# Conteúdo



# 1 Resolução

**Regra da Resolução (geral)**

Dado $L = \{l_1, \ldots, l_n\}$ um conjunto de literais, seja $\overline{L} = \{\overline{l_1}, \ldots, \overline{l_n}\}$.

Sejam $C_1$ e $C_2$ duas cláusulas sem variáveis em comum. Seja $L_1 \subseteq C_1$ e $L_2 \subseteq C_2$, tal que $L_1$ e $\overline{L_2}$ unificam com umg $\sigma$.

A **resolvente** $C = Res(C_1, C_2)$ de $C_1$ e $C_2$ é:

$$(C_1\sigma - L_1\sigma) \cup (C_2\sigma L_2\sigma)$$

Sendo $\{p(f(x), g(y)), q(x, y)\}$ e $\{\neg p(f(f(a)), g(z)), q(f(a), g(z))\}$.

Um umg de $\{p(f(x), g(y))\}$ e $\{p(f(f(a)), g(z))\}$ é $[f(a)/x, y/z]$. Então, uma resolvente é

$$\{q(f(a), z), q(f(a), g(z))\}$$

**Algoritmo de Satisfabibilidade por Resolução (geral)**

**Entrada** Um conjunto de cláusulas $S$

**Saída** Se terminar, determina se as cláusulas são ou não satisfazíveis. Mas pode não terminar.

Seja $S_0 = S$.

No passo $i = 1, 2, \ldots$ escolher duas cláusulas $C_1$ e $C_2$ com resolvente $C = Res(C_1, C_2)$.

Se $C = []$, terminar e $S$ não é satisfazível. Caso contrário $S_{i+1} = S_i \cup \{C\}$.

Se $S_{i+1} = S_i$ não há mais pares de cláusulas para *resolver* então terminar e $S$ é satisfazível.

**Resolução para cláusulas de Horn**

Seja $\mathcal{P}$ um programa (conjunto de cláusulas de Horn positivas) e $G$ uma cláusula negativa (objectivo), $\leftarrow \beta_1, \ldots, \beta_n$.

Já vimos que

$$\mathcal{P} \models \exists \beta_1 \ \wedge \ \cdots \ \wedge \ \beta_n \text{ sse } \not\models P \ \wedge \ G$$

Sendo a resolução integra, basta que $\mathcal{P} \cup \{\neg G\} \vdash F$.

Mas então

$$\mathcal{P} \models (\beta_1 \ \wedge \ \cdots \ \wedge \ \beta_n)\theta$$

para alguma substituição $\theta$.

Mais especificamente:

**Resposta correcta**

Uma substituição de variáveis $\theta$ em $G$ é uma resposta correcta se $\mathcal{P} \models \forall(\neg G\theta)$ onde $\forall$ é o fecho universal das variáveis livres em $\neg G\theta$.

## 1.1 Resolução SLD

**Resolução para cláusulas de Horn**

**Exemplo 18.1.** *Considera o seguinte programa $\mathcal{P}$:*

$$\begin{array}{lcl} q(x,y) & \leftarrow & p(x,y) \\ q(x,y) & \leftarrow & p(x,z), p(z,y) \\ p(b,a) & & \\ p(c,a) & & \\ p(d,b) & & \\ p(e,b) & & \end{array}$$

**Resolução para cláusulas de Horn**

**Exemplo 18.2.** *Considera o objectivo: $\leftarrow q(y,b), q(b,z)$. Em cada passo escolher um literal que complemente a cabeça duma cláusula do programa (usar variáveis novas para o programa!).*

1. *Escolher $q(y,b)$ e resolver com a primeira cláusula ($q(x_1,y_1) \leftarrow p(x_1,y_1)$): $\leftarrow p(y,b), q(b,z)$ e a substituição $[y/x_1, b/y_1]$*
2. *Escolher $p(y,b)$ e resolver com a quinta cláusula: $\leftarrow q(b,z)$ e subsituição $[d/y]$*
3. *Resolver o literal que resta com a primeira cláusula: $\leftarrow p(b,z)$*
4. *Resolver o literal que resta com a terceira cláusula: $[]$ e a substituição $[a/z]$.*

*A resposta é $[d/y, a/z]$ e $\mathcal{P} \models q(d,b) \ \wedge \ (q(b,a)$. E também, $\mathcal{P} \models \exists y \exists z q(y,b) \ \wedge \ q(b,z)$ (como se pretendia...).*

### Regras de Computação e de Procura

#### Regra de Computação

Uma regra de computação é uma regra para determinar um literal do objectivo com o qual aplicar a regra de resolução (resolver).

#### Estratégia de Procura

Uma estratégia de procura é uma regra para determinar como escolher qual a cláusula do programa que se vai usar para resolver com o literal escolhido no objectivo.

