*. Assim, para o primeiro passo teríamos:

$$\frac{\dfrac{\langle z{:=}x, s_0\rangle \Rightarrow s_0[z \mapsto 5]}{\langle z{:=}x; x{:=}y, s_0\rangle \Rightarrow \langle x{:=}y, s_0[z \mapsto 5]\rangle}}{\langle (z{:=}x; x{:=}y); y{:=}z, s_0\rangle \Rightarrow \langle x{:=}y; y{:=}z, s_0[z \mapsto 5]\rangle}$$

ex: Seja $s\ x = \mathbf{3}$ e
$y{:=}1; \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x{-}1)$. O primeiro passo é:

$$\langle y{:=}1; \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x{-}1), s\rangle \Rightarrow$$

$$\langle \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y{*}x; x{:=}x{-}1), s[y \mapsto 1]\rangle$$

Árvore:

$$\frac{\langle y{:=}1, s\rangle \Rightarrow s[y \mapsto 1]}{\begin{array}{c}\langle y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s\rangle \Rightarrow \\ \langle \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s[y \mapsto 1]\rangle\end{array}}$$

Observação: usa-se $[\mathrm{ass}_{\mathrm{SOS}}]$ e $[\mathrm{comp}^2_{\mathrm{SOS}}]$.

Reescrevendo o laço como um condicional:

$$\langle \textbf{if } \neg(x = 1) \textbf{ then } ((y{:=}y{*}x; x{:=}x{-}1);\ \textbf{while}$$

$$\neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1)) \textbf{ else skip}, s[y \mapsto 1]\rangle$$

Segundo $[\text{if}^{\text{tt}}_{\text{SOS}}]$, o passo seguinte resultará em:

$$\langle (y{:=}y{*}x; x{:=}x{-}1);$$

$$\textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s[y \mapsto 1]\rangle$$

Usando-se $[\text{ass}_{\text{SOS}}]$, $[\text{comp}^2_{\text{SOS}}]$ e $[\text{comp}^1_{\text{SOS}}]$, obtem-se a transição:

$$\langle (y{:=}y{*}x;\, x{:=}x{-}1);$$
$$\textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x;\, x{:=}x{-}1), s[y \mapsto 1]\rangle \Rightarrow$$
$$\langle x{:=}x{-}1;\ \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x;\, x{:=}x{-}1), s[y \mapsto 3]\rangle$$

Árvore:

$$\dfrac{\dfrac{\langle y{:=}y{*}x, s[y{\mapsto}1]\rangle \Rightarrow s[y{\mapsto}3]}{\langle y{:=}y{*}x; x{:=}x{-}1, s[y{\mapsto}1]\rangle \Rightarrow \langle x{:=}x{-}1, s[y{\mapsto}3]\rangle}}{\begin{array}{c}\langle (y{:=}y{*}x;\, x{:=}x{-}1);\\ \textbf{while } \neg(x=1) \textbf{ do } (y{:=}y{*}x;\, x{:=}x{-}1), s[y\mapsto 1]\rangle \Rightarrow\\ \langle x{:=}x{-}1;\\ \textbf{while } \neg(x=1) \textbf{ do } (y{:=}y{*}x;\, x{:=}x{-}1), s[y\mapsto 3]\rangle\end{array}}$$

Usando $[\text{ass}_{\text{SOS}}]$ e $[\text{comp}^2_{\text{SOS}}]$ obtemos a próxima configuração:

$$\langle \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1), s[y \mapsto 3][x \mapsto 2] \rangle$$

Continuando, chegaremos ao estado final $s[y \mapsto 6][x \mapsto 1]$.

Exercício: Seja $s = [x \mapsto 17, y \mapsto 5]$. Construa a seqüência de derivação do comando:

$$z{:=}0; \textbf{while } y \leq x \textbf{ do } (z{:=}z{+}1; x{:=}x{-}y)$$

Importante observar que a linguagem exemplo While não possui configurações emperradas.

A execução de um comando S no estado s:

- *termina* se e somente se existe uma seqüência de derivação finita começando com $\langle S, s\rangle$;
- entra em *loop* se e somente se existe uma seqüência de derivação infinita começando com $\langle S, s\rangle$.

Dizemos que s execução de S em s *termina com sucesso* se $\langle S, s\rangle \Rightarrow^* s'$, para algum estado s'. Na linguagem exemplo While a execução termina com sucesso se e somente se termina pois não existem configurações emperradas.

8.2.1 Propriedades da Semântica

Lema 1 *Se* $\langle S_1; S_2, s\rangle \Rightarrow^k s''$, *então existe um estado* s' *e números naturais* k_1 *e* k_2 *tal que* $\langle S_1, s\rangle \Rightarrow^{k_1} s'$ *e* $\langle S_2, s'\rangle \Rightarrow^{k_2} s''$, *onde* $k = k_1 + k_2$.

Prova: (por indução sobre o tamanho da seqüência de derivação)

Base: $k = 0$ *(por vacuidade)*

Passo: Assumimos como hipótese que o enunciado vale para todo $k \leq k_0$. *Assumimos* $\langle S_1; S_2, s\rangle \Rightarrow^{k_0+1} s''$ *ou* $\langle S_1; S_2, s\rangle \Rightarrow \gamma \Rightarrow^{k_0} s''$ *para alguma configuração* γ.

O primeiro passo pode ser resultado de $[\text{comp}^1_{\text{SOS}}]$ *ou* $[\text{comp}^2_{\text{SOS}}]$*:*

1. $[\text{comp}^1_{\text{SOS}}]$ *e* $\gamma = \langle S_1'; S_2, s' \rangle$*, ou seja*

$$\langle S_1; S_2, s \rangle \Rightarrow \langle S_1'; S_2, s' \rangle$$

Logo, $\langle S_1, s \rangle \Rightarrow \langle S_1', s' \rangle$ *e* $\langle S_1'; S_2, s' \rangle \Rightarrow^{k_0} s''$*.*
Esta segunda seqüência satisfaz a hipótese indutiva.
Assim, existe um estado s_0 *e números* k_1 *e* k_2 *tal que*

$$\langle S_1', s' \rangle \Rightarrow^{k_1} s_0 \text{ e } \langle S_2, s_0 \rangle \Rightarrow^{k_2} s''$$

onde $k_1 + k_2 = k_0$*. Logo,* $\langle S_1, s \rangle \Rightarrow^{k_1+1} s_0$ *e* $\langle S_2, s_0 \rangle \Rightarrow^{k_2} s''$*, e como* $(k_1 + 1) + k_2 = k_0 + 1$*, está provado o resultado;*

2. $[\mathrm{comp}^2_{\mathrm{SOS}}]$ *e* $\gamma = s'$. *Assim,* $\langle S_1, s\rangle \Rightarrow s'$ *e* $\langle S_2, s'\rangle \Rightarrow^{k_0} s''$. *O resultado fica provado fazendo-se* $k_1 = 1$ *e* $k_2 = k_0$.

Exercício: Prove que se $\langle S_1, s\rangle \Rightarrow^k s'$ então $\langle S_1; S_2, s\rangle \Rightarrow^k \langle S_2, s'\rangle$, isto é, a execução de S_1 não é influenciada pelo comando que o segue.

Teorema 3 *A semântica operacional estruturada é determinística.*

8.2.2 Função Semântica

$$\mathcal{S}_{\mathrm{SOS}} : \mathbf{Smt} \to (\mathbf{State} \hookrightarrow \mathbf{State}),$$

dado por:

$$\mathcal{S}_{\mathrm{SOS}}[\![S]\!]s = \begin{cases} s' & \text{Se } \langle S, s\rangle \Rightarrow^* s' \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

8.3 Um Resultado de Equivalência

Teorema 4 *Para todo comando S da linguagem exemplo While nós temos* $\mathcal{S}_{\mathrm{ns}}[\![S]\!] = \mathcal{S}_{\mathrm{SOS}}[\![S]\!]$.

Este teorema expressa duas propriedades:

1. Se a execução de S, começando em algum estado, termina em uma semântica, então ela também termina na outra, e os estados resultantes são iguais;
2. Se a execução de S, para algum estado, entra em loop em uma semântica então também entra em loop na outra.

9 Semântica Denotacional

Semântica Operacional *Como* ocorre a execução do comando;

Semântica Denotacional O *efeito* da execução do comando.

- Na abordagem denotacional cada construção sintática é mapeada, por uma função semântica, para um objeto matemático (de um modo geral uma função);
- Exemplos de funções semânticas denotacionais: $\mathcal{A}$ (mapeia expressões aritméticas para funções em $\mathbf{State} \rightarrow \mathbf{Z}$) e $\mathcal{B}$ (mapeia expressões booleanas para funções em $\mathbf{State} \rightarrow \mathbf{T}$);

- Funções $\mathcal{S}_{\text{ns}}$ e $\mathcal{S}_{\text{SOS}}$ são exemplos de funções semânticas que mapeiam comandos em funções em $\textbf{State} \hookrightarrow \textbf{State}$, no entanto, elas *não* são exemplos de funções denotacionais porque elas não são definidas de forma *composicional*;
- Na semântica denotacional, estruturas de laço de repetição são interpretadas usando-se *pontos fixos*:
  - pontos fixos são raízes de uma *equação funcional*;
  - um *funcional* é uma função que mapeia uma função em outra função.

9.1 Composicionalidade

Significa expressar uma propriedade de uma estrutura em função da composição das propriedades de suas sub-estruturas.

ex: $\mathcal{A}[\![a_1 + a_2]\!]s = \mathcal{A}[\![a_1]\!]s + \mathcal{A}[\![a_2]\!]s$

9.2 Ponto Fixo

Um *funcional* mapeia uma função em outra função.

