

Lógica, Raciocínio Automatizado e PROLOG

Silvio do Lago Pereira

slago@ime.usp.br

1 Introdução

A *lógica* é um formalismo matemático através do qual podemos abstrair a estrutura de um argumento, eliminando a ambigüidade existente na linguagem natural. Esse formalismo é composto por uma linguagem formal e por um conjunto de regras de inferência que nos permitem analisar um argumento de forma precisa e decidir a sua validade [3,5,6].

Informalmente, um *argumento* é uma seqüência de *premissas* seguida de uma *conclusão*. Dizemos que um argumento é *válido* quando sua conclusão é uma conseqüência necessária de suas premissas. Por exemplo, o argumento

Sempre que chove, a marginal fica congestionada.

Está chovendo muito.

Logo, a marginal deve estar congestionada.

é válido; pois sua conclusão é uma conseqüência necessária de suas premissas.

Nesse artigo, vamos estudar a lógica proposicional como uma forma de introduzir os conceitos básicos da lógica dedutiva. Em seguida, estudaremos a lógica de predicados, um formalismo mais expressivo, que estende a lógica proposicional com variáveis e quantificadores. Finalmente, apresentaremos um algoritmo para raciocínio automatizado capaz de manipular uma base de conhecimento, especificada na linguagem da lógica de predicados. Conforme veremos, esse algoritmo (que constitui o núcleo da linguagem PROLOG [7]) poderá ser usado tanto para provar a validade de um argumento, quanto para recuperar ou gerar novas informações, através de raciocínio dedutivo.

2 Lógica proposicional

Uma *proposição* é uma declaração afirmativa à qual se pode associar um valor verdadeiro ou falso, mas não ambos. Por exemplo, “*O Brasil fica na América*” é uma proposição verdadeira, enquanto “*A lua é de queijo*” é uma proposição falsa. A proposição é o elemento básico a partir do qual os argumentos são construídos, sendo também o principal objeto de estudo na lógica proposicional.

2.1 Sintaxe da lógica proposicional

Os símbolos usados na lógica proposicional são as constantes \perp (*falso*) e \top (*verdade*), os símbolos proposicionais (*i.e.*, letras minúsculas do alfabeto latino, possivelmente indexadas) e os conectivos lógicos \neg (*não*), \wedge (*e*), \vee (*ou*) e \rightarrow (*então*). São *fórmulas bem-formadas* na lógica proposicional:

- as constantes \perp e \top (*valores-verdade*);
- os símbolos proposicionais;
- e, se α e β forem fórmulas bem-formadas¹, $\neg\alpha$, $\alpha \wedge \beta$, $\alpha \vee \beta$ e $\alpha \rightarrow \beta$.

Uma fórmula da forma $\neg\alpha$ é denominada *negação* da fórmula α e dizemos que α e $\neg\alpha$ são fórmulas *complementares*. Fórmulas da forma $\alpha \wedge \beta$ e $\alpha \vee \beta$ são denominadas, respectivamente, *conjunção* e *disjunção* das fórmulas α e β . Uma fórmula da forma $\alpha \rightarrow \beta$ é denominada *condicional*, sendo α o seu *antecedente* e β o seu *conseqüente*.

A ordem de precedência dos conectivos é (da maior para a menor): \neg , \wedge , \vee e \rightarrow . Caso uma ordem diferente seja desejada, podemos usar parênteses. Por exemplo, na fórmula $\neg p \wedge q$, a negação afeta apenas o símbolo proposicional p ; para que ela afete a conjunção de p e q , devemos escrever $\neg(p \wedge q)$.

Formalização de argumentos: Podemos usar a lógica proposicional para formalizar um argumento, *i.e.*, para abstrair a sua forma lógica. No processo de formalização, devemos reconhecer as proposições e conectivos que compõem o argumento, de modo que possamos expressá-lo usando fórmulas bem-formadas.

Como exemplo, vamos formalizar o seguinte argumento:

Se o time joga bem, ganha o campeonato.
Se o time não joga bem, o técnico é culpado.
Se o time ganha o campeonato, os torcedores ficam contentes.
Os torcedores não estão contentes.
Logo, o técnico é culpado.

Primeiro, associamos a cada proposição um símbolo proposicional distinto:

p : “o time joga bem”
 q : “o time ganha o campeonato”
 r : “o técnico é culpado”
 s : “os torcedores ficam contentes”

Em seguida, usando esses símbolos proposicionais, escrevemos as fórmulas correspondentes às sentenças do argumento:

¹ Usamos letras minúsculas do alfabeto grego para denotar fórmulas genéricas.

- (1) $p \rightarrow q$
- (2) $\neg p \rightarrow r$
- (3) $q \rightarrow s$
- (4) $\neg s$
- (5) r

Agora, podemos representar o argumento como $\{p \rightarrow q, \neg p \rightarrow r, q \rightarrow s, \neg s\} \models r$. A notação $\Delta \models \phi$ estabelece que a fórmula ϕ é uma consequência lógica do conjunto de fórmulas Δ . Para verificarmos a validade de uma tal afirmação, precisamos antes definir o significado das fórmulas envolvidas no argumento.

Exercício 1 Usando lógica proposicional, formalize as sentenças a seguir:

- Se Ana é alta e magra, então ela é elegante.
- Se Beto é rico, então ele não precisa de empréstimos.
- Se Caio ama a natureza, então ele ama as plantas e os animais.
- Se Denis jogar na loteria, então ele ficará rico ou desiludido.
- Se faz frio ou chove, então Eva fica em casa e vê tevê. □

Exercício 2 Usando a lógica proposicional, formalize os argumentos a seguir:

- Se o filme é bom, o cinema fica lotado. Como a crítica diz que o filme é muito bom, podemos imaginar que não encontraremos lugares livres.
- Sempre que chove à tarde, à noite, o trânsito na marginal do rio Tietê fica congestionado. Como agora à noite o trânsito na marginal está fluindo bem, concluímos que não choveu à tarde.
- Se existissem ET's, eles já nos teriam enviado algum sinal. Se nos tivessem enviado um sinal, teríamos feito contato. Portanto, se existissem ET's, já teríamos feito contato com eles. □

2.2 Semântica da lógica proposicional

O significado de uma fórmula bem-formada é derivado da interpretação de seus símbolos proposicionais e da *tabela-verdade* dos conectivos.

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$
⊥	⊥	⊤	⊥	⊥	⊤
⊥	⊤	⊤	⊥	⊤	⊤
⊤	⊥	⊥	⊥	⊤	⊥
⊤	⊤	⊥	⊤	⊤	⊤

Tabela 1. Tabela-verdade dos conectivos

Seja ϕ uma fórmula contendo os símbolos proposicionais p_1, \dots, p_n . Uma *interpretação* de ϕ é uma associação de valores-verdade aos símbolos p_1, \dots, p_n , tal que nenhum p_i seja associado a \perp e \top ao mesmo tempo. Uma interpretação *satisfaz* uma fórmula se essa fórmula é verdadeira sob essa interpretação.

Dada uma fórmula bem-formada, podemos criar uma tabela-verdade contendo uma linha para cada possível interpretação de seus símbolos proposicionais. A cada linha, calculamos o valor da fórmula. Caso a fórmula seja verdadeira em todas as interpretações possíveis, dizemos que ela é *válida* ou *tautológica*. Caso a fórmula seja falsa em todas as interpretações possíveis, dizemos que ela é *insatisfável* ou *contraditória*. Uma fórmula que não é tautológica nem contraditória é denominada *satisfável* [3,6].

Validade de argumentos. Tabelas-verdade também são usadas para se decidir a validade de fórmulas bem-formadas representando argumentos.