#### Resolução SLD (Selectiva Linear para c. Definidas)

Seja $\mathcal{P}$ um programa definido, $R$ uma regra de computação e $G$ um objectivo. Uma derivação por resolução-SLD é uma sequência de passos de resolução entre as cláusulas do objectivo e as de programa. Seja $G_0 = G$. Dada a cláusula $G_i$, $G_{i+1}$ é obtida selecionando um literal $\beta_i$ de $G_i$, de acordo com $R$ e escolhendo $C_i \in P$ (com variáveis novas) tal que a cabeça de $C_i$ unifica com $\beta_i$ com umg $\theta_i$:

$$\begin{aligned} G_i &= \leftarrow \beta_1, \ldots, \beta_i, \beta_{i+1}, \ldots, \beta_n \\ C_i &= \alpha \leftarrow \alpha_1, \ldots, \alpha_k \\ \alpha\theta_i &= \beta_i\theta_i \\ G_{i+1} &= \leftarrow (\beta_1, \ldots, \alpha_1, \ldots, \alpha_k, \beta_{i+1}, \ldots, \beta_n)\theta_i \end{aligned}$$

Uma refutação SLD é uma derivação-SLD de []. Se $G_n = []$ dizemos que a refutação tem comprimento $n$.

**Exemplo 18.3.** *Seja P:*

**Refutação *SLD***

$\mathsf{Max(x, y, x)} \leftarrow \mathsf{Less\_eq(y, x)}$
$\mathsf{Max(x, y, y)} \leftarrow \mathsf{Less\_eq(x, y)}$
$\mathsf{Less\_eq(0, x)} \leftarrow$
$\mathsf{Less\_eq(s(x), s(y))} \leftarrow \mathsf{Less\_eq(x, y)}$

*e* $G : \leftarrow Max(s(0), x, s(s(0)))$

**Exemplo 18.4.** *Uma refutação para* $P \cup \{G\}$ *de comprimento* $n = 3$ *é:*

| | |
|---|---|
| $\leftarrow \mathsf{Max(s(0), x, s(s(0)))}$ | $(\mathsf{Max(x_1, y_1, y_1)} \leftarrow \mathsf{Less\_eq(x_1, y_1)}$ |
| | $\theta_1 = [\mathsf{s(0)/x_1, s(s(0))/x, s(s(0))/y_1}])$ |
| $\leftarrow \mathsf{Less\_eq(s(0), s(s(0)))}$ | $(\mathsf{Less\_eq(s(x_2), s(y_2))} \leftarrow \mathsf{Less\_eq(x_2, y_2)})$ |
| | $\theta_2 = [\mathsf{0/x_2, s(0)/y_2}])$ |
| $\leftarrow \mathsf{Less\_eq(0, s(0))}$ | $(\mathsf{Less\_eq(0, x_3)} \leftarrow, \theta_3 = [\mathsf{s(0)/x_3}])$ |
| $\epsilon$ | |

### Árvore de derivação SLD

Dado um programa definido $\mathcal{P}$, uma regra de computação e um objectivo $G$, todas as derivações-SLD podem ser representadas por uma árvore tal que:

**A raiz** é etiquetada por $G$

**Cada nó** é etiquetado com uma cláusula objectivo $G_i$ e cria-se um novo ramo para cada $G_{i_j}$ que pode ser obtido resolvendo $G_i$ com uma cláusula $C_j$ cuja cabeça unifique com um literal de $G_i$.

**As folhas** se forem $\epsilon$ são designadas nós de sucesso (que correspondem a soluções), senão nós de insucesso (ou falha).

### Árvore de derivação SLD

Os ramos da árvore podem então ser

**de sucesso** se terminarem numa folha de sucesso

**de falha** se terminarem numa folha de falha

**infinitos** se corresponderem a uma derivação infinita

**Exemplo 18.5.** *Determinar a árvore-SLD do exemplo 18.1, considerando como regra de computação a escolha do literal mais à esquerda.*

# 2 Prolog

### Prolog

De *PROgrammation en LOGique*

`Yap` primeiro compilador de Prolog desenvolvido no DCC desde os anos 1980s

**Regra de Computação** Escolhe o literal mais à esquerda num objectivo

**Regra de procura** Escolhe as cláusulas de um predicado de cima para baixo

**Retrocesso** Permite obter mais soluções da árvore SLD: ponto e vírgula (`;`)

**Unificação** Não faz verificação de ocorrência de variáveis

**Notação**

1. variáveis começam por maiúsculas
2. símbolos de predicado, constantes e símbolos funcionais começam com letras minúsculas
3. $\leftarrow$ é `:-`
4. Todas cláusulas terminam com ponto final `.`

# 3 Bases de conhecimento

**Prolog**

Uma base de conhecimento

```
filho(bob,pam).  %Bob é filho de Pam filho(bob,tom). filho(liz,tom).
filho(ann,bob). filho(pat,bob). filho(jim,pat).

mulher(pam). mulher(liz). mulher(ann). mulher(pat).

homem(tom). homem(bob). homem(jim).
```

**Prolog**

Define os predicados especificados abaixo.

| | |
|---|---|
| *mae*/2 | mae(X,Y) sucede se X é mãe de Y |
| *avo*/2 | avo(X,Z) sucede se X é avô ou avó de Z |
| *irma*/2 | irma(X,Y) sucede se X é irmã de Y |
| *antecessor*/2 | antecessor(X,Z) sucede se X é antecessor de Z |

**Prolog**

Caminhos num grafo acíclico dirigido

```
r(g,h).  r(g,d). r(e,d). r(h,f). r(e,f). r(a,e). r(a,b). r(b,f).
r(b,c). r(f,c).
```

**Prolog**

Define os predicados especificados abaixo.

| | |
|---|---|
| *path*/2 | path(X,Y) sucede se no grafo existir um caminho de X para Y |
| *path_n*/3 | path_n(X,N,Y) sucede se no grafo existir um caminho de comprimento N de X para Y |