Seja F um funcional, então:

$$\begin{aligned} F\ g_1 &= g_2 \\ F\ g_0 &= g_0 \text{ (ponto fixo)} \\ F^k\ g_0 &= g_0 \end{aligned}$$

ex: $F\ g =$ **se** $x = 0$ **então** 1 **caso contrário** $x{*}g(x-1)$,
e o ponto fixo de F é a função fatorial. Ou seja,
$F\ g_0 = g_0$ e g_0 é a função fatorial.

Observação: usamos uma linguagem de programação funcional hipotética neste exemplo.

Seja $g_1(x) = x^2$. Então:

$F\ g_1 = \textbf{se}\ x = 0\ \textbf{então}\ 1\ \textbf{caso contrário}\ x * (x-1)^2 = g_2$

e $g_1 \neq g_2$.

Seja $g_0(x) = x!$. Então:

$F\ g_0 = \textbf{se}\ x = 0\ \textbf{então}\ 1\ \textbf{caso contrário}\ x * ((x-1)!) =$

$\textbf{se}\ x = 0\ \textbf{então}\ 1\ \textbf{caso contrário}\ x! = x! = g_0$

9.3 Definição

$$
\begin{aligned}
\mathcal{S}_{\mathrm{ds}}[\![x{:=}a]\!]s &= s[x \mapsto \mathcal{A}[\![a]\!]s] \\
\mathcal{S}_{\mathrm{ds}}[\![\mathbf{skip}]\!] &= \mathrm{id} \\
\mathcal{S}_{\mathrm{ds}}[\![S_1; S_2]\!] &= \mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!] \\
\mathcal{S}_{\mathrm{ds}}[\![\mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2]\!] &= \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]) \\
\mathcal{S}_{\mathrm{ds}}[\![\mathbf{while}\ b\ \mathbf{do}\ S]\!] &= \mathrm{FIX}\ F
\end{aligned}
$$

Onde $F\ g = \mathrm{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})$.

Explicação:

$$\mathcal{S}_{\mathrm{ds}}[\![x{:=}a]\!]s = s[x \mapsto \mathcal{A}[\![a]\!]s]$$

Observação: corresponde a $[\mathrm{ass}_{\mathrm{ns}}]$ e $[\mathrm{ass}_{\mathrm{SOS}}]$.

$$\mathcal{S}_{\mathrm{ds}}[\![\mathbf{skip}]\!] = \mathrm{id}$$

Observação: id é a função *identidade*.

$$\mathcal{S}_{\mathrm{ds}}[\![S_1; S_2]\!] = \mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]$$

Observação: isto significa que o efeito da execução de $S_1; S_2$ é a composição funcional do efeito da execução de S_1 com o efeito da execução de S_2.

$$\mathcal{S}_{\mathrm{ds}}[\![S_1; S_2]\!]s = (\mathcal{S}_{\mathrm{ds}}[\![S_2]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S_1]\!])s =$$

$$= \begin{cases} s'' & \text{se existe } s' \text{ tal que} \\ & \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \text{ e } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s' = s'' \\ \underline{\text{undef}} & \text{se } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = \underline{\text{undef}} \text{ ou} \\ & \text{se existe } s' \text{ tal que} \\ & \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \text{ e} \\ & \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s' = \underline{\text{undef}} \end{cases}$$

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2]\!] = \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!])$$

Observação: a função auxiliar cond tem funcionalidade

$$\mathrm{cond} : (\mathbf{State} \to \mathbf{T}) \times (\mathbf{State} \hookrightarrow \mathbf{State}) \times (\mathbf{State} \hookrightarrow \mathbf{State}) \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

e é definida por:

$$\mathrm{cond}(p, g_1, g_2)s = \begin{cases} g_1\ s & \text{se } p\ s = \mathbf{tt} \\ g_2\ s & \text{se } p\ s = \mathbf{ff} \end{cases}$$

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{if } b \textbf{ then } S_1 \textbf{ else } S_2]\!]s =$$

$$= \quad \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![S_1]\!], \mathcal{S}_{\mathrm{ds}}[\![S_2]\!])s$$

$$= \begin{cases} s' & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{tt} \text{ e } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = s' \text{ ou} \\ & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{ff} \text{ e } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s = s' \\ \underline{\text{undef}} & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{tt} \text{ e } \mathcal{S}_{\mathrm{ds}}[\![S_1]\!]s = \underline{\text{undef}} \text{ ou} \\ & \text{se } \mathcal{B}[\![b]\!]s = \mathbf{ff} \text{ e } \mathcal{S}_{\mathrm{ds}}[\![S_2]\!]s = \underline{\text{undef}} \end{cases}$$

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{while } b \textbf{ do } S]\!] = \mathrm{FIX}\ F$$

Observe que:

$$\textbf{while } b \textbf{ do } S \ = \ \textbf{if } b \textbf{ then } (S; \textbf{while } b \textbf{ do } S) \textbf{ else skip}$$

e aplicando a definição de $\mathcal{S}_{\mathrm{ds}}$ ao lado direito resulta em:

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{while } b \textbf{ do } S]\!] =$$

$$= \mathrm{cond}(\mathcal{B}[\![b]\!], \mathcal{S}_{\mathrm{ds}}[\![\textbf{while } b \textbf{ do } S]\!] \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})$$

- Não podemos usar esta última expressão, pois ela não está definida de forma *composicional*;
- No entanto, isto expressa que $\mathcal{S}_{\text{ds}}[\![\textbf{while } b \textbf{ do } S]\!]$ é o *ponto fixo* do funcional definido por:

$$F\ g = \text{cond}(\mathcal{B}[\![b]\!], g \circ \mathcal{S}_{\text{ds}}[\![S]\!], \text{id})$$

ou seja,

$$\mathcal{S}_{\text{ds}}[\![\textbf{while } b \textbf{ do } S]\!] = F(\mathcal{S}_{\text{ds}}[\![\textbf{while } b \textbf{ do } S]\!])$$

- Agora, esta nova definição respeita a composicionalidade.

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{while } b \textbf{ do } S]\!] = \mathrm{FIX}\ F$$

e a funcionalidade da função auxiliar FIX é

$$\mathrm{FIX} : ((\textbf{State} \hookrightarrow \textbf{State}) \to (\textbf{State} \hookrightarrow \textbf{State})) \to$$

$$(\textbf{State} \hookrightarrow \textbf{State})$$

ex: Seja o seguinte comando:

$$\textbf{while } \neg(x = 0) \textbf{ do skip}$$

O funcional correspondente é:

$$(F'\ g)s = \begin{cases} g\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

Determinação do funcional:

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{while } \neg(x = 0) \textbf{ do skip}]\!]s = (\mathrm{FIX}\ F')s$$

$$\begin{aligned}
(F'\ g)s &= \mathrm{cond}(\mathcal{B}[\![\neg(x = 0)]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![\textbf{skip}]\!], \mathrm{id})s \\
&= \begin{cases} g \circ \mathcal{S}_{\mathrm{ds}}[\![\textbf{skip}]\!]s & \text{se } \mathcal{B}[\![\neg(x = 0)]\!]s = \textbf{tt} \\ \mathrm{id}\ s & \text{se } \mathcal{B}[\![\neg(x = 0)]\!]s = \textbf{ff} \end{cases} \\
&= \begin{cases} g\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}
\end{aligned}$$

Observação: $g \circ \mathcal{S}_{\mathrm{ds}}[\![\textbf{skip}]\!]s = g \circ \mathrm{id}\ s = g\ s$, pois $g \circ \mathrm{id} = g$.

A função g_1 definida como:

$$g_1\ s = \begin{cases} \underline{\text{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

é ponto fixo de F' porque

$$\begin{aligned} (F'\ g_1)s &= \begin{cases} g_1\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases} \\ &= \begin{cases} \underline{\text{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases} \\ &= g_1\ s \end{aligned}$$

Já, por sua vez, g_2, definida como:

$$g_2\ s = \underline{\text{undef}} \text{ para todo } s$$

não é ponto fixo de F' porque se $s'\ x = 0$ então $(F'\ g_2)s' = s'$, enquanto que $g_2\ s' = \underline{\text{undef}}$.

- Existem funcionais com mais de um ponto fixo. Um exemplo é F', já que qualquer função g' de $\mathbf{State} \hookrightarrow \mathbf{State}$ satisfazendo $g'\ s = s$ se $s\ x = 0$ é ponto fixo de F';
- Existem também funcionais que não tem ponto fixo. Considere F_1 definido como:

$$F_1\ g = \begin{cases} g_1 & \text{se } g = g_2 \\ g_2 & \text{caso contrário} \end{cases}$$

Se $g_1 \neq g_2$ então não existe função g_0 tal que $F_1\ g_0 = g_0$.