Dizemos que um argumento da forma $\{\alpha_1, \dots, \alpha_n\} \models \beta$ é *válido* se e somente se a fórmula $(\alpha_1 \wedge \dots \wedge \alpha_n) \rightarrow \beta$ for tautológica. Por exemplo, para decidir a validade do argumento $\{\neg p, \neg p \rightarrow q\} \models q$, basta construir a tabela-verdade para a fórmula $(\neg p \wedge (\neg p \rightarrow q)) \rightarrow q$ e verificar se ela é tautológica (veja a tabela 2).

p	q	$\neg p$	$(\neg p \rightarrow q)$	$(\neg p \wedge (\neg p \rightarrow q))$	$(\neg p \wedge (\neg p \rightarrow q)) \rightarrow q$
\perp	\perp	\top	\perp	\perp	\top
\perp	\top	\top	\top	\top	\top
\top	\perp	\perp	\top	\perp	\top
\top	\top	\perp	\top	\perp	\top

Tabela 2. Tabela-verdade para o argumento $\{\neg p, \neg p \rightarrow q\} \models q$

Dizemos que uma fórmula ϕ é uma *conseqüência lógica* de um conjunto de fórmulas $\Delta = \{\alpha_1, \dots, \alpha_n\}$, denotado por $\Delta \models \phi$, se e só se toda interpretação que satisfaz $\alpha_1 \wedge \dots \wedge \alpha_n$ também satisfaz ϕ .

Exercício 3 Usando tabela-verdade, verifique a validade dos argumentos a seguir:

- Se chove, a rua fica molhada. A rua está molhada. Logo, choveu.
- Se chove, a rua fica molhada. A rua não está molhada. Logo, não choveu.
- Se chove, a rua fica molhada. Não choveu. Logo, a rua está seca. \square

2.3 Regras de inferência

Embora a tabela-verdade seja um mecanismo bastante simples para verificar a validade de um argumento, dependendo do tamanho da fórmula, sua construção pode se tornar inviável. De modo geral, se uma fórmula contém n símbolos

proposicionais distintos, sua tabela-verdade terá 2^n linhas (uma linha para cada interpretação possível). Por exemplo, a tabela-verdade para o argumento $\{p \rightarrow q, \neg p \rightarrow r, q \rightarrow s, \neg s\} \models r$ teria $2^4 = 16$ linhas. Assim, quando o número de símbolos proposicionais numa fórmula é muito grande, um método mais eficiente para validação de argumentos, denominado prova, é necessário.

Prova. Uma *prova* de uma fórmula ϕ , a partir de um conjunto de fórmulas Δ , consiste numa seqüência finita de fórmulas $\gamma_1, \gamma_2, \dots, \gamma_n$, onde $\gamma_n = \phi$ e cada γ_i é uma fórmula em Δ ou é derivada a partir de uma regra de inferência aplicada a fórmulas em $\Delta \cup \{\gamma_1, \dots, \gamma_{i-1}\}$. Usamos a notação $\Delta \vdash \phi$ para indicar que a fórmula ϕ pode ser derivada a partir das fórmulas em Δ .

Regra de inferência. Uma *regra de inferência* é um padrão que estabelece como uma nova fórmula pode ser gerada a partir de outras duas. As regras de inferência clássicas (*modus ponens*, *modus tollens* e *silogismo hipotético*) representam formas de raciocínio dedutivo estudadas, desde a antiguidade, por *Aristóteles* (384-322 a.C.).

- **Modus Ponens (MP):** de $\alpha \rightarrow \beta$ e α , conclui-se β .
- **Modus Tollens (MT):** de $\alpha \rightarrow \beta$ e $\neg\beta$, conclui-se $\neg\alpha$.
- **Silogismo Hipotético (SH):** de $\alpha \rightarrow \beta$ e $\beta \rightarrow \gamma$, conclui-se $\alpha \rightarrow \gamma$.

Dado um conjunto de fórmulas Δ , uma regra de inferência é *correta* se permite derivar apenas fórmulas que representam conseqüências lógicas de Δ e é *completa* se permite derivar todas as fórmulas que representam conseqüências lógicas de Δ . As regras de inferência clássicas são corretas e completas para todo conjunto consistente de fórmulas bem-formadas da lógica proposicional [3].

Temos a seguir uma prova da validade do argumento $\{p \rightarrow q, \neg p \rightarrow r, q \rightarrow s, \neg s\} \vdash r$, usando as regras de inferência clássicas. Nessa prova, em cada linha, há uma justificativa de como a fórmula foi obtida. Por exemplo, a justificativa Δ na linha (1) indica que a fórmula $p \rightarrow q$ é uma das premissas originais do argumento e a justificativa $MT(4, 5)$, na linha (6), indica que a fórmula $\neg p$ foi derivada das fórmulas nas linhas (4) e (5), pela aplicação de *modus tollens*.

(1)	$p \rightarrow q$	Δ
(2)	$\neg p \rightarrow r$	Δ
(3)	$q \rightarrow s$	Δ
(4)	$\neg s$	Δ
(5)	$p \rightarrow s$	$SH(1, 3)$
(6)	$\neg p$	$MT(4, 5)$
(7)	r	$MP(2, 6)$

Exercício 4 Prove usando as regras de inferência clássicas:

- $\{p \rightarrow q, \neg q, \neg p \rightarrow r\} \vdash r$
- $\{\neg p \rightarrow \neg q, q, p \rightarrow \neg r\} \vdash \neg r$
- $\{p \rightarrow q, q \rightarrow r, \neg r, \neg p \rightarrow s\} \vdash s$

□

2.4 Refutação

Embora a prova seja um mecanismo mais eficiente que a tabela-verdade, ainda é muito difícil obter algoritmos de prova que possam ser implementados eficientemente em computadores. Nesse caso, podemos usar um terceiro mecanismo para validação de argumentos, denominado refutação.

A *refutação* é um processo em que se demonstra que uma determinada hipótese contradiz um conjunto de premissas consistente [3]. Dizemos que um conjunto de fórmulas é *consistente* se e só se existe uma interpretação para seus símbolos proposicionais que torna todas as suas fórmulas verdadeiras. Caso não exista uma tal interpretação, dizemos que o conjunto de fórmulas é *inconsistente*. Formalmente, dado um conjunto de fórmulas consistente Δ , provar $\Delta \vdash \gamma$ corresponde a demonstrar que $\Delta \cup \{\neg\gamma\}$ é inconsistente. Nesse contexto, a fórmula γ é denominada *tese* e a fórmula $\neg\gamma$ é denominada *hipótese*.

Para ter uma idéia intuitiva de refutação, considere o argumento a seguir:

- Se o time joga bem, ganha o campeonato.* (P1)
Se o time não joga bem, o técnico é culpado. (P2)
Se o time ganha o campeonato, os torcedores ficam contentes. (P3)
Os torcedores não estão contentes. (P4)
Logo, o técnico é culpado.

Nesse argumento, a tese é que “o técnico é culpado”. Assim, vamos admitir como hipótese que “o técnico **não** seja culpado”. O nosso objetivo é demonstrar que essa hipótese leva a uma contradição. Se tal contradição for encontrada, como o conjunto de premissas é consistente, podemos concluir que ela foi derivada da hipótese e que, portanto, a tese é uma consequência necessária das premissas.

- (a) *O técnico não é culpado.* hipótese
 (b) *O time joga bem.* $MT(a, P2)$
 (c) *O time ganha o campeonato.* $MP(b, P1)$
 (d) *O torcedores ficam contentes.* $MP(c, P3)$
 (e) *contradição!* $confrontando (d) e P4$

Exercício 5 Usando refutação, mostre que o argumento a seguir é válido:

- *Se Ana sente dor estômago, ela fica irritada.*
- *Se Ana toma remédio para dor de cabeça, ela sente dor de estômago.*
- *Ana não está irritada.*
- *Logo, ela não tomou remédio para dor de cabeça.* □

Exercício 6 Prove usando refutação:

- $\{p \rightarrow q, \neg q, \neg p \rightarrow r\} \vdash r$
- $\{\neg p \rightarrow \neg q, q, p \rightarrow \neg r\} \vdash \neg r$
- $\{p \rightarrow q, q \rightarrow r, \neg r, \neg p \rightarrow s\} \vdash s$ □

2.5 Forma normal conjuntiva

Podemos automatizar o processo de refutação, descrevendo-o como um algoritmo computacional. Para que esse algoritmo seja mais simples e eficiente, é necessário que as fórmulas manipuladas por ele sejam convertidas em uma forma conhecida como *forma normal conjuntiva* ou FNC.

