Raciocínio Automatizado

Prof. Dr. Silvio do Lago Pereira

Departamento de Tecnologia da Informação

Faculdade de Tecnologia de São Paulo

Introdução

Raciocínio automatizado

simula raciocínio lógico por meio de processos computacionais

SLD-refutação é um procedimento para raciocínio automatizado que apresenta as seguintes características:

- restringe-se à uma classe de fórmulas denominadas *cláusulas de Horn*
- usa um mecanismo de prova por *refutação*, que combina *unificação* e *resolução*
- usa uma estratégia de *busca em profundidade* para controlar as inferências
- introduz o conceito de *predicados computáveis* (ou predefinidos no sistema)
- introduz o conceito de *negação por falha finita*

Inferência com cláusulas de Horn

Cláusulas de Horn

São fórmulas da forma

$\varphi \leftarrow \varphi_1, ..., \varphi_n$

para $n \geq 0$, onde φ é uma conclusão e $\varphi_1, ..., \varphi_n$ são premissas (condições)

Tipos de cláusulas:

- Fato.....: $\varphi \leftarrow$
- Regra.....: $\varphi \leftarrow \varphi_1, ..., \varphi_n$
- Consulta.....: $\leftarrow \varphi_1, ..., \varphi_n$
- Contradição.....: $\leftarrow$

Um programa lógico é composto apenas por fatos e regras!

Inferência com cláusulas de Horn

Inferências com cláusulas de Horn são efetuadas sempre entre:

- **um fato e uma consulta**

$$\frac{\alpha_0 \leftarrow \quad \leftarrow \beta_1, \beta_2, \ldots, \beta_n}{\leftarrow \beta_2', \ldots, \beta_n'}$$

a unificação de α_0 e β_1 tem efeito colateral no valor dos demais literais (i.e., na nova consulta)

- **uma regra e uma consulta**

$$\frac{\alpha_0 \leftarrow \alpha_1, \alpha_2, \ldots, \alpha_m \quad \leftarrow \beta_1, \beta_2, \ldots, \beta_n}{\leftarrow \alpha_1', \alpha_2', \ldots, \alpha_m', \beta_2', \ldots, \beta_n'}$$

O resultado de uma inferência é uma nova consulta ou uma contradição!

Inferência com cláusulas de Horn

Exemplo 1 – inferência entre fato e consulta

pai(adao,abel) ←

← pai(adao,Y), pai(Y,Z)

← pai(abel,Z) {Y=abel}

Exemplo 2 – inferência entre regra e consulta

avo(X,Z) ← pai(X,Y), pai(Y,Z)

← avo(adao,A), pai(A,B)

← pai(adao,Y), pai(Y,A), pai(A,B) {X=adao, Z=A}

SLD-refutação

Programa lógico 1 – O que é saudável?

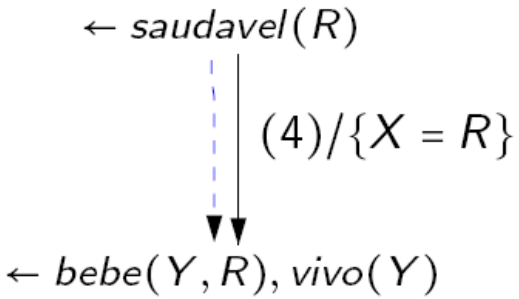
```
(1) bebe(ze,pinga) ←
(2) bebe(mane,agua) ←
(3) vivo(mane) ←
(4) saudavel(X) ← bebe(Y,X), vivo(Y)
```

$$\leftarrow saudavel(R)$$

SLD-refutação

Programa lógico 1 – O que é saudável?