Exercício: Determine o funcional F associado ao comando:

$$\textbf{while } \neg(x = 0) \textbf{ do } x{:=}x - 1$$

Solução:

$$\mathcal{S}_{\mathrm{ds}}[\![\textbf{while } \neg(x = 0) \textbf{ do } x{:=}x - 1]\!]s = (\mathrm{FIX}\ F)s$$

$$\begin{aligned}
(F\ g)s &= \mathrm{cond}(\mathcal{B}[\![\neg(x = 0)]\!], g \circ \mathcal{S}_{\mathrm{ds}}[\![x{:=}x - 1]\!], \mathrm{id})s \\
&= \begin{cases} g \circ \mathcal{S}_{\mathrm{ds}}[\![x{:=}x - 1]\!]s & \text{se } \mathcal{B}[\![\neg(x = 0)]\!]s = \mathbf{tt} \\ \mathrm{id}\ s & \text{se } \mathcal{B}[\![\neg(x = 0)]\!]s = \mathbf{ff} \end{cases} \\
&= \begin{cases} g\ s[x \mapsto s\ x - 1] & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}
\end{aligned}$$

Exercício: Considere as seguintes funções parciais:

1. $g_1\ s = \underline{\text{undef}}$ para todo s
2. $$g_2\ s = \begin{cases} s[x \mapsto 0] & \text{se } s\ x \geq 0 \\ \underline{\text{undef}} & \text{se } s\ x < 0 \end{cases}$$
3. $$g_3\ s = \begin{cases} s[x \mapsto 0] & \text{se } s\ x \geq 0 \\ s & \text{se } s\ x < 0 \end{cases}$$
4. $g_4\ s = s[x \mapsto 0]$ para todo s
5. $g_5\ s = s$ para todo s

Determine quais são pontos fixos de F.

Exercício: Considere o seguinte fragmento de programa:

$$\textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x - 1)$$

Determine o funcional F associado ao programa.
Determine também ao menos dois pontos fixos de F.

9.4 Requisitos sobre o Ponto Fixo

Perante o problema de haver mais de um ponto fixo, ou de não existir nenhum, se faz necessário:

- Definir os requisitos de um ponto fixo e mostrar que ao menos um ponto fixo preenche estes requisitos;
- Mostrar que todos os funcionais obtidos da linguagem While tem um ponto fixo que satisfaz estes requisitos.

Seja o comando **while** b **do** S. Quanto a terminação, existem três possibilidades:

1. Ele *termina*;
2. Ele *entra em loop localmente*, ou seja, S entra em loop;
3. Ele *entra em loop globalmente*, ou seja, a estrutura while entra em loop.

1. Caso em que o laço termina: Assim, existe uma seqüência de estados $s_0, s_1, \ldots, s_n$ tal que

$$\mathcal{B}[\![b]\!]s_i = \begin{cases} \mathbf{tt} & \text{se } i < n \\ \mathbf{ff} & \text{se } i = n \end{cases}$$

e

$$\mathcal{S}_{\text{ds}}[\![S]\!]s_i = s_{i+1} \text{ para } i < n$$

Seja g_0 um ponto fixo de F. Logo, no caso $i < n$ temos:

$$\begin{aligned} g_0\ s_i &= (F\ g_0)s_i \\ &= \mathrm{cond}(\mathcal{B}[\![b]\!], g_0 \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})s_i \\ &= g_0(\mathcal{S}_{\mathrm{ds}}[\![S]\!]s_i) \\ &= g_0\ s_{i+1} \end{aligned}$$

No caso $i = n$, temos:

$$\begin{aligned} g_0\ s_n &= (F\ g_0)s_n \\ &= \mathrm{cond}(\mathcal{B}[\![b]\!], g_0 \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})s_n \\ &= \mathrm{id}\ s_n \\ &= s_n \end{aligned}$$

Conclui-se que todos os pontos fixos satisfazem $g_0\ s_0 = s_n$, não sendo possível tirar daqui nenhum requisito.

2. Caso em que o laço entra em loop localmente, ou seja, S entra em loop: Existe uma seqüência de estados $s_0, s_1, \ldots, s_n$ tal que

$$\mathcal{B}[\![b]\!]s_i = \mathbf{tt} \text{ para } i \leq n$$

e

$$\mathcal{S}_{\mathrm{ds}}[\![S]\!]s_i = \begin{cases} s_{i+1} & \text{para } i < n \\ \underline{\text{undef}} & \text{para } i = n \end{cases}$$

Seja g_0 um ponto fixo do funcional F. No caso $i < n$ obtemos:

$$g_0\ s_i = g_0\ s_{i+1}$$

e no caso $i = n$ obtemos:

$$\begin{aligned} g_0\ s_n &= (F\ g_0)s_n \\ &= \mathrm{cond}(\mathcal{B}[\![b]\!], g_0 \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!], \mathrm{id})s_n \\ &= (g_0 \circ \mathcal{S}_{\mathrm{ds}}[\![S]\!])s_n \\ &= \underline{\mathrm{undef}} \end{aligned}$$

E novamente não se obtem nenhum requisito.

3. Caso em que o laço entra em loop globalmente, ou seja, a estrutura while entra em loop: Existe um seqüência infinita de estados $s_0, s_1, s_2, \ldots$ tal que

$$\mathcal{B}[\![b]\!]s_i = \mathbf{tt} \text{ para todo } i$$

e

$$\mathcal{S}_{\mathrm{ds}}[\![S]\!]s_i = s_{i+1} \text{ para todo } i$$

Seja g_0 um ponto fixo de F, então

$$g_0\ s_i = g_0\ s_{i+1}$$

para todo $i \geq 0$. E temos $g_0\ s_0 = g_0\ s_i$ para todo i. Esta é a situação em que podemos ter vários pontos fixos diferentes.

ex:

$$S = \textbf{while } \neg(x = 0) \textbf{ do skip}$$

cujo funcional é

$$(F'\ g)s = \begin{cases} g\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

Sabemos que qualquer função g de $\textbf{State} \hookrightarrow \textbf{State}$ satisfazendo $g\ s = s$ se $s\ x = 0$ é ponto fixo de F'. No entanto, nossa experiência computacional nos diz que

$$\mathcal{S}_{\mathrm{ds}}[\![S]\!]s_0 = \begin{cases} \underline{\text{undef}} & \text{se } s_0\ x \neq 0 \\ s_0 & \text{se } s_0\ x = 0 \end{cases}$$

de forma a representar o comportamento do loop.

Assim, nosso ponto fixo preferencial é

$$g_0\ s = \begin{cases} \underline{\text{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

A propriedade distintiva deste ponto fixo é que para qualquer outro ponto fixo g' de F', se $g_0\ s = s'$ então $g'\ s = s'$, mas não o contrário.

Generalizando para qualquer funcional F: O ponto fixo desejado deve ser alguma função parcial $g_0 : \mathbf{State} \hookrightarrow \mathbf{State}$ tal que:

- g_0 é um ponto fixo de F;
- se g é outro ponto fixo de F, então $g_0\ s = s'$ implica $g\ s = s'$, para todos os s e s'.

9.5 Teoria de Pontos Fixos

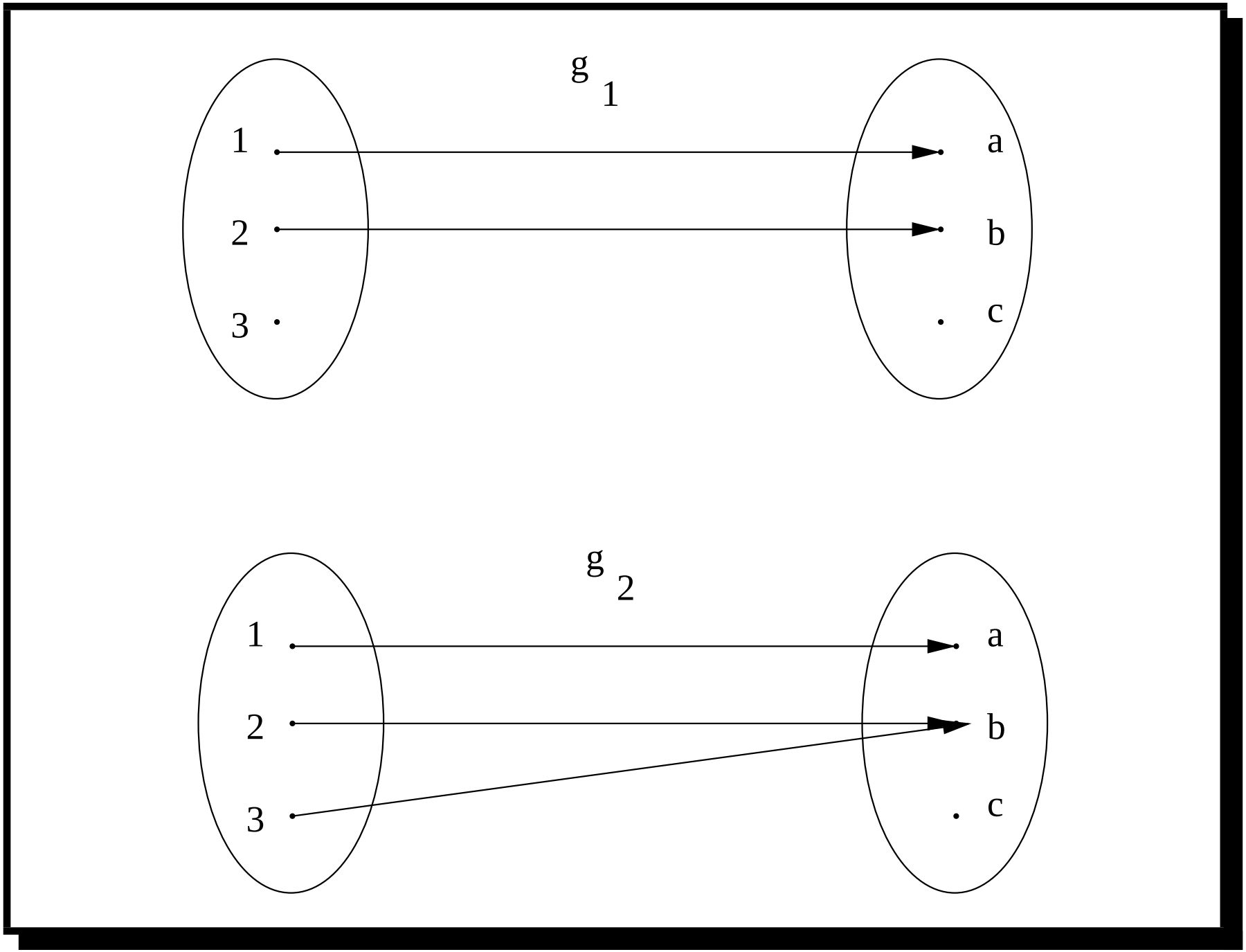
- Desenvolver uma teoria que permita tratar com funcionais e pontos fixos;
- Garantir a existência do ponto fixo preferencial.