Qualquer fórmula bem-formada pode ser convertida para a forma normal conjuntiva, através dos seguintes passos:

- 1^o elimine todas as implicações: $\alpha \rightarrow \beta \equiv \neg\alpha \vee \beta$
- 2^o reduza o escopo das negações: $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ e $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$
- 3^o reduza o escopo das disjunções: $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

Como exemplo, vamos normalizar a fórmula $p \vee q \rightarrow r \wedge s$. Eliminando a implicação, obtemos $\neg(p \vee q) \vee (r \wedge s)$. Reduzindo o escopo da negação, obtemos $(\neg p \wedge \neg q) \vee (r \wedge s)$. Finalmente, reduzindo o escopo da disjunção, obtemos $((\neg p \wedge \neg q) \vee r) \wedge ((\neg p \wedge \neg q) \vee s)$. Como o terceiro passo ainda não foi concluído (veja que ainda há duas disjunções cujos escopos podem ser reduzidos), continuamos a conversão e obtemos $(\neg p \vee r) \wedge (\neg q \vee r) \wedge (\neg p \vee s) \wedge (\neg q \vee s)$, que é a forma normal conjuntiva. Eliminando as conjunções na forma normal conjuntiva, obtemos o seguinte conjunto de fórmulas normais ou cláusulas: $\{\neg p \vee r, \neg q \vee r, \neg p \vee s, \neg q \vee s\}$.

Exercício 7 Formalize as sentenças a seguir e normalize as fórmulas obtidas:

- Se não é noite e nem há lua cheia, então não há lobisomem.
- Se eu fosse rico ou famoso, não precisaria trabalhar tanto.
- Se o programa está correto, então o compilador não exhibe mensagens de erro e gera um arquivo executável.
- Se o motorista é multado, então ele passou um sinal vermelho ou excedeu o limite de velocidade. □

2.6 Inferência por resolução

A vantagem da FNC é que ela torna a forma das fórmulas mais simples e uniforme, permitindo o uso de resolução. *Resolução* é uma regra de inferência que generaliza as regras de inferência clássicas. A idéia da resolução é a seguinte: $RES(\alpha \vee \beta, \neg\beta \vee \gamma) \equiv \alpha \vee \gamma$. Além disso, definimos $RES(\alpha, \neg\alpha) \equiv \square$. Note que a resolução é equivalente às três regras de inferência clássicas:

$$\begin{array}{ll}
 MP(\alpha \rightarrow \beta, \alpha) \equiv \beta & \text{é equivalente a } RES(\neg\alpha \vee \beta, \alpha) \equiv \beta \\
 MT(\alpha \rightarrow \beta, \neg\beta) \equiv \neg\alpha & \text{é equivalente a } RES(\neg\alpha \vee \beta, \neg\beta) \equiv \neg\alpha \\
 SH(\alpha \rightarrow \beta, \beta \rightarrow \gamma) \equiv \alpha \rightarrow \gamma & \text{é equivalente a } RES(\neg\alpha \vee \beta, \beta \vee \gamma) \equiv \neg\alpha \vee \gamma
 \end{array}$$

Como exemplo, vamos usar a forma normal conjuntiva, resolução e refutação para provar a validade do argumento $\{p \rightarrow q, \neg p \rightarrow r, q \rightarrow s, \neg s\} \vdash r$:

(1)	$\neg p \vee q$	Δ
(2)	$p \vee r$	Δ
(3)	$\neg q \vee s$	Δ
(4)	$\neg s$	Δ
(5)	$\neg r$	<i>hipótese</i>
(6)	p	<i>RES(2, 5)</i>
(7)	q	<i>RES(1, 6)</i>
(8)	s	<i>RES(3, 7)</i>
(9)	\square	<i>RES(4, 8)</i>

Observe que, no processo de refutação, começamos resolvendo a hipótese com alguma cláusula em Δ . A partir daí, sempre usamos o resultado da última resolução efetuada, combinado com alguma cláusula em Δ . Se num desses passos não houver em Δ uma cláusula que possa ser utilizada pela resolução, então significa que a hipótese não produz contradição e que, portanto, a tese não é uma consequência lógica da base Δ .

Exercício 8 Usando refutação e resolução, prove os argumentos a seguir:

- *O participante vai ao paredão se o líder o indica ou os colegas o escolhem. Se o participante vai ao paredão e chora, então ele conquista o público. Se o participante conquista o público, ele não é eliminado. O líder indicou um participante e ele foi eliminado. Logo, o participante não chorou.*
- *Se o programa é bom ou passa no horário nobre, o público assiste. Se o público assiste e gosta, então a audiência é alta. Se a audiência é alta, a propaganda é cara. O programa, passa no horário nobre, mas a propaganda é barata. Logo, o público não gosta do programa.* \square

3 Lógica de predicados

Há vários tipos de argumentos que não podem ser expressos em lógica proposicional. Como exemplo, considere o seguinte argumento:

*Sócrates é homem.
 Todo homem é mortal.
 Logo, Sócrates é mortal.*

Intuitivamente, podemos ver que esse argumento é válido. No entanto, usando lógica proposicional, a forma desse argumento seria $\{p, q\} \models r$ e não haveria como mostrar que a conclusão r é uma consequência lógica das premissas p e q . Isso acontece porque a validade desse argumento depende da semântica da palavra “*todo*”, que não é considerada na lógica proposicional.

Para tratar esse tipo de argumento, a lógica de predicados estende a lógica proposicional com as noções de *predicados*, *variáveis* e *quantificadores*.

3.1 Sintaxe da lógica de predicados

Além dos símbolos da lógica proposicional, a lógica de predicados utiliza variáveis (*i.e.*, letras maiúsculas do alfabeto latino, possivelmente indexadas) e os quantificadores universal (\forall) e existencial (\exists). São *fórmulas bem-formadas* na lógica de predicados:

- todas as fórmulas bem-formadas da lógica proposicional;
- e, se ϕ é uma fórmula bem-formada e X é uma variável, $\forall X[\phi]$ e $\exists X[\phi]$.

Usando essa sintaxe, o argumento acima poderia ser representado como:

- (1) $homem(socrates)$
- (2) $\forall X[homem(X) \rightarrow mortal(X)]$
- (3) $mortal(socrates)$

3.2 Semântica da lógica de predicados

Assim como na lógica proposicional, o significado das fórmulas na lógica de predicados depende da interpretação de seus símbolos. Uma *interpretação* I na lógica de predicados consiste de:

- um conjunto $\mathcal{D} \neq \emptyset$, denominado *domínio da interpretação*;
- um mapeamento que associa cada predicado a uma relação em \mathcal{D} ;
- um mapeamento que associa cada variável ou função a um elemento em \mathcal{D} ;
- um mapeamento que associa cada constante a um elemento fixo em \mathcal{D} .

O quantificador universal (\forall) corresponde a uma conjunção e o quantificador existencial (\exists) corresponde a uma disjunção. Por exemplo, supondo $\mathcal{D} = \{a, b, c\}$, a fórmula $\forall X p(X)$ denota $p(a) \wedge p(b) \wedge p(c)$ e a fórmula $\exists X q(X)$ denota $q(a) \vee q(b) \vee q(c)$. Assim, lembrando que $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$, é fácil perceber que $\neg\forall X p(X) \equiv \exists X \neg p(X)$. De modo análogo, concluímos que $\neg\exists X p(X) \equiv \forall X \neg p(X)$.

