# Inferência em Lógica de Primeira Ordem

## Ano Lectivo 2010/2011 - $1^o$ Semestre – Capítulo 9

# Resumo

♢ Redução de inferência em lógica de primeira ordem à inferência em lógica proposicional

♢ Unificação

♢ Modus Ponens Generalizado

♢ Encadeamento para a frente e para trás

♢ Programação em Lógica

♢ Resolução

# Resenha Histórica

| | | |
|---|---|---|
| 450A.C. | Estóicos | lógica proposicional, inferência (possivelmente) |
| 322A.C. | Aristóteles | "silogismos" (regras de inferência), quantificadores |
| 1565 | Cardano | teoria da probabilidade (lógica proposicional + incerteza) |
| 1847 | Boole | lógica proposicional (novamente) |
| 1879 | Frege | lógica de primeira ordem |
| 1922 | Wittgenstein | prova por tabelas de verdade |
| 1930 | Gödel | $\exists$ algoritmo completo para LPO |
| 1930 | Herbrand | algoritmo completo para LPO (redução ao caso proposicional) |
| 1931 | Gödel | $\neg\exists$ algoritmo completo para aritmética |
| 1960 | Davis/Putnam | algoritmo "eficaz" para lógica proposicional |
| 1965 | Robinson | algoritmo "eficaz" para LPO—resolução |

# Métodos de Inferência para LPO

1. Proposicionalização
2. Resolução
3. Sequentes
4. Dedução Natural
5. Tableaux
6. Conexão de Matrizes
7. Reescrita de termos

# Instanciação Universal (IU)

Qualquer instanciação de uma frase quantificada universalmente é consequência desta última:

$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)}$$

para qualquer variável $v$ e termo básico (concreto) $g$

E.g., $\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$ origina

$$King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(Father(John)) \wedge Greedy(Father(John)) \;\Rightarrow\; Evil(Father(John))$$
$$\vdots$$

# Instanciação Existencial (IE)

Para qualquer frase $\alpha$, variável $v$, e símbolo de constante $k$
*que não ocorre na base de conhecimento*:

$$\frac{\exists\, v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

E.g., $\exists\, x \;\; Crown(x) \wedge OnHead(x, John)$ origina

$$Crown(C_1) \wedge OnHead(C_1, John)$$

desde que $C_1$ seja um novo símbolo de constante, designado por constante de Skolem

Outro exemplo: de $\exists\, x \;\; d(x^y)/dy = x^y$ obtemos

$$d(e^y)/dy = e^y$$

desde que $e$ seja um novo símbolo de constante.

# Instanciação Existencial (cont.)

IU pode ser aplicado repetidamente para *adicionar* novas frases;
a nova KB é logicamente equivalente à inicial

IE pode ser aplicada uma vez para *substituir* a frase existencial;
a nova KB *não é* equivalente à inicial, mas é satisfazível sse a KB inicial era satisfazível!

# Redução à inferência proposicional

Suponhamos que a KB contém apenas o seguinte:

$$\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

Instanciando a frase universal de *todas as maneiras possíveis*, ficamos com

$$King(John) \wedge Greedy(John) \;\Rightarrow\; Evil(John)$$
$$King(Richard) \wedge Greedy(Richard) \;\Rightarrow\; Evil(Richard)$$
$$King(John)$$
$$Greedy(John)$$
$$Brother(Richard, John)$$

A nova KB foi proposicionalizada: os símbolos proposicionais são

$King(John),\; Greedy(John),\; Evil(John), King(Richard)$ etc.

# Redução (cont.)

Resultado: uma frase básica é consequência da nova KB sse é consequência da KB original

Resultado: toda a KB em LPO pode ser proposicionalizada preservando a relação de consequência lógica

Ideia: proposicionalizar KB e pergunta, aplicar resolução, devolver resultado

Problema: com símbolos de função, existe um número infinito de termos básicos, e.g., $Father(Father(Father(John)))$

Teorema: Herbrand (1930). Se frase $\alpha$ é consequência de uma KB em LPO,
então é consequência de um subconjunto *finito* da KB proposicional

Ideia: De $n = 0$ até $\infty$ fazer
gerar uma KB prop. instanciando os termos com profundidade-$n$
verificar se $\alpha$ é consequência desta KB

Problema: funciona se $\alpha$ é consequência, pode não terminar se $\alpha$ não é consequência

Teorema: Turing (1936), Church (1936), consequência em LPO é semidecidível

# Problemas com a proposicionalização

A proposicionalização pode gerar inúmeras frases irrelevantes.
E.g., de

$$\forall x \;\; King(x) \wedge Greedy(x) \;\Rightarrow\; Evil(x)$$
$$King(John)$$
$$\forall y \;\; Greedy(y)$$
$$Brother(Richard, John)$$

é óbvio que $Evil(John)$, mas a proposicionalização produz muitos factos, por exemplo $Greedy(Richard)$, que são irrelevantes

Com $p$ predicados $k$-ários e $n$ constantes, existem $p \cdot n^k$ instanciações!