Seja a ordem $\sqsubseteq$ sobre as funções parciais **State** $\hookrightarrow$ **State**:

$$g_1 \sqsubseteq g_2$$

quando a função parcial g_1 compartilha seus resultados com a função parcial g_2, ou seja, se $g_1\ s = s'$ então $g_2\ s = s'$, para todo s e s'.

$g_1 \sqsubseteq g_2$ significa que em todos os pontos em que g_1 é definido, g_2 também o é, e os seus valores são iguais, mas o inverso não é exigido. $g_1 \sqsubseteq g_2$ pode ser lido como g_1 é *menos ou igualmente definido* que g_2 ou g_1 *aproxima* g_2.


g 1
1
2
3
a
b
c
g 2
1
2
3
a
b
c

ex:

$$g_1\ s = s \text{ para todo } s$$

$$g_2\ s = \begin{cases} s & \text{se } s\ x \geq 0 \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

$$g_3\ s = \begin{cases} s & \text{se } s\ x = 0 \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

$$g_4\ s = \begin{cases} s & \text{se } s\ x \leq 0 \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

$$g_1 \sqsubseteq g_1$$

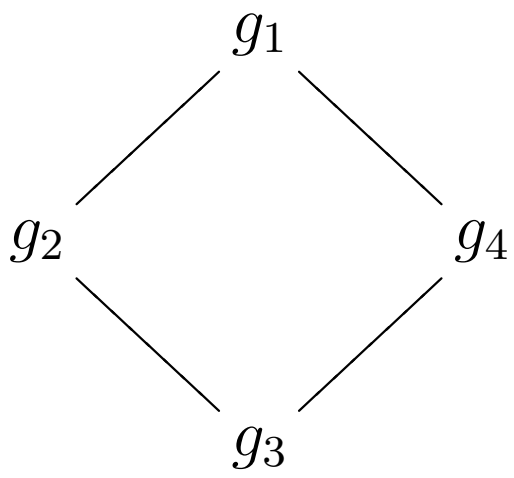
$$g_2 \sqsubseteq g_1 \quad g_2 \sqsubseteq g_2$$

$$g_3 \sqsubseteq g_1 \quad g_3 \sqsubseteq g_2 \quad g_3 \sqsubseteq g_3 \quad g_3 \sqsubseteq g_4$$

$$g_4 \sqsubseteq g_1 \quad g_4 \sqsubseteq g_4$$

$$g_2 \not\sqsubseteq g_4 \quad g_4 \not\sqsubseteq g_2$$

Diagrama de Hasse:



Exercício: Sejam g_1, g_2 e g_3, como a seguir definidos:

$$g_1\ s = \begin{cases} s & \text{se } s\ x \text{ é par} \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

$$g_2\ s = \begin{cases} s & \text{se } s\ x \text{ é um primo} \\ \underline{\text{undef}} & \text{caso contrário} \end{cases}$$

$$g_3\ s = s$$

1. Determine a ordem entre estas funções;
2. Determine uma função parcial g_4 tal que $g_4 \sqsubseteq g_1$, $g_4 \sqsubseteq g_2$ e $g_4 \sqsubseteq g_3$;
3. Determine uma função parcial g_5 tal que $g_1 \sqsubseteq g_5$, $g_2 \sqsubseteq g_5$ e $g_5 \sqsubseteq g_3$, e g_5 não é igual a g_1, g_2 nem a g_3.

Uma caracterização alternativa da ordem $\sqsubseteq$ em **State** $\hookrightarrow$ **State** é

$$g_1 \sqsubseteq g_2 \text{ se e somente se } \text{graph}(g_1) \subseteq \text{graph}(g_2)$$

onde $\text{graph}(f) = \{< x, y > \in X \times Y | f(x) = y\}$ e $f : X \hookrightarrow Y$.

O conjunto $\mathbf{State} \hookrightarrow \mathbf{State}$ munido da ordem $\sqsubseteq$ é um exemplo de um *conjunto parcialmente ordenado*.

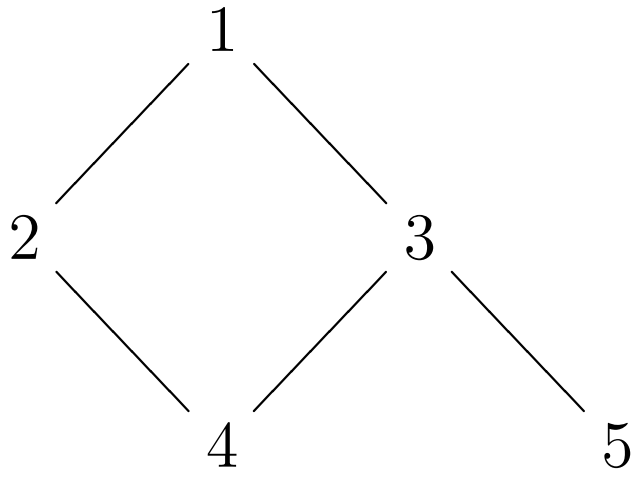
Seja D um conjunto e $d, d_1, d_2, d_3 \in D$. Um *conjunto parcialmente ordenado* é um par $(D, \sqsubseteq_D)$, onde $\sqsubseteq_D$ é uma relação em D satisfazendo:

1. $d \sqsubseteq_D d$ (reflexividade);
2. $d_1 \sqsubseteq_D d_2$ e $d_2 \sqsubseteq_D d_3$ implica $d_1 \sqsubseteq_D d_3$ (transitividade);
3. $d_1 \sqsubseteq_D d_2$ e $d_2 \sqsubseteq_D d_1$ implica $d_1 = d_2$ (anti-simetria).

Um elemento $d \in D$ satisfazendo $d \sqsubseteq d'$ para todo $d' \in D$ é chamado de *elemento mínimo* de D. Podemos dizer que ele não contém informação.

Teorema 5 *Se um conjunto parcialmente ordenado* $(D, \sqsubseteq_D)$ *tem um elemento mínimo* d*, então* d *é único.*

Prova: Assuma que D *tem dois elementos mínimos* d_1 *e* d_2*. Desde que* d_1 *é elemento mínimo,* $d_1 \sqsubseteq d_2$*. Como* d_2 *também é elemento mínimo,* $d_2 \sqsubseteq d_1$*. Pela propriedade da anti-simetria de* $\sqsubseteq$ *obtemos* $d_1 = d_2$*.*


1
2
3
4
5

ex: Seja S um conjunto não vazio e defina:

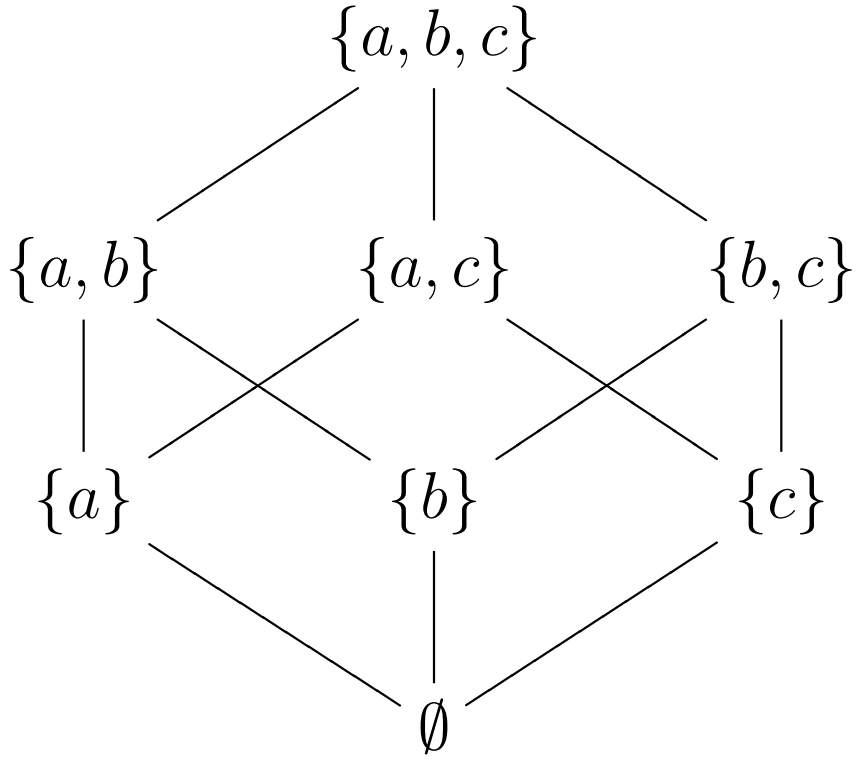
$$\mathcal{P}(S) = \{K | K \subseteq S\}$$

chamado *conjunto das partes*.