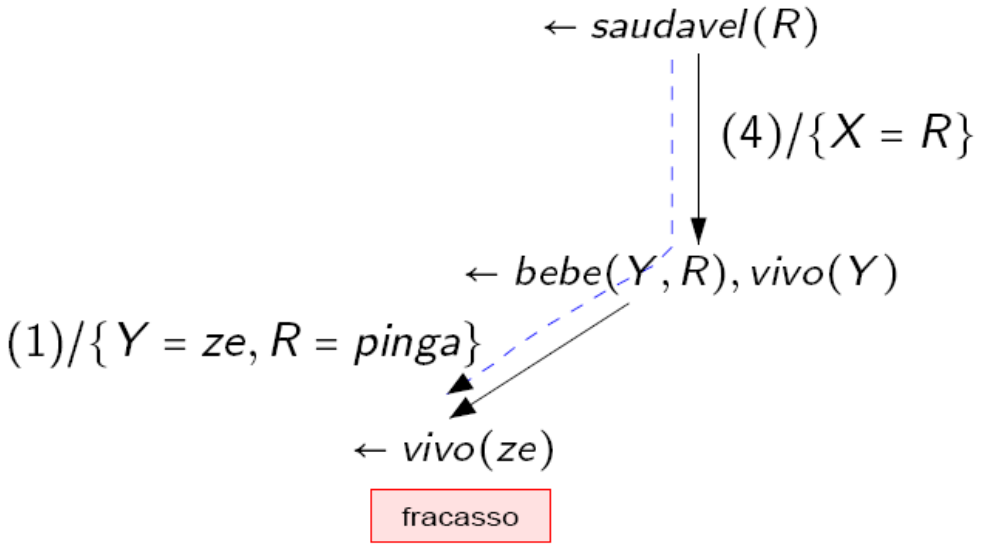
```
(1) bebe(ze,pinga) ←
(2) bebe(mane,agua) ←
(3) vivo(mane) ←
(4) saudavel(X) ← bebe(Y,X), vivo(Y)
```



SLD-refutação

Programa lógico 1 – O que é saudável?

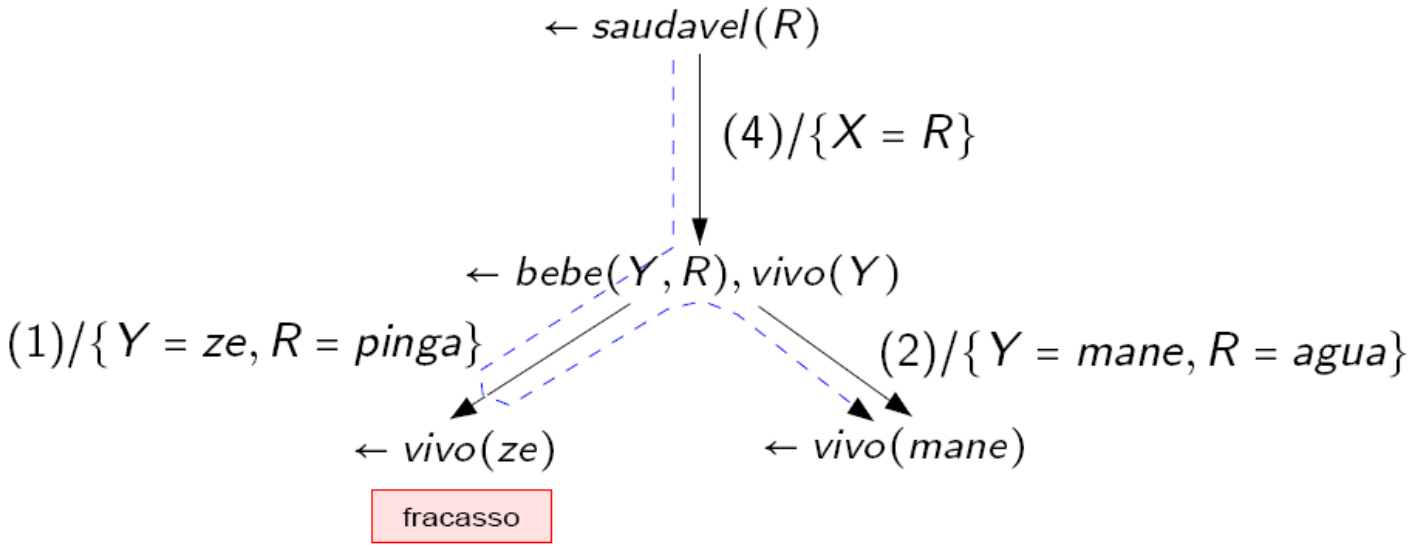
```
(1) bebe(ze,pinga) ←
(2) bebe(mane,agua) ←
(3) vivo(mane) ←
(4) saudavel(X) ← bebe(Y,X), vivo(Y)
```