# Unificação

Podemos obter a conclusão imediatamente se conseguirmos encontrar uma substituição $\theta$ tal que $King(x)$ e $Greedy(x)$ concordem com $King(John)$ e $Greedy(y)$

$\theta = \{x/John, y/John\}$ funciona

$\text{UNIFICAR}(\alpha, \beta) = \theta$ se $\alpha\theta = \beta\theta$

| $p$ | $q$ | $\theta$ |
|---|---|---|
| $Knows(John, x)$ | $Knows(John, Jane)$ | |
| $Knows(John, x)$ | $Knows(y, OJ)$ | |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | |
| $Knows(John, x)$ | $Knows(x, OJ)$ | |

# Unificação

Podemos obter a conclusão imediatamente se conseguirmos encontrar uma substituição $\theta$ tal que $King(x)$ e $Greedy(x)$ concordem com $King(John)$ e $Greedy(y)$

$\theta = \{x/John, y/John\}$ funciona

$\text{UNIFICAR}(\alpha, \beta) = \theta$ se $\alpha\theta = \beta\theta$

| $p$ | $q$ | $\theta$ |
|---|---|---|
| $Knows(John, x)$ | $Knows(John, Jane)$ | $\{x/Jane\}$ |
| $Knows(John, x)$ | $Knows(y, OJ)$ | |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | |
| $Knows(John, x)$ | $Knows(x, OJ)$ | |

# Unificação

Podemos obter a conclusão imediatamente se conseguirmos encontrar uma substituição $\theta$ tal que $King(x)$ e $Greedy(x)$ concordem com $King(John)$ e $Greedy(y)$

$\theta = \{x/John, y/John\}$ funciona

$\text{UNIFICAR}(\alpha, \beta) = \theta$ se $\alpha\theta = \beta\theta$

| $p$ | $q$ | $\theta$ |
|---|---|---|
| $Knows(John, x)$ | $Knows(John, Jane)$ | $\{x/Jane\}$ |
| $Knows(John, x)$ | $Knows(y, OJ)$ | $\{x/OJ, y/John\}$ |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | |
| $Knows(John, x)$ | $Knows(x, OJ)$ | |

# Unificação

Podemos obter a conclusão imediatamente se conseguirmos encontrar uma substituição $\theta$ tal que $King(x)$ e $Greedy(x)$ concordem com $King(John)$ e $Greedy(y)$

$\theta = \{x/John, y/John\}$ funciona

$\text{UNIFICAR}(\alpha, \beta) = \theta$ se $\alpha\theta = \beta\theta$

| $p$ | $q$ | $\theta$ |
|---|---|---|
| $Knows(John, x)$ | $Knows(John, Jane)$ | $\{x/Jane\}$ |
| $Knows(John, x)$ | $Knows(y, OJ)$ | $\{x/OJ, y/John\}$ |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | $\{y/John, x/Mother(John)\}$ |
| $Knows(John, x)$ | $Knows(x, OJ)$ | |

# Unificação

Podemos obter a conclusão imediatamente se conseguirmos encontrar uma substituição $\theta$ tal que $King(x)$ e $Greedy(x)$ concordem com $King(John)$ e $Greedy(y)$

$\theta = \{x/John, y/John\}$ funciona

$\text{UNIFICAR}(\alpha, \beta) = \theta$ se $\alpha\theta = \beta\theta$

| $p$ | $q$ | $\theta$ |
|---|---|---|
| $Knows(John, x)$ | $Knows(John, Jane)$ | $\{x/Jane\}$ |
| $Knows(John, x)$ | $Knows(y, OJ)$ | $\{x/OJ, y/John\}$ |
| $Knows(John, x)$ | $Knows(y, Mother(y))$ | $\{y/John, x/Mother(John)\}$ |
| $Knows(John, x)$ | $Knows(x, OJ)$ | $fail$ |

Standardização evita colisões de nomes de variáveis, e.g., $Knows(z_{17}, OJ)$

# Modus Ponens Generalizado (MPG)

$$\frac{p_1',\ p_2',\ \ldots,\ p_n',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

em que $p_i'\theta = p_i\theta$ para todo $i$

| | |
|---|---|
| $p_1'$ é $King(John)$ | $p_1$ é $King(x)$ |
| $p_2'$ é $Greedy(y)$ | $p_2$ é $Greedy(x)$ |
| $\theta$ é $\{x/John, y/John\}$ | $q$ é $Evil(x)$ |
| $q\theta$ é $Evil(John)$ | |

MPG utilizando KB de cláusulas definidas (*exactamente* um literal positivo)

Todas as variáveis estão implicitamente quantificadas universalmente

# MPG é sólido

Temos de demonstrar que

$$p_1', \ \ldots, \ p_n', \ \ (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$$

desde que $p_i'\theta = p_i\theta$ para todo o $i$

Lema: Para qualquer cláusula definida $p$, temos $p \models p\theta$ por IU

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$

2. $p_1', \ \ldots, \ p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$

3. De 1 e 2, $q\theta$ conclui-se por Modus Ponens

# Exemplo de base de conhecimento

A lei afirma que é crime um Americano vender armas a nações hostis. O país Nono, é um inimigo da América, tem alguns mísseis, e todos esses mísseis forma vendidos pelo Coronel West, que é Americano.

Provar que Cor. West é criminoso

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono . . . tem alguns mísseis

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono . . . tem alguns mísseis, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . todos os seus mísseis foram-lhe vendidos pelo Coronel West

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono . . . tem alguns mísseis, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . todos os seus mísseis foram-lhe vendidos pelo Coronel West

$\forall x\ \ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

Mísseis são armas:

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono . . . tem alguns mísseis, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . todos os seus mísseis foram-lhe vendidos pelo Coronel West

$$\forall x\ \ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

Mísseis são armas:

$$Missile(x) \Rightarrow Weapon(x)$$

Um inimigo da América é "hostil":

# Exemplo de base de conhecimento (cont.)

. . . é crime um Americano vender armas a nações hostis:

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$$

Nono . . . tem alguns mísseis, i.e., $\exists x\ Owns(Nono, x) \land Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

. . . todos os seus mísseis foram-lhe vendidos pelo Coronel West

$$\forall x\ \ Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

Mísseis são armas:

$$Missile(x) \Rightarrow Weapon(x)$$

Um inimigo da América é "hostil":

$$Enemy(x, America) \Rightarrow Hostile(x)$$

West, é Americano . . .

$$American(West)$$

O país Nono, um inimigo da América . . .

$$Enemy(Nono, America)$$

# Algoritmo de Encadeamento para a Frente

**function** FOL-FC-ASK($KB, \alpha$) **returns** a substitution or *false*

**repeat until** *new* is empty

$new \leftarrow \{\}$

**for each** sentence $r$ **in** $KB$ **do**

$(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)