Então $(\mathcal{P}(S), \subseteq)$ é um conjunto parcialmente ordenado porque

- $\subseteq$ é reflexiva: $K \subseteq K$;
- $\subseteq$ é transitiva: se $K_1 \subseteq K_2$ e $K_2 \subseteq K_3$ então $K_1 \subseteq K_3$;
- $\subseteq$ é anti-simétrica: se $K_1 \subseteq K_2$ e $K_2 \subseteq K_1$ então $K_1 = K_2$.

No caso em que $S = \{a, b, c\}$,
$\mathcal{P}(S) = \{\emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$, e temos:



$(\mathcal{P}(S), \subseteq)$ tem um elemento mínimo, $\emptyset$.

Exercício: Mostre que $(\mathcal{P}(S), \supseteq)$ é um conjunto parcialmente ordenado e determine o elemento mínimo. Desenhe o diagrama de Hasse no caso em que $S = \{a, b, c\}$.

Lema 2 $(\mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq)$ *é um conjunto parcialmente ordenado. A função parcial* $\bot : \mathbf{State} \hookrightarrow \mathbf{State}$ *definida como*

$$\bot\ s = \underline{\text{undef}} \text{ para todo } s$$

é o elemento mínimo de $\mathbf{State} \hookrightarrow \mathbf{State}$.

Prova: Primeiro provamos que $\sqsubseteq$ *é ordem parcial:*

Reflexiva *Trivialmente* $g \sqsubseteq g$*, porque* $g\ s = s'$ *implica* $g\ s = s'$*;*

Transitiva *Assumimos* $g_1 \sqsubseteq g_2$ *e* $g_2 \sqsubseteq g_3$*. Assumimos* $g_1\ s = s'$ *e de* $g_1 \sqsubseteq g_2$ *concluimos* $g_2\ s = s'$*, e então* $g_2 \sqsubseteq g_3$ *fornece* $g_3\ s = s'$*;*

Anti-simétrica *Assumimos* $g_1 \sqsubseteq g_2$ *e* $g_2 \sqsubseteq g_1$*. Assumimos que* $g_1\ s = s'$*. Então,* $g_2\ s = s'$ *segue de* $g_1 \sqsubseteq g_2$*, e neste caso ambas as funções dão valores iguais. No caso em que* $g_1\ s = \underline{\text{undef}}$*, é necessário que* $g_2\ s = \underline{\text{undef}}$*, já que caso contrário,* $g_2\ s = s'$ *e de* $g_2 \sqsubseteq g_1$ *obteriamos* $g_1\ s = s'$*, que é uma contradição.*

Finalmente, temos que provar que $\perp$ *é o elemento mínimo de* $\mathbf{State} \hookrightarrow \mathbf{State}$. *É fácil ver que* $\perp$ *é um elemento de* $\mathbf{State} \hookrightarrow \mathbf{State}$ *e trivialmente* $\perp \sqsubseteq g$ *vale para todos* g, *dado que* $\perp\ s = s'$ *implica* $g\ s = s'$ *por vacuidade.*

Formalizando os requisitos de FIX F:

- FIX F é ponto fixo de F, isto é, $F(\text{FIX } F) = \text{FIX } F$;
- FIX F é o *menor* ponto fixo de F, isto é, se $F\ g = g$ então FIX $F \sqsubseteq g$.

Exercício: Mostre que se F tem um menor ponto fixo g_0 então g_0 é único.

Exercício: Determine o menor ponto fixo dos funcionais associados aos programas:

1. **while** $\neg(x = 0)$ **do** $x{:=}x - 1$
2. **while** $\neg(x = 1)$ **do** $(y{:=}y * x; x{:=}x - 1)$

- Seja $(D, \sqsubseteq)$, e assumimos um subconjunto $Y \subseteq D$;
- Existe um elemento de D que resume toda a informação de Y;
- Um elemento d é chamado de *limite superior* de Y se:

$$\forall d' \in Y \; d' \sqsubseteq d$$

- Um limite superior d de Y é um *menor limite superior* (ou *supremo*) se e somente se

$$d' \text{ é limite superior de } Y \text{ implica que } d \sqsubseteq d'$$

- O supremo de Y tem pouca informação a mais que a presente nos elementos de Y.

Exercício: Mostre que se Y tem um supremo, então ele é único.

Notação: o (único) supremo de Y é denotado por $\bigsqcup Y$.

Um subconjunto Y é chamado de *cadeia* se ele é consistente no sentido que quaisquer dois elementos de Y compartilham informação um com o outro, ou seja,

$$\forall d_1, d_2 \in Y \; d_1 \sqsubseteq d_2 \vee d_2 \sqsubseteq d_1$$

ex: Considere o conjunto parcialmente ordenado $(\mathcal{P}(\{a, b, c\}), \subseteq)$. Então, o subconjunto

$$Y_0 = \{\emptyset, \{a\}, \{a, c\}\}$$

é uma cadeia. Neste caso, $\{a, b, c\}$ e $\{a, c\}$ são limites superiores de Y_0 e $\{a, c\}$ é o supremo.

contra-ex: O subconjunto $\{\emptyset, \{a\}, \{c\}, \{a, c\}\}$ não é uma cadeia. Por que?

ex: Seja $g_n : \mathbf{State} \hookrightarrow \mathbf{State}$ definido como:

$$g_n\ s = \begin{cases} \underline{\text{undef}} & \text{se } s\ x > n \\ s[x \mapsto -1] & \text{se } 0 \leq s\ x \text{ e } s\ x \leq n \\ s & \text{se } s\ x < 0 \end{cases}$$

e $g_n \sqsubseteq g_m$ se $n \leq m$.

Seja $Y_0 = \{g_n | n \geq 0\}$. Y_0 é uma cadeia porque $g_n \sqsubseteq g_m$ se $n \leq m$. A função parcial

$$g\ s = \begin{cases} s[x \mapsto -1] & \text{se } 0 \leq s\ x \\ s & \text{se } s\ x < 0 \end{cases}$$

é o supremo de Y_0.

Exercício: Seja g_n a função parcial definida por:

$$g_n\ s = \begin{cases} s[y \mapsto (s\ x)!][x \mapsto 1] & \text{se } 0 < s\ x \text{ e } s\ x \leq n \\ \underline{\text{undef}} & \text{se } s\ x \leq 0 \text{ ou } s\ x > n \end{cases}$$

Definimos $Y_0 = \{g_n | n \geq 0\}$. Mostre que Y_0 é uma cadeia. Caracterize os limites superiores de Y_0 e determine o supremo.

Um conjunto parcialmente ordenado $(D, \sqsubseteq)$ é chamado de *cadeia completa parcialmente ordenada* (*ccpo*) se $\bigsqcup Y$ existe para todas as cadeias Y de D. Ele é um *reticulado completo* se $\bigsqcup Y$ existe para todos os subconjuntos Y de D.

Exercício: Mostre que $(\mathcal{P}(S), \subseteq)$ é um reticulado completo e um ccpo para todos os conjuntos não vazios S.

Contra-exemplo de ccpo: Seja
$\mathcal{P}_{\text{fin}}(S) = \{K \subseteq S | K \text{ é finito}\}$. Considere $(\mathcal{P}_{\text{fin}}(\mathbf{N}), \subseteq)$. Então

$$Y = \{\{0\}, \{0,1\}, \{0,1,2\}, \ldots, \{0,1,2,\ldots,n\}, \ldots\}$$

é cadeia. A união contável dos elementos A_i da cadeia é

$$\bigcup_{i=1}^{\infty} A_i = \mathbf{N}\,,$$

porém ele não pode ser o supremo da cadeia pois não pertence a ela.

Teorema 6 *Se* $(D, \sqsubseteq)$ *é um ccpo então existe um elemento mínimo* $\perp$ *dado por* $\perp = \bigsqcup \emptyset$.

Prova:

1. $\emptyset$ *é uma cadeia pois* $\forall d_1, d_2 \in \emptyset \; d_1 \sqsubseteq d_2 \vee d_2 \sqsubseteq d_1$ *(por vacuidade);*
2. *Como* $(D, \sqsubseteq)$ *é uma ccpo então existe* $\bigsqcup \emptyset$ *pois* $\emptyset \subseteq D$ *é cadeia de* D*;*
3. $\bigsqcup \emptyset \sqsubseteq d$, $d \in D$, *então* $\bigsqcup \emptyset$ *é elemento mínimo de* D.

Lema 3 $(\mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq)$ *é um ccpo. O supremo* $\bigsqcup Y$ *da cadeia* Y *de* $\mathbf{State} \hookrightarrow \mathbf{State}$ *é dado por:*

$$\mathrm{graph}(\bigsqcup Y) = \bigcup \{\mathrm{graph}(g) | g \in Y\}$$

isto é, $(\bigsqcup Y)\ s = s'$ *se e somente se* $g\ s = s'$ *para algum* $g \in Y$*.*

Observações:

- $\mathrm{graph}(f) = \{< x, y > \in X \times Y | f(x) = y\}$ e $f : X \to Y$;
- $\mathbf{State} \hookrightarrow \mathbf{State}$ não é um reticulado, mas é um ccpo e suas cadeias possuem supremo.

Sejam $(D, \sqsubseteq)$ e $(D', \sqsubseteq')$ duas ccpo's e considere a função (total) $f : D \to D'$. f é *monotônica* se e somente se

$$d_1 \sqsubseteq d_2 \text{ implica } f(d_1) \sqsubseteq' f(d_2)$$

para todos os d_1 e d_2.