3.3 Enunciados categóricos e tradução de sentenças

Há quatro tipos de sentenças, de especial interesse na lógica de predicados, denominadas *enunciados categóricos*:

- **Universal afirmativo:** são enunciados da forma $\forall X[p(X) \rightarrow q(X)]$ como, por exemplo, “*Todos os homens são mortais*”.
- **Universal negativo:** são enunciados da forma $\forall X[p(X) \rightarrow \neg q(X)]$ como, por exemplo, “*Nenhum homem é extra-terrestre*”.
- **Particular afirmativo:** são enunciados da forma $\exists X[p(X) \wedge q(X)]$ como, por exemplo, “*Alguns homens são cultos*”.

- **Particular negativo:** são enunciados da forma $\exists X[p(X) \wedge \neg q(X)]$ como, por exemplo, “*Alguns homens não são honestos*”.

Reconhecer o tipo de uma sentença facilita a sua tradução para a lógica de predicados. Veja outros exemplos:

- “*Toda cobra é venenosa*”: $\forall X[cobra(X) \rightarrow venenosa(X)]$
- “*Os remédios são perigosos*”: $\forall X[remedio(X) \rightarrow perigoso(X)]$
- “*Nenhuma bruxa é bela*”: $\forall X[bruxa(X) \rightarrow \neg bela(X)]$
- “*Não existe bêbado feliz*”: $\forall X[bebado(X) \rightarrow \neg feliz(X)]$
- “*Algumas pedras são preciosas*”: $\exists X[preda(X) \wedge preciosa(X)]$
- “*Existem plantas que são carnívoras*”: $\exists X[planta(X) \wedge carnívora(X)]$
- “*Alguns políticos não são honestos*”: $\exists X[politico(X) \wedge \neg honesto(X)]$
- “*Há aves que não voam*”: $\exists X[ave(X) \wedge \neg voa(X)]$

Exercício 9 Usando lógica de predicados, formalize as sentenças a seguir:

- *Tudo que sobe, desce.*
- *Nenhum leão é manso.*
- *Todo circo tem palhaço.*
- *Toda pedra preciosa é cara.*
- *Nenhum homem é infalível.*
- *Alguns escritores são cultos.*
- *Ninguém gosta de impostos.*
- *Existem impostos que não são bem empregados.* □

Equivalência entre sentenças. Há sentenças que podem ser escritas, equivalentemente, de mais de uma forma: Por exemplo, considere a sentença “*Nem tudo que brilha é ouro*”. Ora, se nem tudo que brilha é ouro, então significa que existe alguma coisa que brilha e não é ouro. Assim, a sentença “*Nem tudo que brilha é ouro*” pode ser escrita como $\neg \forall X[brilha(X) \rightarrow ouro(X)]$ ou como $\exists X[brilha(X) \wedge \neg ouro(X)]$. Para ver que as duas formas são equivalentes, analise a manipulação algébrica a seguir:

$$\begin{aligned} & \neg \forall X[brilha(X) \rightarrow ouro(X)] \\ \equiv & \neg \forall X[\neg brilha(X) \vee ouro(X)] \\ \equiv & \exists X \neg [\neg brilha(X) \vee ouro(X)] \\ \equiv & \exists X[brilha(X) \wedge \neg ouro(X)] \end{aligned}$$

Há também sentenças mais complexas como, por exemplo, “*Nem todo ator americano é famoso*”. Nesse caso, o antecedente da fórmula condicional deverá ser uma conjunção, veja: $\neg \forall X[ator(X) \wedge americano(X) \rightarrow famoso(X)]$. Uma interpretação dessa sentença seria a seguinte: ora, se nem todo ator americano é famoso, então deve existir ator americano que não é famoso. Assim, a sentença também poderia ser traduzida como $\exists X[ator(X) \wedge americano(X) \wedge \neg famoso(X)]$. A equivalência entre essas duas formas de traduzir a sentença é demonstrada a seguir:

$$\begin{aligned}
& \neg \forall X [ator(X) \wedge americano(X) \rightarrow famoso(X)] \\
\equiv & \neg \forall X [\neg(ator(X) \wedge americano(X)) \vee famoso(X)] \\
\equiv & \neg \forall X [\neg ator(X) \vee \neg americano(X) \vee famoso(X)] \\
\equiv & \exists X \neg [\neg ator(X) \vee \neg americano(X) \vee famoso(X)] \\
\equiv & \exists X [ator(X) \wedge americano(X) \wedge \neg famoso(X)]
\end{aligned}$$

Exercício 10 Verifique se as sentenças a seguir são equivalentes:

- “Nem toda estrada é perigosa” e “Algumas estradas não são perigosas”.
- “Nem todo bêbado é fumante” e “Alguns bêbados são fumantes”.

3.4 Inferência na lógica de predicados

Inferir conclusões corretas, a partir de um conjunto de premissas, é uma importante característica de todo sistema lógico. Para entendermos como a inferência pode ser realizada na lógica de predicados, vamos considerar o argumento:

$$\{homem(socrates), \forall X [homem(X) \rightarrow mortal(X)] \models mortal(socrates)$$

Normalizando² essas fórmulas, obtemos:

$$\{homem(socrates), \neg homem(X) \vee mortal(X)\} \models mortal(socrates)$$

Observe que a regra de inferência por resolução não pode ser aplicada diretamente para deduzir que Sócrates é mortal, pois as fórmulas $homem(socrates)$ e $homem(X)$ não são complementares. Entretanto, como a variável X é universal, podemos substituí-la por qualquer constante do domínio. Então, fazendo $X = socrates$, obtemos uma nova instância da fórmula $\neg homem(X) \vee mortal(X)$ e, assim, podemos inferir a conclusão desejada. Veja:

$$\begin{array}{ll}
(1) & homem(socrates) & \Delta \\
(2) & \neg homem(socrates) \vee mortal(socrates) & \Delta/X=socrates \\
\hline
(3) & mortal(socrates) & RES(1,2)
\end{array}$$

Instanciação universal. Infelizmente, o princípio de *instanciação universal*, que nos permite substituir uma variável por uma constante qualquer do seu domínio, só funciona corretamente para variáveis universais. Para entender o porquê, considere a sentença “*Todo mestre tem um discípulo*”:

$$\forall X [mestre(X) \rightarrow \exists Y [discipulo(Y, X)]]$$

Sendo X uma variável universal, podemos substituí-la por qualquer constante e a sentença obtida continuará sendo verdadeira. Particularmente, poderíamos fazer $X = xisto$ e obter a seguinte instância:

$$mestre(xisto) \rightarrow \exists Y [discipulo(Y, xisto)].$$

Note que se $mestre(xisto)$ for verdade, a semântica da sentença original forçará $\exists Y [discipulo(Y, xisto)]$ a ser verdade também e, como $\top \rightarrow \top \equiv \top$, concluímos

² Na forma normal, todas as variáveis são universais.

que a instância obtida é verdadeira. Por outro lado, se $mestre(xisto)$ for falso, independentemente do valor da fórmula $\exists Y [discipulo(Y, xisto)]$, a instância obtida também é verdadeira, pois $\perp \rightarrow \perp \equiv \top$ e $\perp \rightarrow \top \equiv \top$.

Agora, substituindo a variável existencial, obtemos a instância:

$$\forall X [mestre(X) \rightarrow [discipulo(xisto, X)],$$

que afirma é que “*Todo mestre tem um discípulo chamado Xisto*”. Evidentemente, o significado da sentença original foi perdido. Isso acontece porque o valor de Y depende do valor escolhido para X .

Skolemização. Uma forma de eliminar uma variável existencial³, sem alterar o significado da sentença original, é admitir a existência de uma função que representa o valor correto para substituir a variável existencial. Por exemplo, poderíamos substituir a variável existencial Y pela função $seguidor(X)$, veja:

$$\forall X [mestre(X) \rightarrow [discipulo(seguidor(X), X)]$$

Note que, nessa instância, o significado da sentença original é mantido; já que ela não se compromete com nenhum valor particular de Y .