SLD-refutação

Programa lógico 1 – O que é saudável?

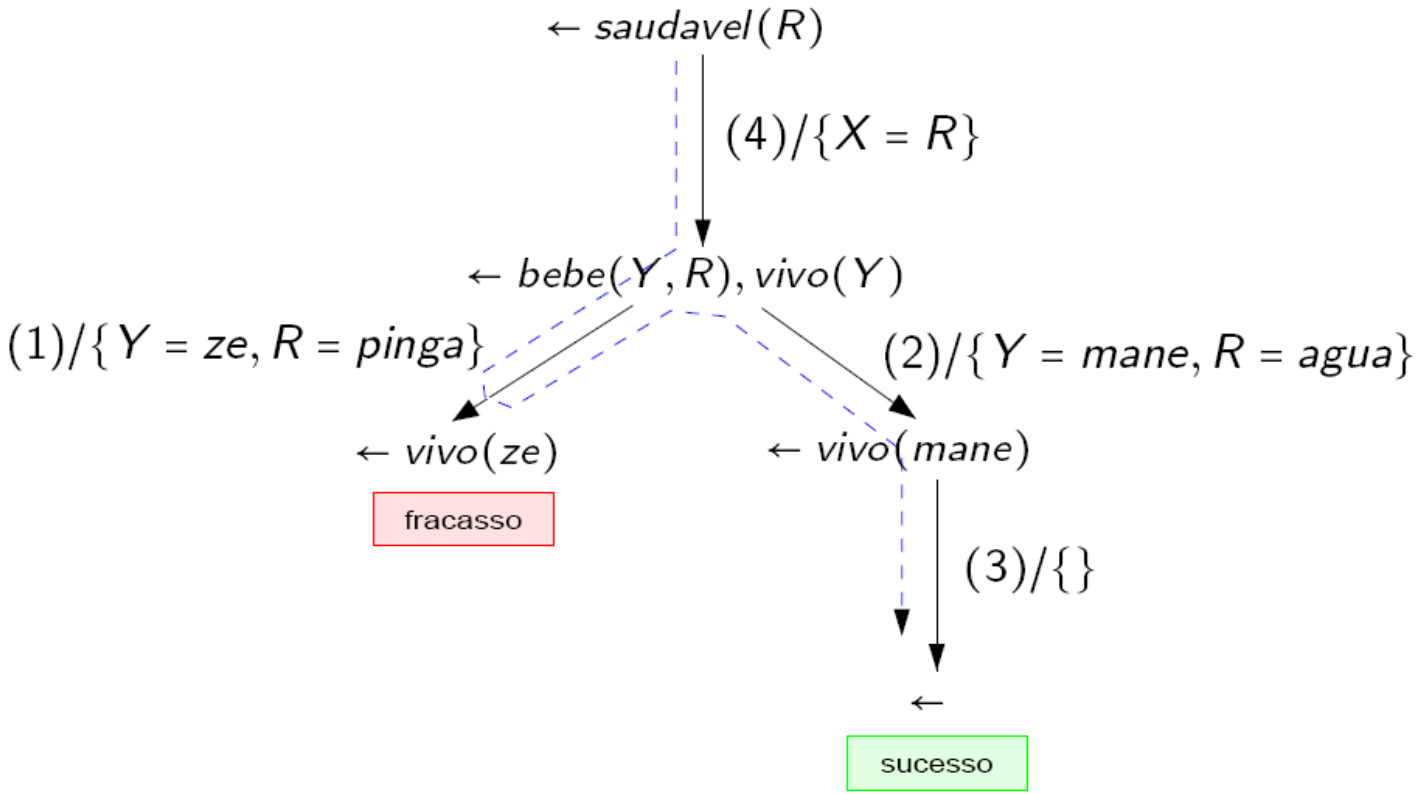
```
(1) bebe(ze,pinga) ←
(2) bebe(mane,agua) ←
(3) vivo(mane) ←
(4) saudavel(X) ← bebe(Y,X), vivo(Y)
```