Exercício: Considere o ccpo $(\mathcal{P}(\mathbf{N}), \subseteq)$. Determine quais das seguintes funções em $\mathcal{P}(\mathbf{N}) \to \mathcal{P}(\mathbf{N})$ são monotônicas:

- $f_1(X) = \mathbf{N} \backslash X$
- $f_2(X) = X \bigcup \{\mathbf{27}\}$
- $f_3(X) = X \bigcap \{\mathbf{7}, \mathbf{9}, \mathbf{13}\}$
- $f_4(X) = \{n \in X | n \text{ é primo}\}$
- $f_5(X) = \{\mathbf{2} * n | n \in X\}$

Exercício: Determine quais dos seguintes funcionais em $(\mathbf{State} \hookrightarrow \mathbf{State}) \to (\mathbf{State} \hookrightarrow \mathbf{State})$ são monotônicos:

- $F_0\ g = g$
- $F_1\ g = \begin{cases} g_1 & \text{se } g = g_2 \\ g_2 & \text{caso contrário} \end{cases}$ onde $g_1 \neq g_2$
- $(F'\ g)\ s = \begin{cases} g\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$

Teorema 7 *Sejam* $(D, \sqsubseteq)$, $(D', \sqsubseteq')$ *e* $(D'', \sqsubseteq'')$ *três ccpo's e sejam* $f : D \to D'$ *e* $f' : D' \to D''$ *duas funções monotônicas. Então,* $f' \circ f : D \to D''$ *é uma função monotônica.*

Prova: Assumimos que $d_1 \sqsubseteq d_2$. *A monotonicidade de* f *resulta em* $f(d_1) \sqsubseteq' f(d_2)$. *A monotonicidade de* f' *resulta em* $f'(f(d_1)) \sqsubseteq'' f'(f(d_2))$, *como queriamos provar.*

Observação: Isto significa que a operação de composição de funções preserva a monotonicidade.

Lema 4 *Sejam* $(D, \sqsubseteq)$ *e* $(D', \sqsubseteq')$ *duas ccpo's e seja* $f : D \to D'$ *uma função monotônica. Se* Y *é cadeia de* D *então* $\{f(d)|d \in Y\}$ *é uma cadeia de* D'*. Além disto,*

$$\sqcup' \{f(d)|d \in Y\} \sqsubseteq' f(\sqcup Y)$$

Observação: Isto significa que:

- a imagem de uma cadeia sob uma função monotônica também é uma cadeia;
- o supremo desta segunda cadeia aproxima a imagem do supremo da primeira.

Prova: Se $Y = \emptyset$ o enunciado é trivialmente válido porque

$$\begin{aligned} \sqcup' \{f(d) | d \in \emptyset\} &\sqsubseteq' f(\perp) \\ \sqcup' \emptyset &\sqsubseteq' f(\perp) \\ \perp' &\sqsubseteq' f(\perp) \end{aligned}$$

Assumimos $Y \neq \emptyset$. Seja $A = \{f(d) | d \in Y\}$ e sejam $d'_1, d'_2 \in A$. Então, existe $d_1, d_2 \in Y$ tal que $d'_1 = f(d_1)$ e $d'_2 = f(d_2)$. Desde que Y é cadeia então $d_1 \sqsubseteq d_2$ ou $d_2 \sqsubseteq d_1$.

Pela monotonicidade de f vale o mesmo entre d'_1 e d'_2. Ou seja, A é cadeia de D'.

Para a segunda parte da prova, assumimos $d \in Y$. Então, $d \sqsubseteq \sqcup Y$, e pela monotonicidade de f, $f(d) \sqsubseteq' f(\sqcup Y)$. Desde que isto vale para todos os $d \in Y$, então $f(\sqcup Y)$ é limite superior de A.

Estamos interessados em funções f que preservem os limites superiores de cadeias, ou seja, que satisfazem:

$$\sqcup' \{f(d) | d \in Y\} = f(\sqcup Y) \tag{1}$$

Uma função $f : D \to D'$ definida sobre duas ccpo's $(D, \sqsubseteq)$ e $(D', \sqsubseteq')$ é *contínua* se ela é monotônica e (1) vale para todas as cadeias não vazias Y.

Observação: Isto significa que obtemos a mesma informação independente de determinarmos o supremo antes ou depois de aplicar f.

Se vale também $\perp = f(\perp)$ então dizemos que f é *estrita*.

Lema 5 *Sejam* $(D, \sqsubseteq)$,$(D', \sqsubseteq')$ *e* $(D'', \sqsubseteq'')$*três ccpo's e sejam* $f : D \to D'$ *e* $f' : D' \to D''$ *duas funções contínuas. Então,* $f' \circ f : D \to D''$ *é uma função contínua.*

Prova: Do Teorema 7 sabemos que $f' \circ f$ *é monotônica. Da continuidade de* f *obtemos:*

$$\sqcup' \{f(d) | d \in Y\} = f(\sqcup Y)$$

Desde que $\{f(d)|d \in Y\}$ *é uma cadeia não vazia de* D' *e usando a continuidade de* f' *obtemos:*

$$\sqcup''\{f'(d')|d' \in \{f(d)|d \in Y\}\} = f'(\sqcup'\{f(d)|d \in Y\})$$

que é equivalente a

$$\sqcup''\{f'(f(d))|d \in Y\} = f'(f(\sqcup Y))$$

provando o resultado.

Exercício: Prove que se f e f' são estritas, $f' \circ f$ também o é.

Exercício: Mostre que o funcional associado ao programa

$$\textbf{while } \neg(x = 0) \textbf{ do } x{:=}x - 1$$

é contínuo.

Contra-exemplo de função contínua: Seja
$(\mathcal{P}(\mathbf{N} \cup \{a\}), \subseteq)$ um ccpo. Considere a função
$f : \mathcal{P}(\mathbf{N} \cup \{a\}) \to \mathcal{P}(\mathbf{N} \cup \{a\})$ definida como:

$$f\ X = \begin{cases} X & \text{se } X \text{ é finito} \\ X \cup \{a\} & \text{se } X \text{ é infinito} \end{cases}$$

f é monotônica, já que $X_1 \subseteq X_2$ implica $f\ X_1 \subseteq f\ X_2$.
No entanto, não é contínua. Considere
$Y = \{\{0, 1, \ldots, n\} | n \geq 0\}$ que é uma cadeia, com
$\sqcup Y = \mathbf{N}$. Aplicando f obtemos:
$\sqcup\{f\ X | X \in Y\} = \sqcup Y = \mathbf{N}$ e $f(\sqcup Y) = f\ \mathbf{N} = \mathbf{N} \cup \{a\}$.

Teorema 8 *Seja $f : D \to D$ uma função contínua sobre um ccpo $(D, \sqsubseteq)$ com elemento mínimo $\bot$. Então,*

$$\text{FIX } f = \bigsqcup \{f^n(\bot) | n \geq 0\}$$

define um elemento de D e este elemento é o menor ponto fixo de f.

Observações:

- $f^0 = \text{id}$ e $f^{n+1} = f \circ f^n$ para $n \geq 0$;
- Esta é a definição de ponto fixo desejado.

Prova:

1. $f^0(\bot) = \bot$, e $\bot \sqsubseteq d$ para todo $d \in D$. Por indução podemos mostrar que $f^n(\bot) \sqsubseteq f^n(d)$, já que f é monotônica. Segue que $f^n(\bot) \sqsubseteq f^m(\bot)$ para $n \leq m$. Assim, $\{f^n(\bot) | n \geq 0\}$ é uma cadeia de D e FIX f existe porque D é um ccpo;

2. Precisamos mostrar que FIX f é um ponto fixo, isto é, $f(\text{FIX } f) = \text{FIX } f$:

$$\begin{aligned} f(\text{FIX } f) &= f(\sqcup\{f^n(\bot)|n \geq 0\}) \\ &= \sqcup\{f(f^n(\bot))|n \geq 0\} \\ &= \sqcup\{f^n(\bot)|n \geq 1\} \\ &= \sqcup(\{f^n(\bot)|n \geq 1\} \cup \{\bot\}) \\ &= \sqcup\{f^n(\bot)|n \geq 0\} \\ &= \text{FIX } f \end{aligned}$$

3. Para mostrar que FIX f é o menor ponto fixo basta assumir que d é algum outro ponto fixo. Assim, $\perp \sqsubseteq d$ e pela monotonicidade de f obtemos $f^n(\perp) \sqsubseteq f^n(d)$, para $n \geq 0$. Como d é ponto fixo, $f^n(\perp) \sqsubseteq d$, e sabendo que FIX f é supremo da cadeia $\{f^n(\perp) | n \geq 0\}$, FIX $f \sqsubseteq d$, o que prova o resultado.

ex: Considere o funcional:

$$(F'\ g)\ s = \begin{cases} g\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

Tomamos $\perp\ s = \underline{\text{undef}}$ para todo s, que é o menor elemento de $\mathbf{State} \hookrightarrow \mathbf{State}$, para determinar os elementos de $\{F'^n(\perp) | n \geq 0\}$.

$$
\begin{aligned}
(F'^0\ \bot)\ s &= (\mathrm{id}\ \bot)\ s \\
&= \underline{\mathrm{undef}} \\
(F'^1\ \bot)\ s &= (F'\ \bot)\ s \\
&= \begin{cases} \bot\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases} \\
&= \begin{cases} \underline{\mathrm{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
(F'^2\ \bot)\ s &= F'(F'^1\ \bot)\ s \\
&= \begin{cases} (F'^1\ \bot)\ s & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases} \\
&= \begin{cases} \underline{\text{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}
\end{aligned}
$$

No caso geral, nos temos $F'^n\ \bot = F'^{n+1}\ \bot$ para $n > 0$. Além disto,

$$\sqcup\{F'^n\ \bot | n \geq 0\} = \sqcup\{F'^0\ \bot, F'^1\ \bot\} = F'^1\ \bot$$

porque $F'^0\ \bot = \bot$. Assim, o menor ponto fixo de F' é a função

$$g_1\ s = \begin{cases} \underline{\text{undef}} & \text{se } s\ x \neq 0 \\ s & \text{se } s\ x = 0 \end{cases}$$

Exercício: Determine o ponto fixo dos funcionais associados aos seguintes comandos:

1. **while** $\neg(x = 0)$ **do** $x{:=}x - 1$
2. **while** $\neg(x = 1)$ **do** $(y{:=}y * x; x{:=}x - 1)$

9.6 Existência

Teorema 9 *As equações da semântica denotacional definem uma função total* $\mathcal{S}_{\mathrm{ds}}$ *em*
$\mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$.