No processo de *skolemização*, cada variável existencial é substituída por uma função distinta, cujos argumentos são as variáveis universais, globais⁴ à existencial em questão. Por exemplo, podemos eliminar a variável existencial em $\forall X, Y [p(X, Y) \rightarrow \exists Z \forall W [q(Z, X) \wedge q(W, Z)]]$ fazendo $Z = f(X, Y)$. Caso não haja uma variável universal, global à variável sendo skolemizada, podemos usar uma função sem argumentos. Por exemplo, podemos eliminar a variável existencial em $\exists X \forall Y [p(X) \rightarrow q(Y)]$ fazendo $X = f()$.

Daqui em diante, assumiremos que todas as variáveis são universais (já que as variáveis existenciais sempre podem ser eliminadas) e, portanto, os quantificadores ficarão implícitos.

Exercício 11 *Formalize as sentenças e skolemize as fórmulas obtidas:*

- *Todo cão é fiel ao seu dono.*
- *Existe um lugar onde todos são felizes.*

□

Unificação. Como vimos, a inferência por resolução requer que as fórmulas atômicas canceladas sejam idênticas (a menos da negação que deve ocorrer numa delas). O processo que determina que substituições são necessárias para tornar duas fórmulas atômicas sintaticamente idênticas é denominado *unificação*. Durante esse processo, uma variável pode ser substituída por uma constante, por uma variável ou por uma função. Por exemplo, podemos unificar $gosta(ana, X)$ e $gosta(Y, Z)$, fazendo $Y = ana$ e $X = Z$. Também podemos unificar $ama(deus, Y)$ e $ama(X, filho(X))$, fazendo $X = deus$ e $Y = filho(deus)$. Já as fórmulas $igual(X, X)$ e $igual(bola, bala)$ não podem ser unificadas; pois, fazendo $X =$

³ Proposta e demonstrada pelo matemático Thoralf Skolem.

⁴ Uma variável é global à outra se é declarada antes dessa outra.

bola, obtemos $igual(bola, bola)$ e $igual(bola, bala)$. Como *bola* e *bala* são constantes distintas, não existe substituição que torne as fórmulas atômicas idênticas. Também não é possível substituir uma variável por uma função que tenha essa mesma variável como parâmetro, como por exemplo, $X = f(X)$.

Para unificar duas fórmulas atômicas (sem variáveis em comum):

1. Compare as fórmulas até encontrar uma incorrespondência ou atingir o final de ambas;
2. Ao encontrar uma incorrespondência:
 - (a) se ela não envolver pelo menos uma variável, finalize com *fracasso*;
 - (b) caso contrário, substitua todas as ocorrências da variável pelo outro termo e continue a varredura (no passo 1);
3. Ao atingir o final de ambas, finalize com *sucesso*.

Exercício 12 *Unifique (se possível) as fórmulas atômicas a seguir:*

- $cor(sapato(X), branco)$ e $cor(sapato(suspeito), Y)$
- $mora(X, casa(mae(X)))$ e $mora(joana, Y)$
- $primo(X, Y)$ e $prima(A, B)$
- $ponto(X, 2, Z)$ e $ponto(1, W)$
- $p(f(Y), Y, X)$ e $p(X, f(a), f(Z))$

□

4 Raciocínio automatizado

Um dos principais algoritmos para raciocínio automatizado é denominado SLD-resolução [3]. Trata-se de um método computacional de refutação que apresenta as seguintes características:

1. Restringe-se à uma classe de fórmulas denominada *cláusulas de Horn*.
2. Emprega resolução e unificação como regra de inferência.
3. Adota uma estratégia de busca em profundidade para controlar as inferências.
4. Introduce o conceito de *predicado computável*.
5. Introduce o conceito de *negação por falha finita*.

4.1 Cláusulas de Horn

Cláusulas de Horn [1] são fórmulas da forma $\phi \leftarrow \phi_1, \dots, \phi_n$, sendo $n \geq 0$, onde o literal ϕ é uma conclusão e os literais ϕ_i são condições. Se $n = 0$, a cláusula é denominada *fato*; se $n > 0$, a cláusula é denominada *regra*. Uma fórmula da forma $\leftarrow \phi_1, \phi_2, \dots, \phi_n$ é denominada *consulta*. A cláusula \leftarrow é denominada *vazia* e denota uma contradição.

4.2 Inferência com cláusulas de Horn

A inferências são realizadas sempre entre uma consulta e um fato ou entre uma consulta e uma regra, sendo que o resultado da inferência, denominado *resolvente*, é sempre uma consulta ou a cláusula vazia.

- **1^o caso:** se o fato $\alpha_0 \leftarrow$ unifica-se com o primeiro literal da consulta $\leftarrow \beta_1, \beta_2, \dots, \beta_n$, sob o conjunto de substituições σ , então a resolvente será a consulta $\leftarrow \beta'_2, \dots, \beta'_n$, onde β'_i é uma instância de β_i , sob a substituição σ ;
- **2^o caso:** se a conclusão da regra $\alpha_0 \leftarrow \alpha_1, \dots, \alpha_m$ unifica-se com o primeiro literal da consulta $\leftarrow \beta_1, \beta_2, \dots, \beta_n$, sob o conjunto de substituições σ , então a resolvente será a consulta $\leftarrow \alpha'_1, \dots, \alpha'_m, \beta'_2, \dots, \beta'_n$, onde α'_i é uma instância de α_i e β'_i é uma instância de β_i , sob a substituição σ .

A função *unifica*(α, β, γ) tenta unificar a cláusula α com a consulta β , resultando na nova consulta γ (que pode ser a cláusula vazia). Caso a unificação seja bem sucedida, essa função devolve **verdade**; caso contrário, devolve **falso**.

4.3 Algoritmo de busca SLD-RESOLUÇÃO

O algoritmo SLD-RESOLUÇÃO [1,2] controla a aplicação da regra de inferência, realizando uma busca em profundidade. Ao derivar a cláusula vazia, o algoritmo sinaliza **sucesso** e apresenta como solução a composição das substituições efetuadas no caminho percorrido até a cláusula vazia. Ao atingir um ponto onde a consulta não pode ser unificada com nenhuma cláusula, o algoritmo sinaliza **fracasso** e tenta retroceder na busca.

Sejam Δ um conjunto de cláusulas de Horn (programa lógico) e β uma consulta. O algoritmo a seguir responde à consulta com base em Δ :

```

SLD-RESOLUÇÃO( $\Delta, \beta$ )
1 se  $\beta$  é a cláusula vazia então devolva sucesso
2 para cada cláusula  $\alpha \in \Delta$  (de cima para baixo) faça
3   se unifica( $\alpha, \beta, \gamma$ ) e SLD-RESOLUÇÃO( $\Delta, \gamma$ ) = sucesso então
4     exiba a composição das substituições efetuadas
5 devolva fracasso

```

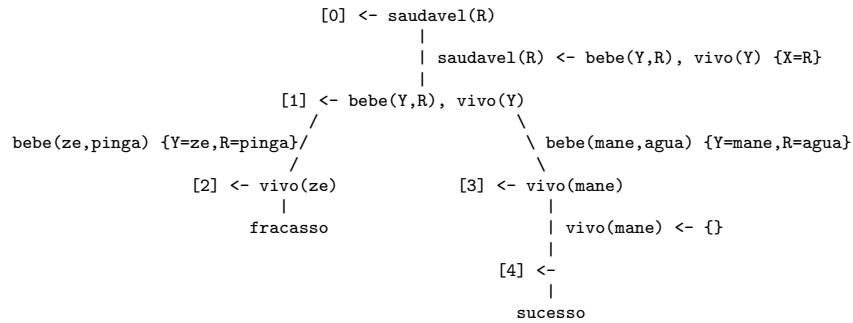
Exemplo 1. Considere Δ contendo as seguintes cláusulas:

```

bebe(ze, pinga)  $\leftarrow$ 
bebe(mane, agua)  $\leftarrow$ 
vivo(mane)  $\leftarrow$ 
saudavel(X)  $\leftarrow$  bebe(Y, X), vivo(Y)