**for each** $\theta$ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$

for some $p'_1, \ldots, p'_n$ in $KB$

$q' \leftarrow$ SUBST($\theta, q$)

**if** $q'$ is not a renaming of a sentence already in $KB$ or $new$ **then do**

add $q'$ to $new$

$\phi \leftarrow$ UNIFY($q', \alpha$)

**if** $\phi$ is not *fail* **then return** $\phi$

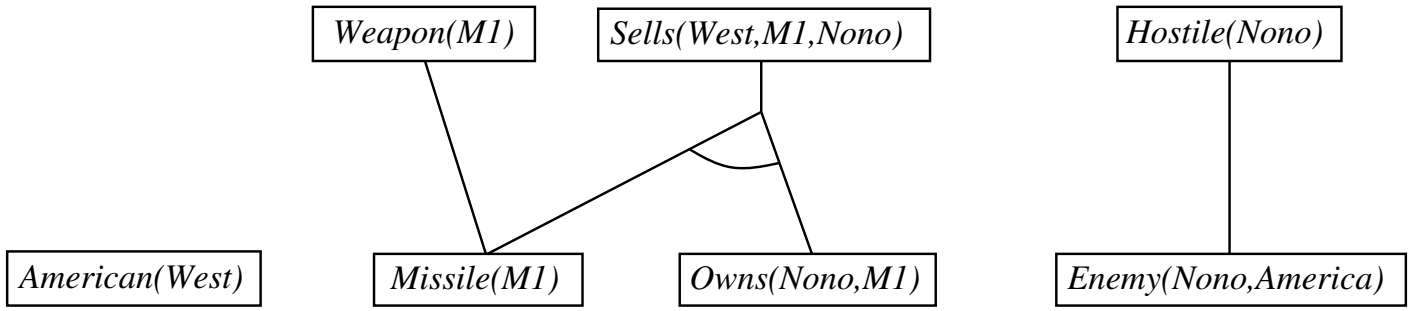
add $new$ to $KB$

**return** *false*

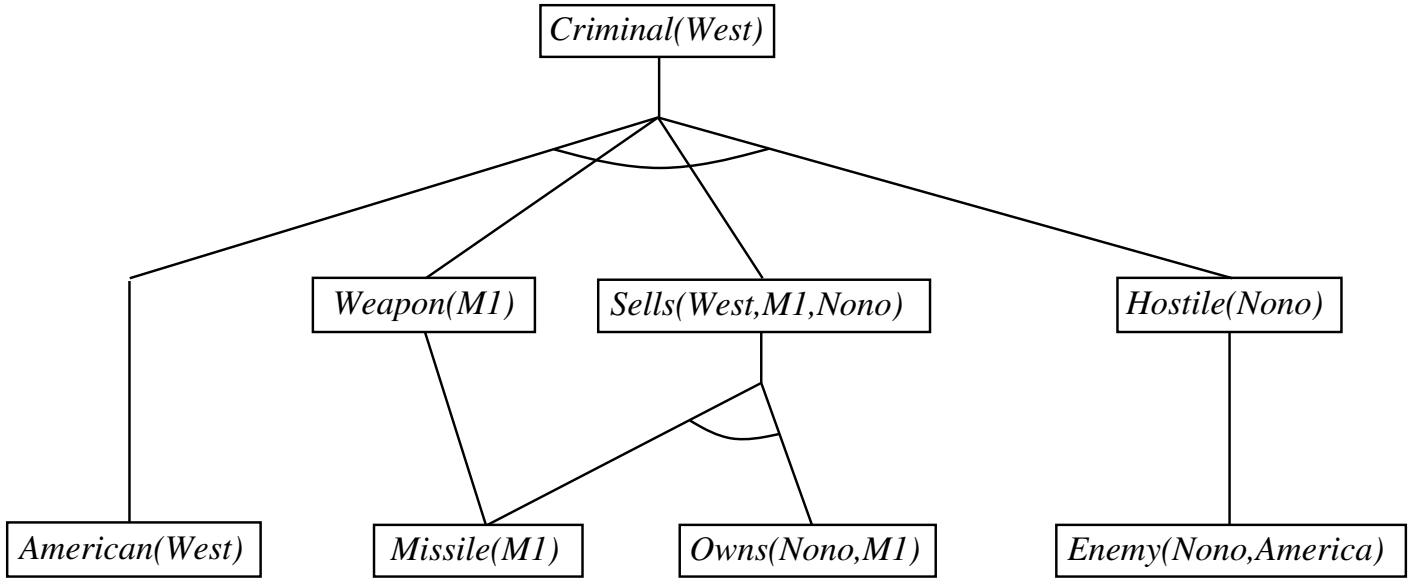
# Prova por Encadeamento para a Frente

*American(West)* | *Missile(M1)* | *Owns(Nono,M1)* | *Enemy(Nono,America)*

# Prova por Encadeamento para a Frente



# Prova por Encadeamento para a Frente



# Propriedades do encadeamento para a frente

Correcto para cláusulas definidas de primeira-ordem
(demonstração semelhante à do caso proposicional)

Datalog = cláusulas definidas de primeira-ordem + *sem funções* (KB crime)
EF termina para Datalog num número polinomial de iterações: limite máximo de $p \cdot n^k$ literais

Pode não terminar no caso geral se $\alpha$ não é consequência

Inevitável: consequência com cláusulas definidas é semidecidível

# Eficiência de encadeamento para a frente

Observação simples: não é necessário utilizar uma regra na iteração $k$
se a premissa não foi adicionada na iteração $k-1$

$\Rightarrow$ testar regras cuja premissa contém apenas literais adicionados recentemente