9.7 Um Resultado de Equivalência

Teorema 10 *Para todo comando S da linguagem exemplo While, nós temos* $\mathcal{S}_{\mathrm{SOS}}[\![S]\!] = \mathcal{S}_{\mathrm{ds}}[\![S]\!]$.

10 Semântica Axiomática

- Correção parcial: estabelece certa relação entre os valores iniciais e finais das variáveis (se o programa terminar);
- Correção parcial+terminação=correção total

10.1 Provas Diretas de Correção de Programas

10.1.1 Semântica Natural

$$y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y * x; x{:=}x - 1)$$

Para todos os estados s e s', se

$$\langle y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y * x; x{:=}x - 1), s\rangle \rightarrow s'$$

então $s'\ y = (s\ x)!$ e $s\ x > 0$.

A prova tem três estágios:

1. Provar que se $\langle y{:=}y * x; x{:=}x - 1, s\rangle \rightarrow s''$ e $s''\ x > 0$ então $(s\ y) * (s\ x)! = (s''\ y) * (s''\ x)!$ e $s\ x > 0$;

2. Provar que se
   $\langle \textbf{while}\ \neg(x = 1)\ \textbf{do}\ (y{:=}y * x; x{:=}x - 1), s\rangle \rightarrow s''$
   então $(s\ y) * (s\ x)! = s''\ y$ e $s''\ x = 1$ e $s\ x > 0$;

3. Provar que se
   $\langle y{:=}1; \textbf{while}\ \neg(x = 1)\ \textbf{do}\ (y{:=}y * x; x{:=}x - 1), s\rangle \rightarrow$
   s' então $s'\ y = (s\ x)!$ e $s\ x > 0$.

1. Considere $\langle y{:=}y * x; x{:=}x - 1, s\rangle \rightarrow s''$. De acordo com $[\text{comp}_{\text{ns}}]$ existem transições $\langle y{:=}y * x, s\rangle \rightarrow s'$ e $\langle x{:=}x - 1, s'\rangle \rightarrow s''$ para algum s'. De $[\text{ass}_{\text{ns}}]$ temos $s' = s[y \mapsto \mathcal{A}[\![y * x]\!]s]$ e $s'' = s'[x \mapsto \mathcal{A}[\![x - 1]\!]s']$. Combinando obtemos
$s'' = s[y \mapsto (s\ y) * (s\ x)][x \mapsto (s\ x) - 1]$. Assumindo $s''\ x > 0$ calculamos

$$(s''\ y)*(s''\ x)! = ((s\ y)*(s\ x))*((s\ x)-1)! = (s\ y)*(s\ x)!$$

e desde que $s\ x = (s''\ x) + 1$, isto prova a primeira parte;

2. Podem ser usadas duas regras:
   - $[\text{while}^{\text{ff}}_{\text{ns}}]$: Neste caso, $s' = s$ e $\mathcal{B}[\![\neg(x = 1)]\!]s = \mathbf{ff}$. Isto significa que $s'\ x = 1$ e, como $1! = 1$, nós obtemos $(s\ y) * (s\ x)! = s\ y$ e $s\ x > 0$;

- [$\text{while}^{\text{tt}}_{\text{ns}}$]: Neste caso, $\mathcal{B}[\![\neg(x = 1)]\!]s = \mathbf{tt}$ e $\langle y{:=}y * x; x{:=}x - 1, s\rangle \rightarrow s''$ e $\langle \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y * x; x{:=}x - 1), s''\rangle \rightarrow s'$ para algum s''. A hipótese indutiva diz que $(s''\ y) * (s''\ x)! = s'\ y$ e $s'\ x = 1$ e $s\ x > 0$. Da primeira parte da prova vem que $(s\ y) * (s\ x)! = (s''\ y) * (s''\ x)!$ e $s\ x > 0$. Combinando os resultados obtem-se $(s\ y) * (s\ x)! = s'\ y$ e $s\ x > 0$ que prova a segunda parte;

3. Considere para a prova da terceira parte

   $\langle y{:=}1; \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y * x; x{:=}x - 1), s\rangle \rightarrow s'$

   De acordo com $[\text{comp}_{\text{ns}}]$ existirá um estado s'' tal que
   $\langle y{:=}1, s\rangle \rightarrow s''$ e
   $\langle \mathbf{while}\ \neg(x = 1)\ \mathbf{do}\ (y{:=}y * x; x{:=}x - 1), s''\rangle \rightarrow s'$
   Do axioma $[\text{ass}_{\text{ns}}]$ obtemos $s'' = s[y \mapsto 1]$ e da
   segunda parte sabemos que $s''\ x > 0$ e $s\ x > 0$.
   Então, $(s\ x)! = (s''\ y) * (s''\ x)!$ vale e usando a
   segunda parte da prova obtemos
   $(s\ x)! = (s''\ y) * (s''\ x)! = s'\ y$ que prova a terceira e
   última parte.

Exercício: Prove a correção parcial do programa:

$$z{:=}0; \textbf{while } y \leq x \textbf{ do } (z{:=}z+1; x{:=}x-y)$$

Ou seja, se o programa pára no estado s' a partir de um estado s, com $s\ x > 0$ e $s\ y > 0$, então
$s'\ z = (s\ x)$ div $(s\ y)$ e $s'\ x = (s\ x)\ \mod (s\ y)$.

10.1.2 Semântica Operacional Estruturada

Para todos os estados s e s', se

$$\langle y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y * x; x{:=}x - 1), s\rangle \Rightarrow^* s'$$

então $s'\ y = (s\ x)!$ e $s\ x > 0$.

A prova tem dois estágios:

1. Prova-se por indução sobre o tamanho das seqüências de derivação que se
   $\langle \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y * x; x{:=}x - 1), s\rangle \Rightarrow^k s'$
   então $s'\ y = (s\ y) * (s\ x)!$ e $s'\ x = 1$ e $s\ x > 0$;

2. Provar que se
   $\langle y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y * x; x{:=}x - 1), s\rangle \Rightarrow^*$
   s' então $s'\ y = (s\ x)!$ e $s\ x > 0$.

10.1.3 Semântica Denotacional

Baseado em uma propriedade, como um predicado ψ sobre o ccpo $(\mathbf{State} \hookrightarrow \mathbf{State}, \sqsubseteq)$, isto é

$$\psi : (\mathbf{State} \hookrightarrow \mathbf{State}) \to \mathbf{T}$$

ex: (fatorial)

$\psi_{\text{fat}}(\mathcal{S}_{\text{ds}}[\![y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}y{*}x; x{:=}x{-}1)]\!]) = \textbf{tt}$

onde ψ_{fat} é definida como:

$$\psi_{\text{fat}}(g) = \textbf{tt}$$

se e somente se, para todos os estados s e s', se $g\ s = s'$ então $s'\ y = (s\ x)!$ e $s\ x > 0$.

Um predicado $\psi : D \to \mathbf{T}$ definido sobre um ccpo $(D, \sqsubseteq)$ é chamado de *admissível* se e somente se nós temos

$$\text{se } \psi(d) = \mathbf{tt} \text{ para todo } d \in Y \text{ então } \psi(\sqcup Y) = \mathbf{tt}$$

para toda cadeia Y de D.

Teorema 11 *Seja* $(D, \sqsubseteq)$ *um ccpo e sejam* $f : D \to D$ *uma função contínua e* ψ *um predicado admissível sobre* D. *Se para todo* $d \in D$

$$\psi(d) = \mathbf{tt} \text{ implica } \psi(f\ d) = \mathbf{tt}$$

então $\psi(\text{FIX } f) = \mathbf{tt}$.

Observação: Isto significa que se a aplicação de f não muda o valor do predicado ψ então o ponto fixo de f também fornece o mesmo valor para o predicado. Assim, podemos generalizar o predicado para o ponto fixo de f.

10.2 Asserções para Correção Parcial

- As abordagens anteriores são muito detalhadas para um uso prático pois elas são muito próximas da semântica de linguagens de programação;
- Precisamos extrair as propriedades essenciais das construções da linguagem;

- Podemos especificar estas propriedades dos programas como *asserções* na forma $\{P\}\ S\ \{Q\}$, que significa:
  1. Se P vale no estado inicial e
  2. a execução de S termina quando inicia neste estado
  3. então Q vale no estado em que S pára.
- Observe que $\{P\}\ S\ \{Q\}$ não garante a terminação.

$$\{P\}\ S\ \{Q\}$$

P chamada de *precondição*;

Q chamada de *pós-condição*.

ex:

$$\{x = n\}$$

$$y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}x * y; x{:=}x - 1)$$

$$\{y = n! \wedge n > 0\}$$

Observação: há dois tipos de variáveis:

1. variáveis do programa. ex: x e y;
2. variáveis lógicas. ex: n.