SLD-refutação

Programa lógico 1 – O que é saudável?

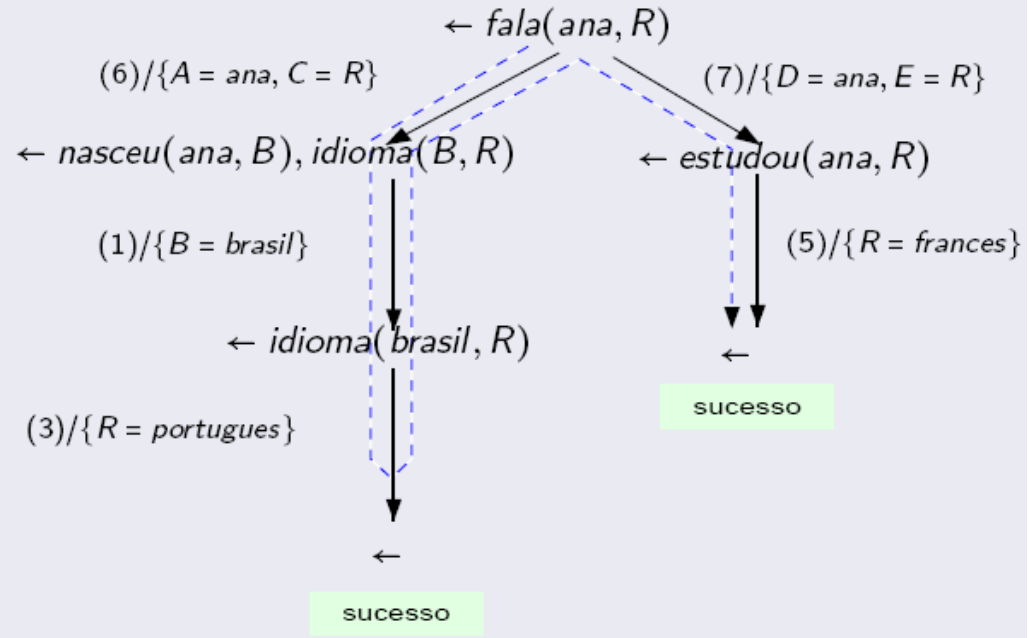
```
(1) bebe(ze,pinga) ←
(2) bebe(mane,agua) ←
(3) vivo(mane) ←
(4) saudavel(X) ← bebe(Y,X), vivo(Y)
```



SLD-refutação

Programa lógico 2 – Ana fala que idioma?