```

Para descobrir o que é saudável, de acordo com Δ , podemos entrar com a consulta \leftarrow *saudavel*(*R*). Vejamos como o algoritmo responde a essa consulta:



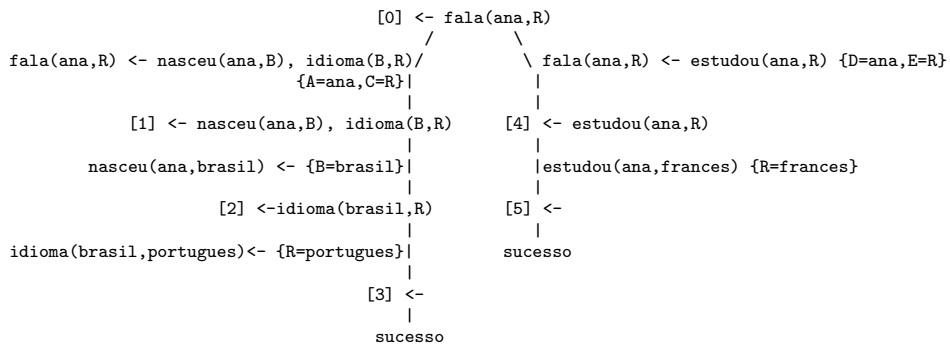
Nessa *árvore de refutação*, as resolventes estão numeradas na ordem em que são geradas pelo algoritmo SLD-RESOLUÇÃO. O primeiro caminho percorrido termina com **fracasso**, então o algoritmo retrocede e tenta outro caminho. Ao encontrar a cláusula vazia, ele sinaliza **sucesso** e exibe a resposta derivada das substituições efetuadas no caminho percorrido até a cláusula vazia. No caso do nosso exemplo, a resposta encontrada foi $R = \textit{agua}$.

Exemplo 2. Vejamos um outro exemplo:

```

nasceu(ana, brasil) <-
nasceu(yves, france) <-
idioma(brasil, portugues) <-
idioma(franca, frances) <-
estudou(ana, frances) <-
fala(A, C) <- nasceu(A, B), idioma(B, C)
fala(D, E) <- estudou(D, E)
    
```

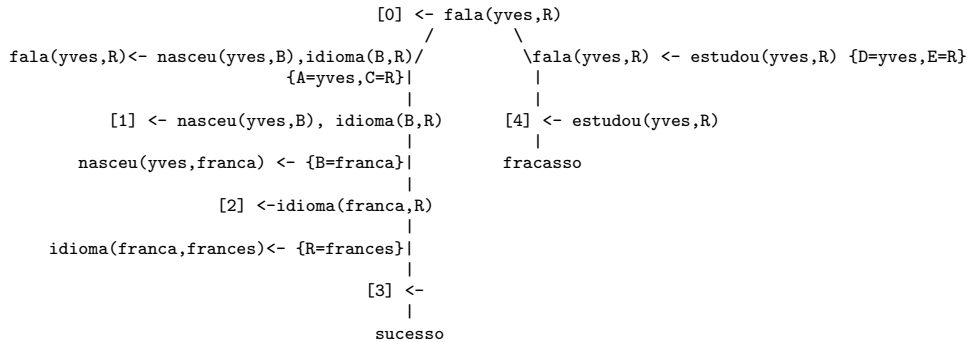
Para descobrir que idiomas Ana fala, podemos usar a consulta $\leftarrow \textit{fala}(\textit{ana}, R)$:



Para essa consulta, o algoritmo é bem sucedido nos dois caminhos percorridos e apresenta $R = \textit{portugues}$ e $R = \textit{frances}$ como respostas. Veja como isso faz sentido: declaramos no programa lógico que uma pessoa fala um idioma se ela

nasce num país onde se fala esse idioma ou se ela estuda esse idioma. Então, como Ana nasceu no Brasil, o algoritmo conclui que Ana fala português. Além disso, como Ana estudou francês, o algoritmo conclui que Ana também fala francês.

Vejamos, agora, as respostas para a consulta $\leftarrow fala(yves, R)$:



Bem, como Yves não estudou nenhum idioma, era de se esperar que ele falasse apenas francês, que é a sua língua nativa. Portanto, o algoritmo encontrará apenas uma resposta: $R = frances$.

Exercício 13 Considere o programa lógico a seguir:

```

gosta(ary, eva) <-
gosta(ivo, ana) <-
gosta(ivo, eva) <-
gosta(eva, ary) <-
gosta(ana, ary) <-
namora(A, B) <- gosta(A, B), gosta(B, A)
  
```

Mostre como o algoritmo SLD-RESOLUÇÃO responde às seguintes consultas:

- Ivo gosta de quem?
- Quem gosta de Ary?
- Ivo namora com Eva?
- Ary namora com quem?
- Eva namora com Ary?

□

Exercício 14 Considere o programa lógico a seguir:

```

pai(adao, cain) <-
pai(adao, abel) <-
pai(adao, seth) <-
pai(seth, enos) <-
avo(X, Z) <- pai(X, Y), pai(Y, Z)
  
```

Mostre como o algoritmo SLD-RESOLUÇÃO responde às seguintes consultas:

- Quem é pai de Abel?
- Adão é pai de quem?
- Quem é avô de Enos?
- Seth é avô de alguém? □

Exercício 15 Mostre que, com base no programa a seguir, o algoritmo SLD-RESOLUÇÃO encontra três respostas para consulta $\leftarrow \text{irmao}(\text{cain}, R)$.

```

pai(adao, cain) ←
pai(adao, abel) ←
pai(adao, seth) ←
irmao(X, Y) ← pai(Z, X), pai(Z, Y) □

```

4.4 Predicados computáveis

Predicados computáveis [5] são predicados que podem ser avaliados diretamente pelo algoritmo de refutação, sem que haja necessidade de serem definidos no programa lógico. Por exemplo, um dos predicados computáveis mais importantes é a desigualdade entre termos (\neq). Quando um predicado computável é encontrado durante uma refutação, o algoritmo SLD-RESOLUÇÃO sinaliza fracasso apenas se a avaliação desse predicado resultar em falso.

```

SLD-RESOLUÇÃO'(Δ, β)
1 se β é a cláusula vazia então devolva sucesso
2 seja β1 o primeiro literal da consulta β
3 se β1 é computável então
4   se β1 é falso então devolva fracasso
5   senão se SLD-RESOLUÇÃO'(Δ, ← β2, ..., βn) = sucesso então
6     exiba a composição das substituições efetuadas
7   senão
8     para cada cláusula α ∈ Δ (de cima para baixo) faça
9       se unifica(α, β, γ) e SLD-RESOLUÇÃO'(Δ, γ) = sucesso então
10        exiba a composição das substituições efetuadas
11 devolva fracasso

```

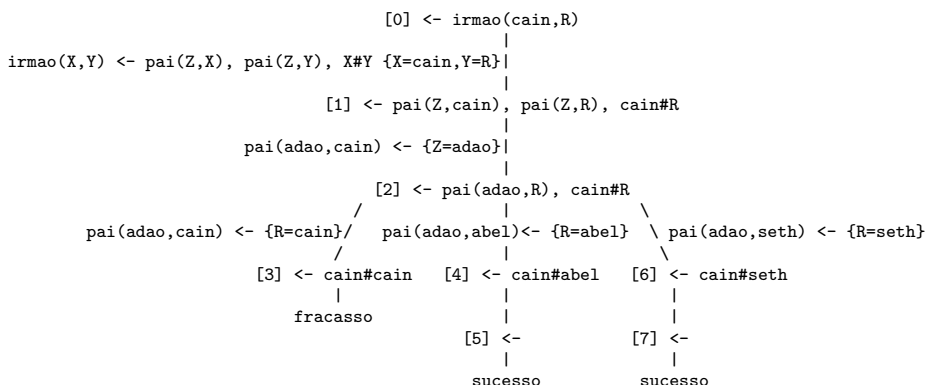
Exemplo 3. Considere o programa lógico a seguir (que usa o predicado \neq):

```

pai(adao, cain) ←
pai(adao, abel) ←
pai(adao, seth) ←
irmao(X, Y) ← pai(Z, X), pai(Z, Y), X ≠ Y □