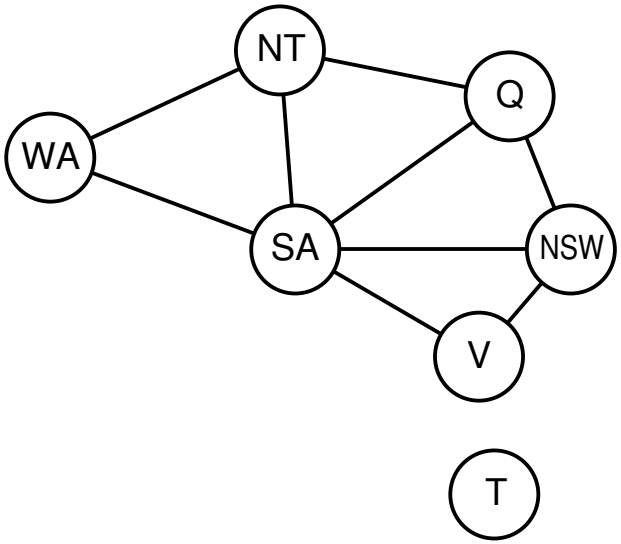
O mecanismo de concordância pode ser dispendioso

Indexação nas Bases de Dados permite a obtenção em tempo $O(1)$ de factos conhecidos. A consulta $Missile(x)$ devolve $Missile(M_1)$

Concordância de premissas conjuntivas com factos conhecidos é NP-difícil

Encadeamento para a frente é amplamente utilizado em bases de dados dedutivas

# Exemplo de Concordância Difícil




$$\begin{aligned}
&\mathit{Diff}(wa, nt) \wedge \mathit{Diff}(wa, sa) \wedge \\
&\qquad \mathit{Diff}(nt, q)\mathit{Diff}(nt, sa) \wedge \\
&\qquad \mathit{Diff}(q, nsw) \wedge \mathit{Diff}(q, sa) \wedge \\
&\qquad \mathit{Diff}(nsw, v) \wedge \mathit{Diff}(nsw, sa) \wedge \\
&\qquad \mathit{Diff}(v, sa) \Rightarrow Colorable()
\end{aligned}$$

$$\mathit{Diff}(Red, Blue) \quad \mathit{Diff}(Red, Green)$$
$$\mathit{Diff}(Green, Red) \quad \mathit{Diff}(Green, Blue)$$
$$\mathit{Diff}(Blue, Red) \quad \mathit{Diff}(Blue, Green)$$

$Colorable()$ é inferido sse o CSP tem uma solução
CSPs incluem 3SAT como caso especial, logo a concordância é NP-difícil

# Algoritmo de encadeamento para trás

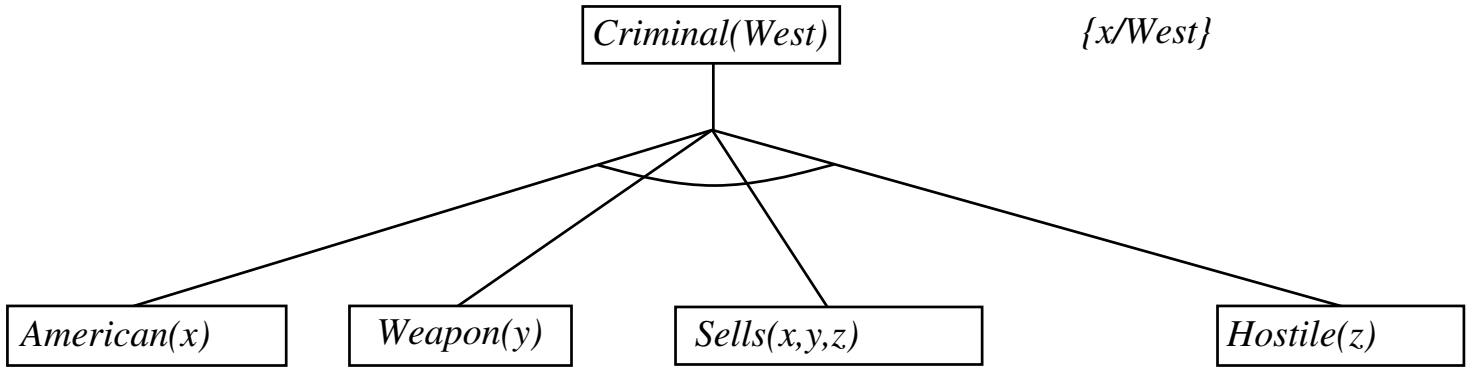
**function** FOL-BC-ASK($KB$, goals, $\theta$) **returns** a set of substitutions
  **inputs**: $KB$, a knowledge base
    $goals$, a list of conjuncts forming a query
    $\theta$, the current substitution, initially the empty substitution $\{\}$
  **local variables**: $ans$, a set of substitutions, initially empty

  **if** $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(goals))$
  **for each** $r$ **in** $KB$ where $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow \text{UNIFY}(q, q')$ succeeds
    $ans \leftarrow \text{FOL-BC-ASK}(KB, [p_1, \ldots, p_n | \text{REST}(goals)], \text{COMPOSE}(\theta', \theta)) \cup ans$
  **return** $ans$