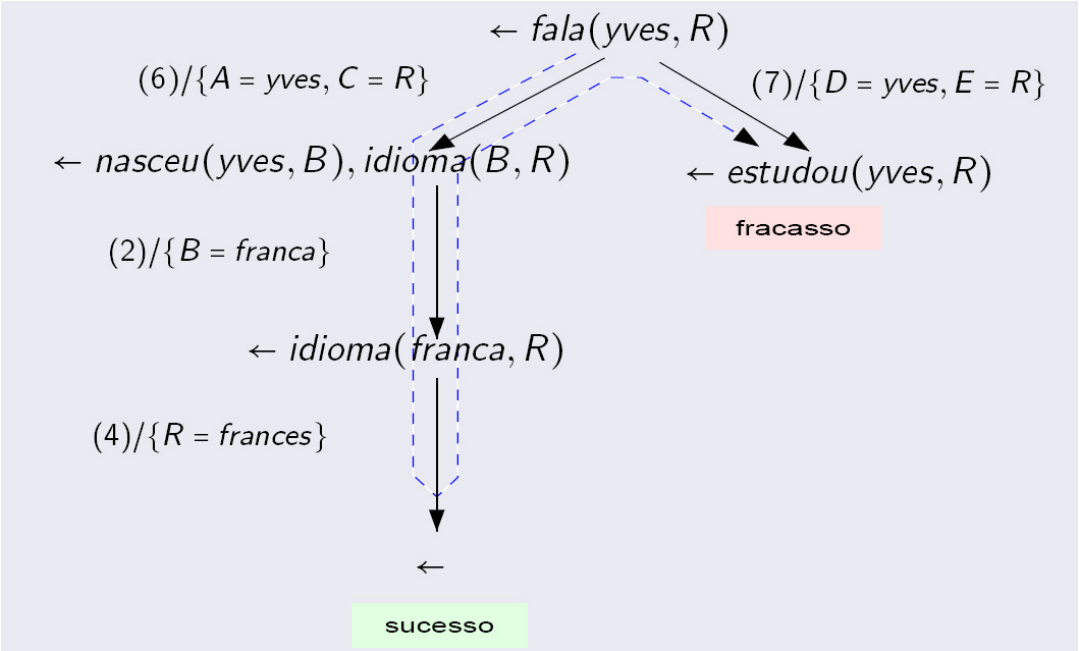
```
(1) nasceu(ana,brasil) ←
(2) nasceu(yves,franca) ←
(3) idioma(brasil,portugues) ←
(4) idioma(franca,frances) ←
(5) estudou(ana,frances) ←
(6) fala(A,C) ← nasceu(A,B), idioma(B,C)
(7) fala(D,E) ← estudou(D,E)
```



SLD-refutação

Programa lógico 2 – Yves fala que idioma?

```
(1) nasceu(ana,brasil) ←
(2) nasceu(yves,franca) ←
(3) idioma(brasil,portugues) ←
(4) idioma(franca,frances) ←
(5) estudou(ana,frances) ←
(6) fala(A,C) ← nasceu(A,B), idioma(B,C)
(7) fala(D,E) ← estudou(D,E)
```



SLD-refutação

Exercício 1

Em Prolog, o operador '←' é omitido nas cláusulas do tipo fato e substituído por '**:-**' nas cláusulas do tipo regra. Ademais, toda cláusula deve ser finalizada com '**.**'. Usando esta convenção, codifique o programa a seguir em Prolog e faça as seguintes consultas:

- *Eva namora com Ary?*
- *Ivo namora com Ana?*
- *Ary namora com quem?*

Programa lógico 3

```
(1) gosta(ary,eva) ←
(2) gosta(ivo,ana) ←
(3) gosta(ivo,eva) ←
(4) gosta(eva,ary) ←
(5) gosta(ana,ary) ←
(6) namora(A,B) ← gosta(A,B), gosta(B,A)
```

SLD-refutação

Exercício 2

Em Prolog, o predicado predefinido `trace/0` permite rastrear o raciocínio feito pelo motor de inferência do sistema, ao responder a uma consulta.

Usando este predicado para rastrear as consultas a seguir e desenhe a árvore de refutação correspondente:

- `?- namora(eva,ary).`
- `?- namora(ivo,ana).`
- `?- namora(ary,Q).`

SLD-refutação

Exercício 3

Codifique o programa a seguir em Prolog e rastreie o raciocínio do sistema ao responder às seguintes consultas:

- *Quem é avô de Enos?*
- *Seth é avô de quem?*
- *Caim é irmão de quem?*

Programa lógico 4

```
(1) pai(adão,caim) ←
(2) pai(adão,abel) ←
(3) pai(adão,seth) ←
(4) pai(seth,enos) ←
(5) avô(A,C) ← pai(A,B), pai(B,C)
(6) irmão(D,E) ← pai(F,D), pai(F,E)
```

Predicados computáveis

Predicado computável

é um predicado avaliado diretamente pelo procedimento de refutação, sem que este tenha que estar definido no programa lógico.