```

Para responder à consulta $\leftarrow \text{irmao}(\text{cain}, R)$, o algoritmo modificado para tratar predicados computáveis procede do seguinte modo:



Exercício 16 Com base no programa lógico a seguir, mostre como o algoritmo SLD-RESOLUÇÃO' responde à consulta $\leftarrow \text{infiel}(R)$.

```

gosta(ary, eva) <-
gosta(ivo, eva) <-
gosta(ary, bia) <-
gosta(eva, ary) <-
namora(A, B) <- gosta(A, B), gosta(B, A)
infiel(C) <- namora(C, D), gosta(C, E), D ≠ E

```

□

4.5 Negação por falha finita

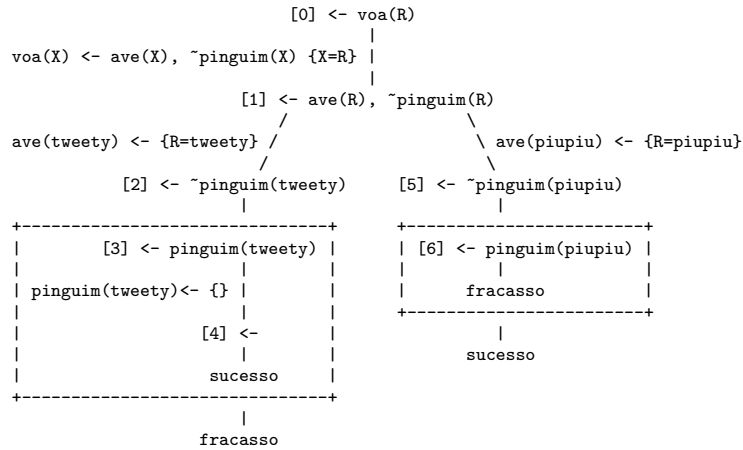
Para tratar corretamente literais negativos, uma última modificação é necessária no algoritmo de refutação: agora, quando o algoritmo encontra um literal negado, ele dispara uma refutação do literal complementar. Se essa refutação sucede, o algoritmo sinaliza **fracasso**; caso contrário, o algoritmo prossegue com a busca.

SLD-RESOLUÇÃO''(Δ, β)

- 1 se β é a cláusula vazia então devolva sucesso
- 2 seja β_1 o primeiro literal da consulta β
- 3 se β_1 é computável então
- 4 se β_1 é falso então devolva fracasso
- 5 senão se SLD-RESOLUÇÃO''($\Delta, \leftarrow \beta_2, \dots, \beta_n$) = sucesso então
- 6 exiba a composição das substituições efetuadas
- 7 senão se $\beta_1 = \neg \phi$ então
- 8 se SLD-RESOLUÇÃO''($\Delta, \leftarrow \phi$) = sucesso então devolva fracasso
- 9 senão se SLD-RESOLUÇÃO''($\Delta, \leftarrow \beta_2, \dots, \beta_n$) = sucesso então
- 10 exiba a composição das substituições efetuadas
- 11 senão
- 12 para cada cláusula $\alpha \in \Delta$ (de cima para baixo) faça
- 13 se $\text{unifica}(\alpha, \beta, \gamma)$ e SLD-RESOLUÇÃO''(Δ, γ) = sucesso então
- 14 exiba a composição das substituições efetuadas
- 15 devolva fracasso

Exemplo 4. Vejamos como a negação por falha finita funciona ao responder à consulta $\leftarrow voa(R)$, com base no programa a seguir:

```
ave(tweety) ←
ave(piupiu) ←
pinguim(tweety) ←
voa(X) ← ave(X), ¬pinguim(X)
```



Exercício 17 Um fato interessante é que o predicado computável \neq pode ser implementado através de negação por falha, conforme especificado a seguir:

```
igual(X, X) ←
diferente(X, Y) ← ¬igual(X, Y)
```

Com base nessa especificação, mostre como o algoritmo SLD-RESOLUÇÃO'' responde às consultas $\leftarrow diferente(a, b)$ e $\leftarrow diferente(a, a)$. □

5 A linguagem PROLOG

PROLOG (**programming in logic**) [7] é uma linguagem de programação declarativa, baseada em cláusulas de Horn, que tem o algoritmo SLD-RESOLUÇÃO'' embutido. Enquanto numa linguagem de programação imperativa um programa consiste numa descrição detalhada de como um determinado problema deve ser solucionado, em PROLOG, um programa é apenas um conjunto de sentenças que declaram o conhecimento que temos a cerca de um determinado contexto. Uma vez que essas sentenças sejam fornecidas ao sistema PROLOG, o usuário pode consultá-lo, fazendo perguntas cujas respostas serão deduzidas, automaticamente, a partir do conhecimento que foi declarado pelas sentenças.

5.1 Fatos, regras e consultas

Para introduzir os elementos básicos da linguagem PROLOG, vamos usar como exemplo um famoso argumento de lógica clássica:

Sócrates é um homem.
Todo homem é mortal.

Logo, Sócrates é mortal.

Nesse argumento, as duas primeiras sentenças são *premissas*, enquanto a última delas é uma *conclusão*. Claramente, essa conclusão pode ser deduzida das premissas. Então, como o sistema PROLOG implementa um algoritmo de raciocínio dedutivo, se fornecermos a ele essas premissas e perguntarmos se Sócrates é mortal ele deverá responder que sim. Para tanto, a primeira coisa que temos a fazer é escrever as premissas numa forma que elas possam ser entendidas pelo sistema. Escrevendo as premissas em PROLOG, obtemos o seguinte programa:

```
homem(sócrates).
mortal(X) :- homem(X).
```

Observe que, com relação à notação lógica usada nas cláusulas de Horn, as diferenças sintáticas em PROLOG são pequenas: primeiro, todas as cláusulas devem finalizar com ponto; segundo, o operador \leftarrow é substituído pelo operador $:-$; e, terceiro, nos fatos, o operador \leftarrow é omitido.

Nesse programa, a cláusula `homem(sócrates)`, que deve ser lida como “*Sócrates é um homem*”, declara um *fato*; enquanto a cláusula `mortal(X) :- homem(X)`, que deve ser lida como “*X é mortal se X é um homem*”, estabelece uma *regra*. Fatos e regras são denominados *cláusulas* e um *programa lógico* nada mais é do que um conjunto de cláusulas.

Para executar o programa, entramos no SWI-PROLOG⁵ e, ao sinal de prontidão do sistema, digitamos `emacs('filósofos.pl')`, seguido de **ponto** e **enter** (veja a figura 1). Isso fará com que o editor EMACS seja aberto para a criação do arquivo⁶ `filósofos.pl`. Em seguida, digitamos as cláusulas que compõem o programa e compilamos (veja a figura 2).

Se nenhum erro de digitação tiver sido cometido, o sistema informará que o programa foi corretamente compilado e que está pronto para ser consultado pelo usuário.

```
filósofos.pl compiled, 0.01 sec, 588 bytes.
yes
?-
```

⁵ Download disponível em <http://www.swi-prolog.org>.

⁶ Apenas arquivos com extensão `.pl` são reconhecidos como programas em PROLOG.