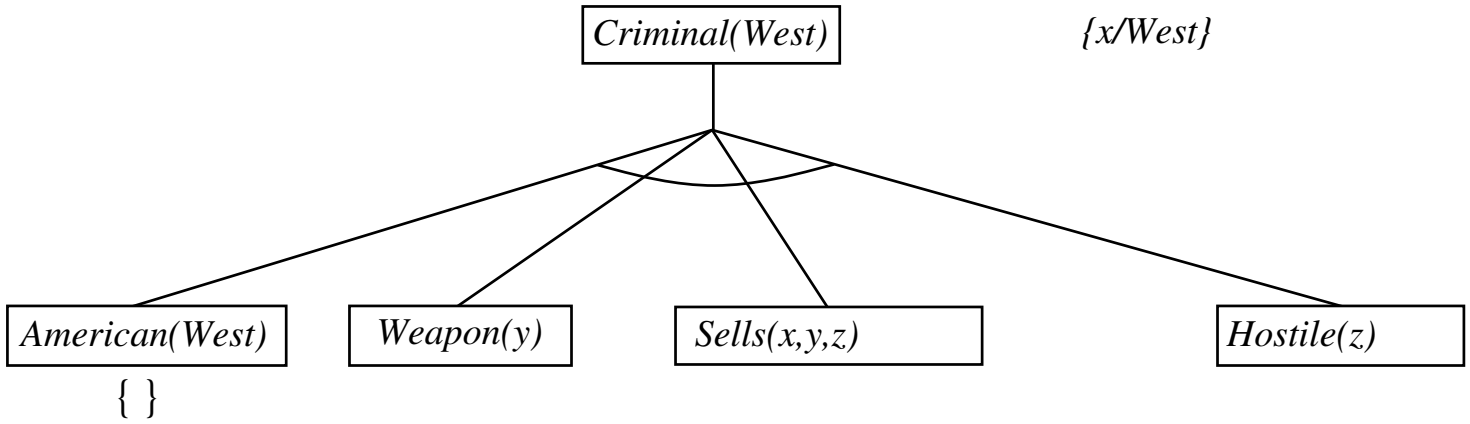
# Exemplo de Encadeamento para trás

*Criminal(West)*

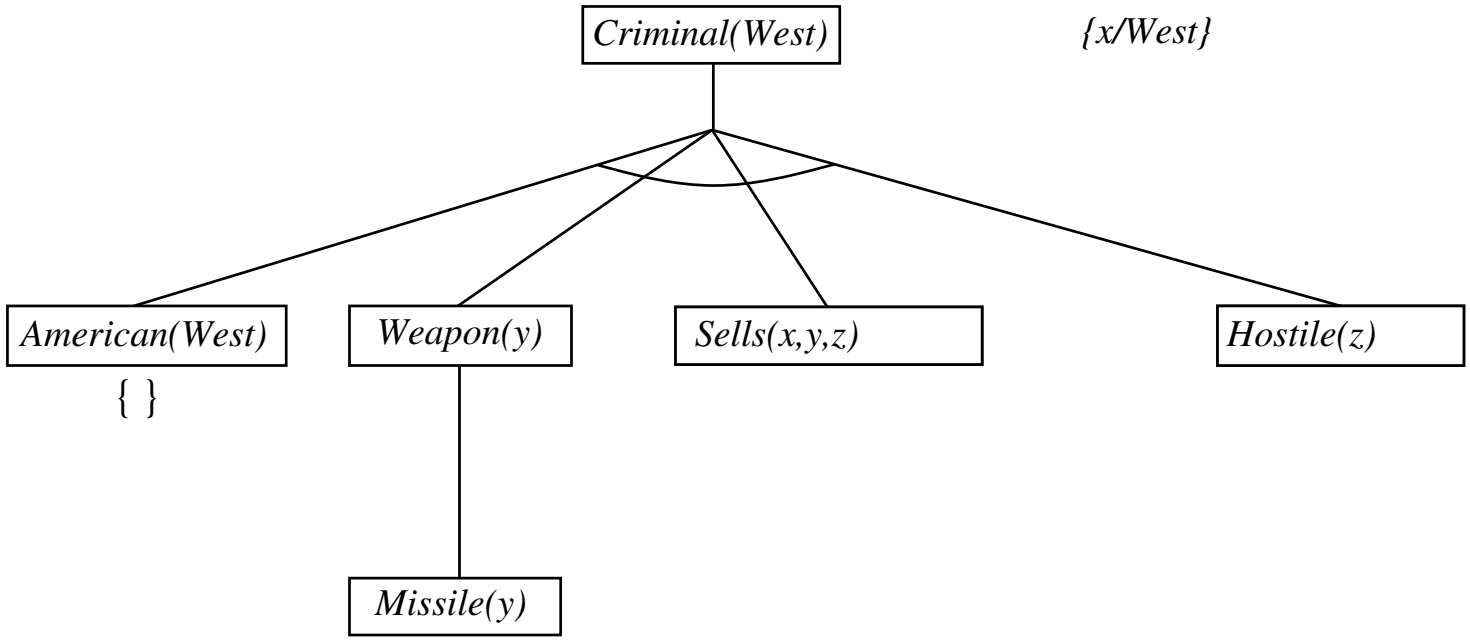
# Exemplo de Encadeamento para trás



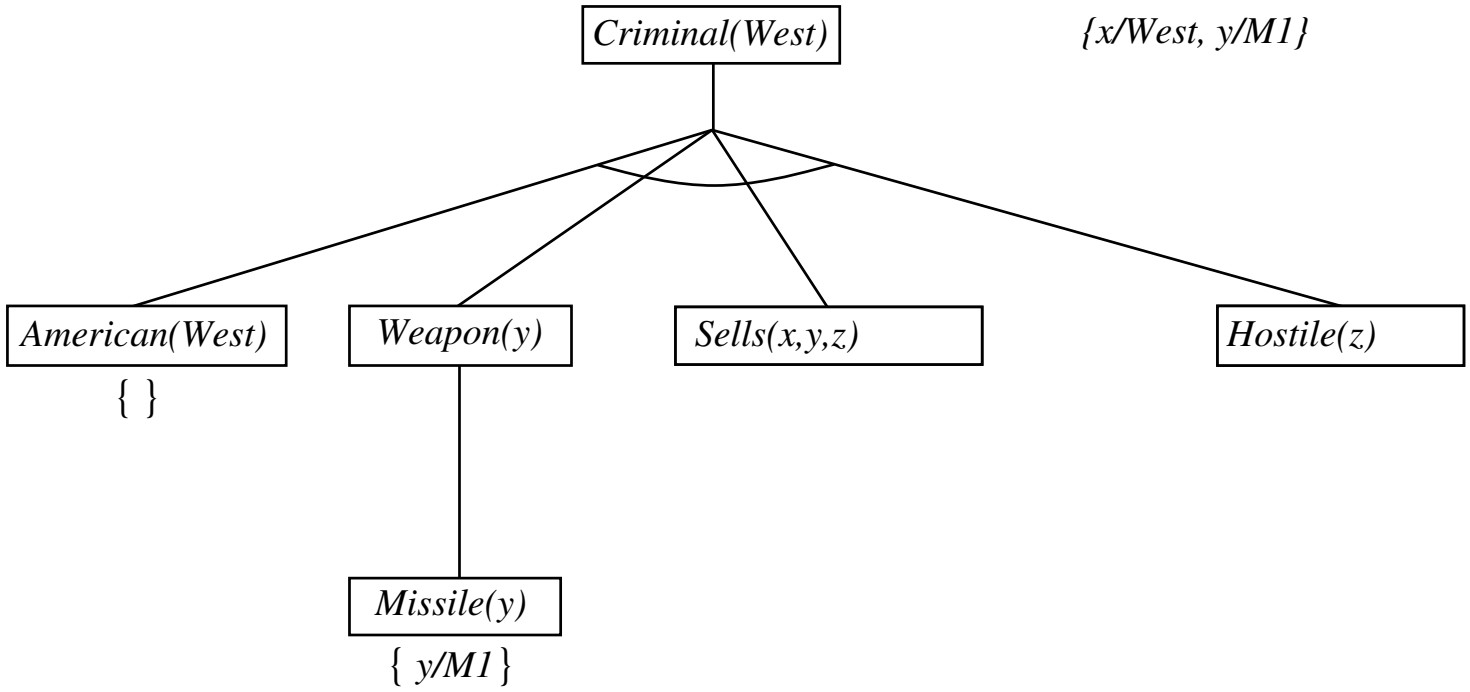
# Exemplo de Encadeamento para trás



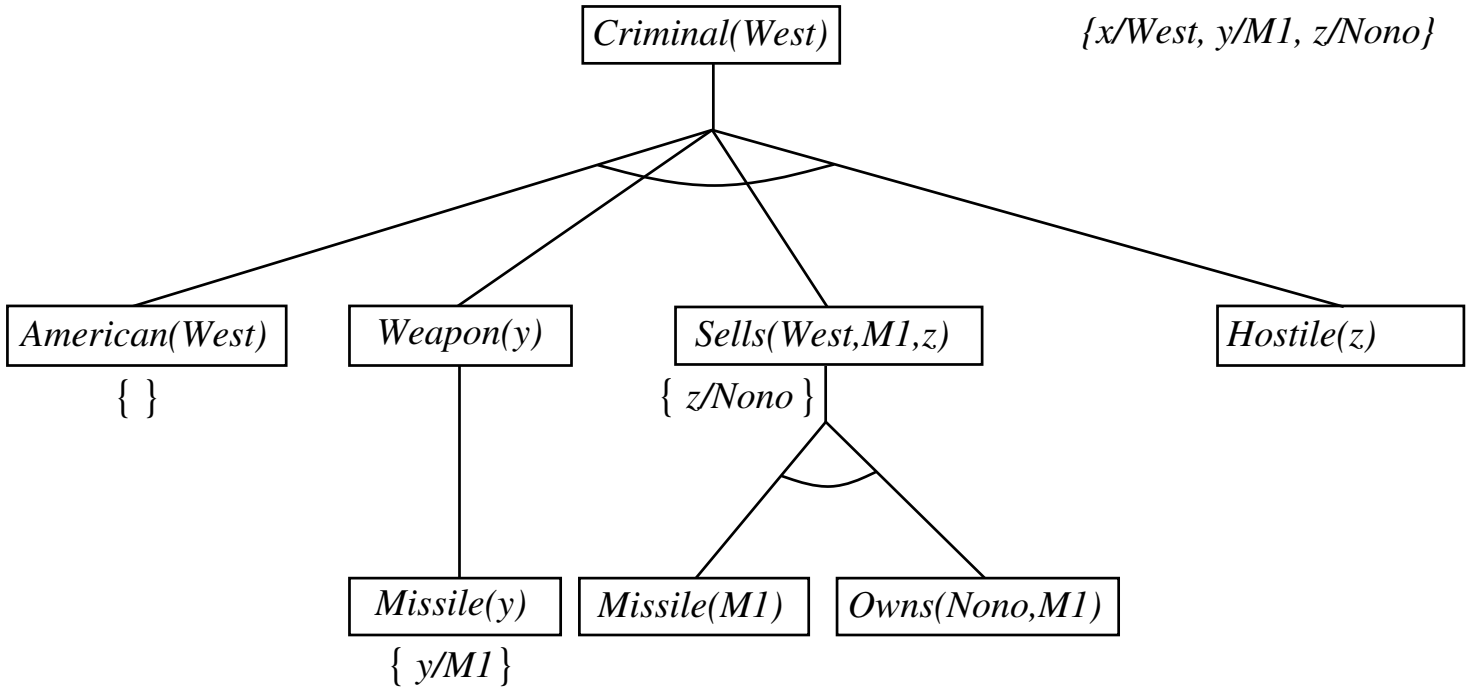
# Exemplo de Encadeamento para trás



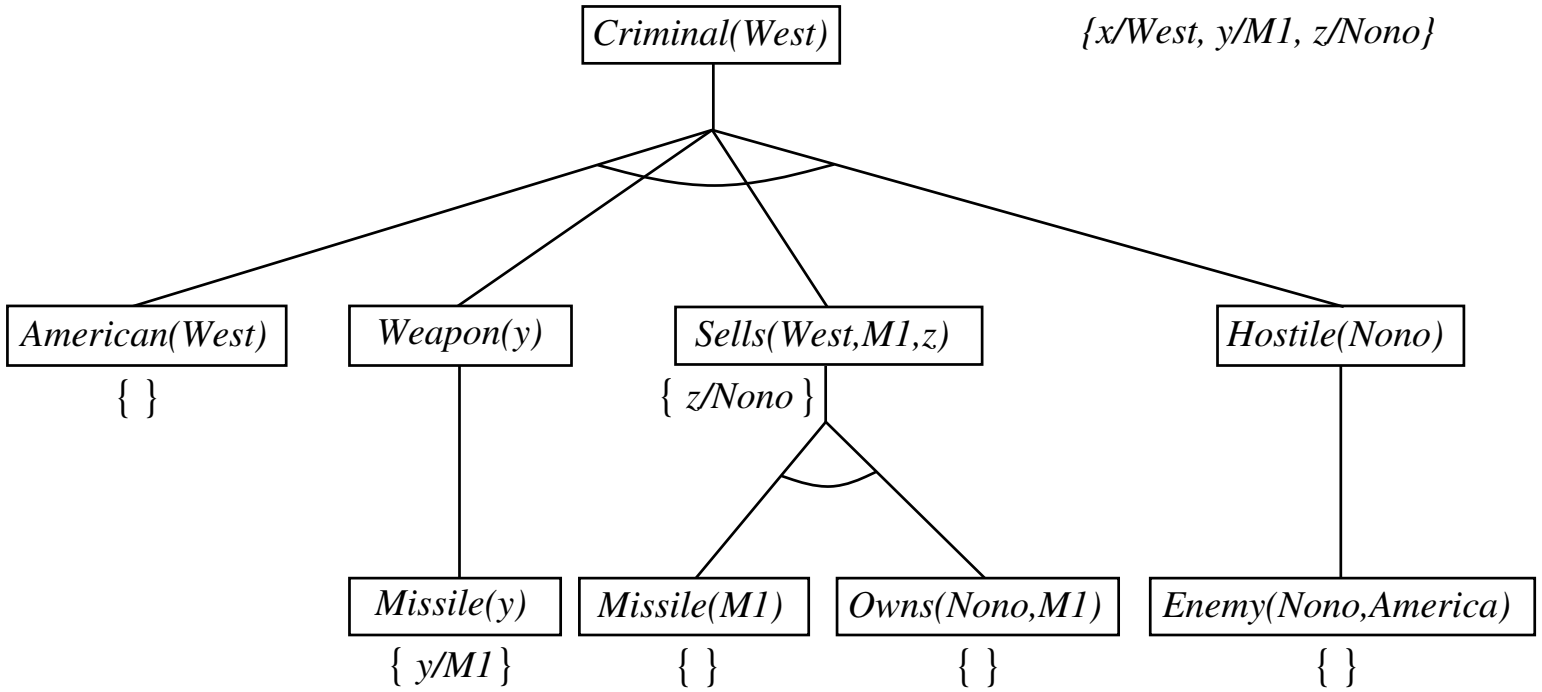
# Exemplo de Encadeamento para trás



# Exemplo de Encadeamento para trás



# Exemplo de Encadeamento para trás



# Propriedades do encadeamento para trás

Pesquisa da prova recursivamente em profundidade primeiro: espaço é linear no tamanho da prova

Incompleto devido a ciclos infinitos

$\Rightarrow$ verificação do objectivo corrente com todos os outros na pilha

Ineficiente devido às subconsultas repetidas (de sucesso e de falha)

$\Rightarrow$ memorização dos resultados anteriores (espaço extra!)