Exemplos:

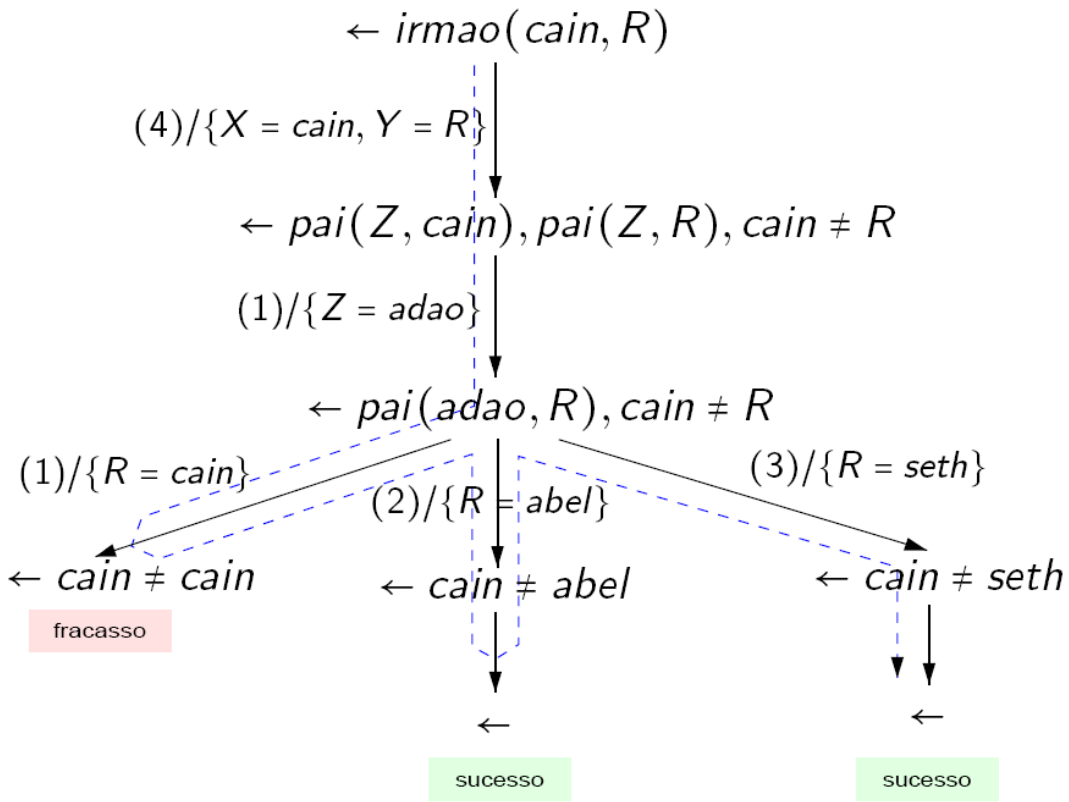
- operadores aritméticos: +, -, *, /
- operadores relacionais: =, ≠, <, ≤, >, ≥

A SLD-refutação sinaliza fracasso se um predicado computável resulta em falso!