Figura 1. Tela de consulta do interpretador SWI-PROLOG

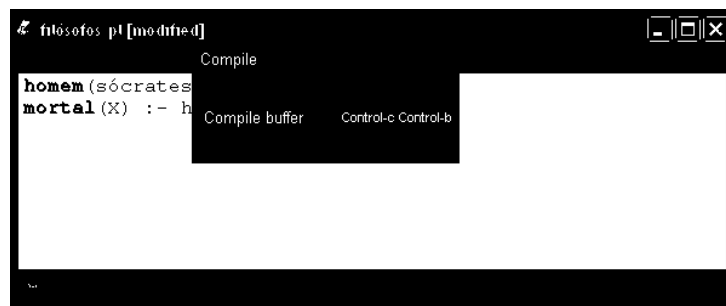


Figura 2. Tela do editor de textos EMACS

Para testar o sistema, podemos começar com a seguinte consulta:

```
?- homem(sócrates).
```

ou seja, “*Sócrates é um homem?*”. A essa consulta o sistema responderá “*yes*”, já que esse fato foi explicitamente estabelecido pela primeira cláusula do programa. Outra consulta que podemos fazer é “*Sócrates é mortal?*”:

```
?- mortal(sócrates).
```

Dessa vez, embora o fato `mortal(sócrates)` não tenha sido explicitamente estabelecido, o sistema pode deduzi-lo a partir das cláusulas do programa `filósofos.pl`. Sendo assim, a resposta para essa segunda consulta também será “*yes*”.

5.2 A hipótese do mundo fechado e a negação por falha finita

Conforme vimos, o sistema PROLOG é capaz de responder positivamente a respeito de informações que lhe comunicamos explicitamente, através de *fatos*, ou

implicitamente, através de *regras*. Mas o que acontece se lhe consultarmos sobre algo que não lhe foi comunicado? Por exemplo, ainda considerando o nosso programa, poderíamos fazer a seguinte consulta:

```
?- mortal(platão).
```

ou seja, “*Platão é mortal?*”. A essa pergunta o sistema responderá “*no*”. Entretanto, essa resposta não significa que Platão seja imortal, mas apenas que o sistema não dispõe de conhecimento suficiente para afirmar que Platão é mortal.

Esse tipo de resposta negativa é baseada na *hipótese do mundo fechado* [4], segundo a qual *o sistema pode supor que conhece todos os fatos verdadeiros a respeito do mundo*. Assim, de acordo com essa hipótese, se um fato não foi comunicado ao sistema, seja explícita ou implicitamente, o sistema pode assumir que esse fato é falso.

Note que, devido à implementação da hipótese do mundo fechado, o PROLOG é um sistema lógico não-monotônico, *i.e.* a adição de novas premissas pode descartar conclusões anteriormente obtidas. Por exemplo, se adicionarmos ao programa `filósofos.pl` o fato `homem(platão)`, e repetirmos a consulta `?- mortal(platão)`, o sistema responderá “*yes*”. Ou seja, a resposta do sistema a uma consulta depende do conhecimento que ele tem a respeito do contexto de discurso: se esse conhecimento muda, suas respostas também podem mudar.

A negação em PROLOG, conhecida como *negação por falha*⁷, é também implementada com base na hipótese do mundo fechado. Quando o sistema tem que responder a respeito de um fato negativo, primeiro ele verifica se esse fato é verdadeiro. Caso o fato seja verdadeiro, ele responde “*no*”; senão, ele responde “*yes*”. Por exemplo, à consulta:

```
?- not( mortal(sócrates) ).
```

o sistema responderá “*no*”, já que o fato `mortal(sócrates)` é verdadeiro. Por outro lado, à consulta:

```
?- not( mortal(zeus) ).
```

o sistema responderá “*yes*”, já que o fato `homem(zeus)` não foi comunicado ao sistema e, portanto, de acordo com a hipótese do mundo fechado, deve ser assumido como sendo falso.

5.3 Consultas com variáveis

Veremos agora que o PROLOG é capaz de responder mais do que simplesmente “*yes*” ou “*no*”. Por exemplo, suponha que desejamos saber *quem* (`x`)⁸ é mortal:

⁷ A negação de um fato é considerada verdadeira se o sistema *falha* ao tentar provar que esse fato é verdadeiro.

⁸ No PROLOG, todo identificador iniciando com maiúscula é considerado uma variável.

```
?- mortal(X).
```

Nesse caso, não basta que o sistema responda “yes”. Será preciso que ele informe *quem* é mortal, ou seja, ele deverá encontrar um valor apropriado para a variável X , que será a resposta à nossa pergunta. Portanto, ele responderá $X = \text{sócrates}$.

Caso haja mais de uma resposta possível, o sistema exibirá a primeira delas e ficará aguardando instruções: se digitarmos *ponto-e-virgula*, ele tentará encontrar uma resposta alternativa; se digitarmos *enter*, ele encerrará a consulta.

Por exemplo, supondo que o fato `homem(platão)` tenha sido incluído no programa `filósofos.pl`, o sistema fornecerá as seguintes respostas:

```
?- mortal(X).
X = sócrates ;
X = platão
yes
```

Exercício 18 Codifique as cláusulas a seguir em PROLOG e consulte o sistema para descobrir o que é saudável.

```
bebe(ze, pinga) ←
bebe(mane, agua) ←
vivo(mane) ←
saudavel(X) ← bebe(Y, X), vivo(Y) □
```

Exercício 19 Codifique as cláusulas a seguir em PROLOG e consulte o sistema para descobrir que idiomas a Ana fala e que idiomas o Yves fala.

```
nasceu(ana, brasil) ←
nasceu(yves, france) ←
idioma(brasil, portugues) ←
idioma(franca, frances) ←
idioma(inglaterra, ingles) ←
estudou(ana, frances) ←
estudou(ana, ingles) ←
estudou(yves, ingles) ←
fala(A, C) ← nasceu(A, B), idioma(B, C)
fala(D, E) ← estudou(D, E) □
```

Exercício 20 Codifique as cláusulas a seguir em PROLOG e consulte o sistema para descobrir quem é irmão de Cain (no PROLOG o predicado de desigualdade é escrito como `\=`).

```
pai(adao, cain) ←
pai(adao, abel) ←
pai(adao, seth) ←
irmao(X, Y) ← pai(Z, X), pai(Z, Y), X ≠ Y □
```

Exercício 21 Codifique as cláusulas a seguir em PROLOG e consulte o sistema para descobrir quem namora com quem e quem é infiel.

```

gosta(ary, eva) ←
gosta(ary, bia) ←
gosta(ivo, ana) ←
gosta(ivo, eva) ←
gosta(eva, ary) ←
gosta(ana, ary) ←
namora(A, B) ← gosta(A, B), gosta(B, A)
infiel(C) ← namora(C, D), gosta(C, E), D ≠ E

```

Exercício 22 Imagine um contexto definido pelos seguintes predicados:

- *mora(Pessoa, Bairro)* : relaciona uma pessoa ao bairro em que ela mora
- *pertence(Bairro, Zona)* : relaciona um bairro à zona à qual ele pertence
- *amigo(Pessoa, Pessoa)* : relaciona duas pessoas que são amigas
- *tem_carro(Pessoa)* : discrimina as pessoas que têm carro

Crie uma coleção de fatos a respeito desses predicados (invente os dados) e defina uma regra estabelecendo em que uma pessoa pode dar carona a outra se essa pessoa tem carro e ambas moram em bairros que ficam na mesma zona. Em seguida, faça consultas ao sistema PROLOG e verifique se as respostas obtidas são coerentes com os dados declarados. □

Referências

1. AMBLE, T. *Logic Programming and Knowledge Engineering*, Addison-Wesley, 1987.
2. BRATKO, I. *Prolog - Programming for Artificial Intelligence*, Addison-Wesley, 1990.
3. GENESERETH, M. R. AND NILSSON, N. J. *Logical Foundations of Artificial Intelligence*, Morgan Kaufmann Publishers, 1988.
4. REITER, R. *On Closed World Data Bases*, In H. Gallaire and J. Minker, *Logic and Data Bases*, pages 55-76, Plenum Press, NY, 1978.
5. RICH, E. AND KNIGHT, K. *Inteligência Artificial*, 2ª ed., Makron Books, 1995.
6. RUSSELL, S. AND NORVIG, P. *Artificial Intelligence - a modern approach*, Prentice-Hall, 1995.
7. STERLING, L. AND SHAPIRO, E. *The Art of Prolog*, MIT Press, 1986.