Amplamente utilizado (sem melhoramentos!) na programação em lógica

# Programação em Lógica

Moto: computação como inferência em KBs lógicas

| | Programação em Lógica | Programação Usual |
|---|---|---|
| 1. | Identificar problema | Identificar problema |
| 2. | Coligir informação | Coligir informação |
| 3. | Pausa para café | Descobrir solução |
| 4. | Codificar informação na KB | Programar solução |
| 5. | Representar instância com factos | Representar instância com dados |
| 6. | Efectuar consultas | Aplicar programa aos dados |
| 7. | Encontrar factos errados | Depurar erros procedimentais |

Deve ser mais fácil depurar $Capital(NewYork, US)$ do que $x := x + 2$ !

# Sistemas Prolog

Essência: encadeamento para trás com cláusulas de Horn
Muito utilizadao na Europa, Japão (base do projecto da 5ª geração)
Técnicas de compilação ⇒ 60 milhões de LIPS

Programa = conjunto de cláusulas = `head :- literal`$_1$`, ... literal`$_n$`.`

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

- Unificação eficiente sem teste de ocorrência
- Obtenção eficiente de cláusulas
- Encadeamento para trás em profundidade primeiro, da esquerda para a direita
- Predicados de sistema para efectuar aritmética etc., e.g., `X is Y*Z+3`
- Assunção do Mundo Fechado ("negação por falha")
  e.g., dado `alive(X) :- not dead(X).`
  `alive(joe)` sucede se `dead(joe)` falha

# Exemplo Criminoso em Prolog

♢ Programa:

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
sells(west,X,nono) :- missile(X), owns(nono,X).
owns(nono,m1).
missile(m1).
weapon(X) :- missile(Y).
hostile(X) :- enemy(X,america).
american(west).
enemy(nono,america).
```

♢ Interrogação:

```
|?- criminal(Who).
Who = west;
no
```

# Prova que West é criminoso em Prolog