Notação:

$P_1 \wedge P_2$	para	P onde $P\ s = (P_1\ s)$ e $(P_2\ s)$
$P_1 \vee P_2$	para	P onde $P\ s = (P_1\ s)$ ou $(P_2\ s)$
$\neg P$	para	P' onde $P'\ s = \neg(P\ s)$
$P[x \mapsto \mathcal{A}[\![a]\!]]$	para	P' onde $P'\ s = P\ (s[x \mapsto \mathcal{A}[\![a]\!]s])$
$P_1 \Rightarrow P_2$	para	$\forall s \in \mathbf{State}\ P_1\ s$ implica $P_2\ s$

$$[\text{ass}_\text{P}] \qquad \{P[x \mapsto \mathcal{A}[\![a]\!]]\}\ x := a\ \{P\}$$

$$[\text{skip}_\text{P}] \qquad \{P\}\ \textbf{skip}\ \{P\}$$

$$[\text{comp}_\text{P}] \qquad \frac{\{P\}\ S_1\ \{Q\},\ \{Q\}\ S_2\ \{R\}}{\{P\}\ S_1;S_2\ \{R\}}$$

$$[\text{if}_\text{P}] \qquad \frac{\{\mathcal{B}[\![b]\!]\wedge P\}\ S_1\ \{Q\},\ \{\neg\mathcal{B}[\![b]\!]\wedge P\}\ S_2\ \{Q\}}{\{P\}\ \textbf{if}\ b\ \textbf{then}\ S_1\ \textbf{else}\ S_2\ \{Q\}}$$

$$[\text{while}_\text{P}] \qquad \frac{\{\mathcal{B}[\![b]\!]\wedge P\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{\neg\mathcal{B}[\![b]\!]\wedge P\}}$$

$$[\text{cons}_\text{P}] \qquad \frac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}}\ \text{se}\ P \Rightarrow P'\ \text{e}\ Q' \Rightarrow Q$$

Explicações:

$$\{P[x \mapsto \mathcal{A}[\![a]\!]]\}\ x := a\ \{P\}$$

ex: $x := x + 1$ (a é $x + 1$) e P é $x = 5$.

$$\{(x = 5)[x \mapsto \mathcal{A}[\![x + 1]\!]]\}\ x := x + 1\ \{x = 5\}$$

$$\{\mathcal{A}[\![x + 1]\!] = 5\}\ x := x + 1\ \{x = 5\}$$

$$\{x + 1 = 5\}\ x := x + 1\ \{x = 5\}$$

$$\{x = 4\}\ x := x + 1\ \{x = 5\}$$

$$\frac{\{\mathcal{B}[\![b]\!] \wedge P\}\ S\ \{P\}}{\{P\}\ \textbf{while}\ b\ \textbf{do}\ S\ \{\neg\mathcal{B}[\![b]\!] \wedge P\}}$$

- P é o *invariante* do laço;
- Claramente o predicado b é verdadeiro imediatamente antes de S e falso ao final do laço.

$$\frac{\{P'\}\ S\ \{Q'\}}{\{P\}\ S\ \{Q\}} \text{ se } P \Rightarrow P' \text{ e } Q' \Rightarrow Q$$

Esta regra se chama *regra da consequência* e significa que podemos *fortalecer* a precondição P' e *enfraquecer* a pós-condição Q'.

A tabela mostrada especifica um sistema formal de inferência para determinar provas de propriedades:

$$\vdash_P \{P\}\ S\ \{Q\}$$

ex:

$$\{x = n\}$$

$$y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}x * y; x{:=}x - 1)$$

$$\{y = n! \wedge n > 0\}$$

e usamos $y = n! \wedge n > 0$ para representar o predicado P onde $P\ s = (s\ y = (s\ n)! \wedge s\ n > 0)$.

O invariante do laço while é INV:

$INV\ s = (s\ x > 0 \text{ implica } ((s\ y)*(s\ x)! = (s\ n)! \text{ e } s\ n \geq s\ x))$

Usando [ass_P]:

$$\vdash_P \ \{INV[x \mapsto x-1]\}\ x{:=}x-1\ \{INV\}$$

De forma similar:

$$\vdash_P \ \{(INV[x \mapsto x{-}1])[y \mapsto y{*}x]\}\ y{:=}y{*}x\ \{INV[x \mapsto x{-}1]\}$$

Usando [comp_P]:

$$\vdash_P \ \{(INV[x \mapsto x{-}1])[y \mapsto y{*}x]\}\ y{:=}y{*}x; x{:=}x{-}1\ \{INV\}$$

Sabemos que
$(\neg(x = 1) \wedge INV) \Rightarrow (INV[x \mapsto x - 1])[y \mapsto y * x]$.
Usando $[\text{cons}_\text{P}]$:

$$\vdash_P \ \{\neg(x = 1) \wedge INV\}\ y{:=}y * x; x{:=}x - 1\ \{INV\}$$

Usando $[\text{while}_\text{P}]$:

$$\vdash_P \ \{INV\}$$

$$\textbf{while}\ \neg(x = 1)\ \textbf{do}\ (y{:=}x * y; x{:=}x - 1)$$

$$\{\neg(\neg(x = 1)) \wedge INV\}$$

Sabemos que $\neg(\neg(x = 1)) \wedge INV \Rightarrow y = n! \wedge n > 0$ e aplicando $[\text{cons}_\text{P}]$:

$$\vdash_P \ \{INV\}$$

$$\textbf{while } \neg(x = 1) \textbf{ do } (y{:=}x * y; x{:=}x - 1)$$

$$\{y = n! \wedge n > 0\}$$

Aplicando [ass_P] ao comando $y{:=}1$:

$$\vdash_P \ \{INV[y \mapsto 1]\}\ y{:=}1\ \{INV\}$$

Usando $x = n \Rightarrow INV[y \mapsto 1]$ em [cons_P] obtemos:

$$\vdash_P \ \{x = n\}\ y{:=}1\ \{INV\}$$

Finalmente, usando $[\text{comp}_\text{P}]$ obtemos:

$$\vdash_P \ \{x = n\}$$

$$y{:=}1; \textbf{while } \neg(x = 1) \textbf{ do } (y{:=}x * y; x{:=}x - 1)$$

$$\{y = n! \wedge n > 0\}$$

Exercício: Use a semântica axiomática para verificar a correção parcial do programa

$$\{x = a \wedge y = b\}$$

$$z{:=}0; \textbf{while } y \leq x \textbf{ do } (z{:=}z + 1; x{:=}x - y)$$

$$\{z = (a \text{ div } b) \wedge x = (a \text{ mod } b) \wedge a > 0 \wedge b > 0\}$$

10.3 Correção e Completude do Sistema Formal

Teorema 12 *Para toda asserção de correção parcial* $\{P\}\ S\ \{Q\}$ *temos* $\models_P \{P\}\ S\ \{Q\}$ *se e somente se* $\vdash_P \{P\}\ S\ \{Q\}$.

10.4 Asserções para Correção Total

A fórmula $\{P\}\ S\ \{\Downarrow Q\}$ significa:

1. se a precondição P é válida
2. então S garantidamente pára (representado pelo símbolo $\Downarrow$)
3. e o estado final satisfaz a pós-condição Q.

$$[\text{ass}_\text{t}] \qquad \{P[x \mapsto \mathcal{A}[\![a]\!]]\}\ x{:=}a\ \{\Downarrow P\}$$

$$[\text{skip}_\text{t}] \qquad \{P\}\ \mathbf{skip}\ \{\Downarrow P\}$$

$$[\text{comp}_\text{t}] \qquad \frac{\{P\}\ S_1\ \{\Downarrow Q\},\ \ \{Q\}\ S_2\ \{\Downarrow R\}}{\{P\}\ S_1;S_2\ \{\Downarrow R\}}$$

$$[\text{if}_\text{t}] \qquad \frac{\{\mathcal{B}[\![b]\!]\wedge P\}\ S_1\ \{\Downarrow Q\},\ \ \{\neg\mathcal{B}[\![b]\!]\wedge P\}\ S_2\ \{\Downarrow Q\}}{\{P\}\ \mathbf{if}\ b\ \mathbf{then}\ S_1\ \mathbf{else}\ S_2\ \{\Downarrow Q\}}$$

$$[\text{while}_\text{t}] \qquad \frac{\{P(\mathbf{z}{+}\mathbf{1})\}\ S\ \{\Downarrow P(\mathbf{z})\}}{\{\exists \mathbf{z} P(\mathbf{z})\}\ \mathbf{while}\ b\ \mathbf{do}\ S\ \{\Downarrow P(\mathbf{0})\}}$$

onde $P(\mathbf{z}+\mathbf{1}) \Rightarrow \mathcal{B}[\![b]\!]$, $P(\mathbf{0}) \Rightarrow \neg\mathcal{B}[\![b]\!]$

e $\mathbf{z} \in \mathcal{N}$, $\mathbf{z} \geq \mathbf{0}$

$$[\text{cons}_\text{t}] \qquad \frac{\{P'\}\ S\ \{\Downarrow Q'\}}{\{P\}\ S\ \{\Downarrow Q\}}$$

onde $P \Rightarrow P'$ e $Q' \Rightarrow Q$

Observação: Na regra [while_t] nós usamos uma família parametrizada de predicados $P(\mathbf{z})$ para o invariante do laço. $\mathbf{z}$ indica o número de voltas do laço que ainda faltam. Isto significa que $P(\mathbf{0})$ implica que b é falso.