Predicados computáveis

Programa lógico 5 – Quem é irmão de Caim?

```
(1) pai(adão,caim) ←
(2) pai(adão,abel) ←
(3) pai(adão,seth) ←
(4) irmão(X,Y) ← pai(Z,X), pai(Z,Y), X ≠ Y
```



Predicados computáveis

Exercício 4

Com base no programa a seguir, mostre como SLD-refutação responde à consulta (no Prolog, o operador '≠' é representado por '\='):

- ***Quem é infiel?***

Programa lógico 6

```
(1) gosta(ary,eva) ←
(2) gosta(ivo,ana) ←
(3) gosta(ary,bia) ←
(4) gosta(eva,ary) ←
(5) namora(A,B) ← gosta(A,B), gosta(B,A)
(6) infiel(C) ← namora(C,D), gosta(C,E), D ≠ E
```

Negação por falha finita

Hipótese do mundo fechado: tudo o que é verdadeiro está declarado!

Mecanismo de negação por falha finita

Ao encontrar um **literal negativo** ($\neg\lambda$) o sistema dispara uma sub-prova do **literal complementar** (λ):

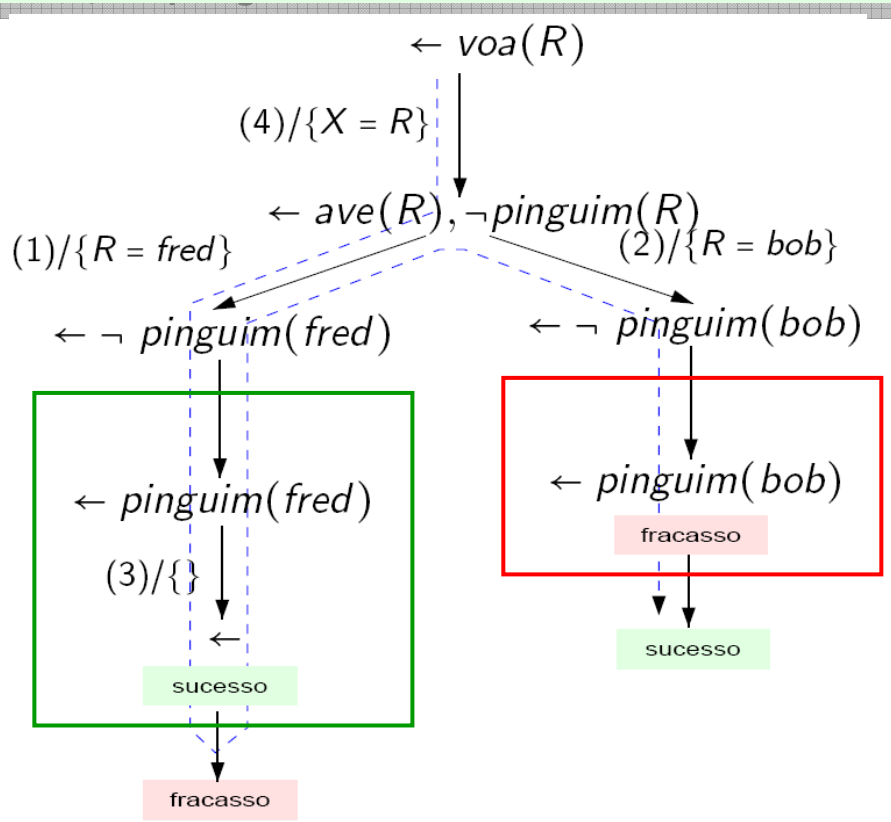
- se a prova de λ termina com sucesso, a prova de $\neg\lambda$ termina com fracasso
- se a prova de λ termina com fracasso, a prova de $\neg\lambda$ termina com sucesso

Prolog implementa negação por falha finita através do predicado computável `not/1`.

Negação por falha finita

Programa lógico 7 – Quem voa?

```
(1) ave(fred) ←
(2) ave(bob) ←
(3) pimguim(fred) ←
(4) voa(X) ← ave(X), ¬ pinguim(X)
```



Negação por falha finita

Exercício 5

Com base no programa a seguir, mostre como SLD-refutação responde às consultas:

- ← `diferente(bola,bola)`
- ← `diferente(bola,bala)`

Programa lógico 8

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

Fim