```
                    ?- criminal(Who).
                              |
?- american(Who), weapon(Y1), sells(Who,Y1,Z1), hostile(Z1).
                              |                              Who = west
       ?- weapon(Y1), sells(west,Y1,Z1), hostile(Z1).
                              |
      ?- missile(Y1), sells(west,Y1,Z1), hostile(Z1).
                              |                              Y1 = m1
            ?- sells(west,m1,Z1), hostile(Z1).
                              |                              Z1 = nono
       ?- missile(m1), owns(nono,m1), hostile(nono).
                              |
             ?- owns(nono,m1), hostile(nono).
                              |
                    ?- hostile(nono).
                              |
                 ?- enemy(nono,america).
                              |
                             ?-
```

# Resolução binária: breve sumário

Versão para lógica de primeira ordem:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

em que $\textsc{Unificar}(\ell_i, \neg m_j) = \theta$.

Por exemplo,

$$\frac{\neg Rich(x) \vee Unhappy(x) \quad Rich(Ken)}{Unhappy(Ken)}$$

com $\theta = \{x/Ken\}$

A regra de resolução binária não é completa.

# Factorização

Seja $C'$ um subconjunto de literais com o mesmo sinal de uma clásula $C$ e unificável com unificador mais geral $\theta$. A cláusula $C\theta$ é um factor de $C$.

A regra de factorização autoriza a adição de qualquer factor de uma cláusula ao conjunto de cláusulas.

Exemplo:

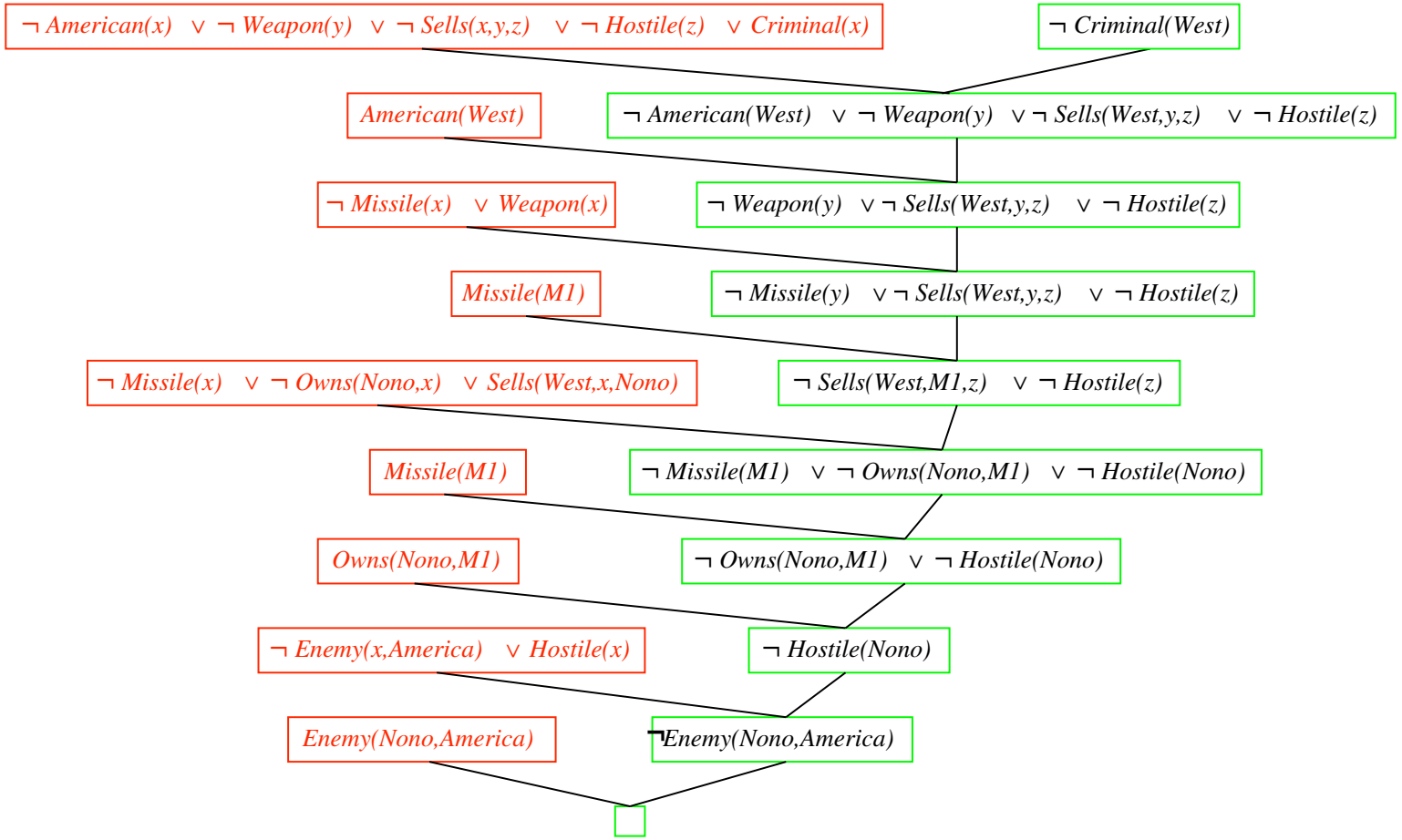
| | | |
|---|---|---|
| 1. | $P(x, f(x), z) \vee P(u, w, w)$ | axioma |
| 2. | $\neg P(x, y, z) \vee \neg P(A, z, z)$ | axioma |
| 3. | $P(x_1, f(x_1), f(x_1))$ | factor de $1$. |
| 4. | $\neg P(A, z_1, z_1))$ | factor de $2$. |
| 5. | $\square$ | resolvente de $3$. e $4$. |

Aplicar resolução a $CNF(KB \wedge \neg\alpha)$

resolução binária + factorização $\Rightarrow$ algoritmo completo para LPO

NOTA: Resolução só é aplicável a KBs na forma clausal (disjunções de literais com todas as variáveis quantificadads universalmente).

# Prova por resolução: cláusulas definidas



# Conversão para forma clausal

Toda a pessoa que ama todos os animais é amada por alguém:

$$\forall x \ [\forall y \ \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y \ \ Loves(y, x)]$$

1. Eliminar bicondicionais e implicações

$$\forall x \ [\neg\forall y \ \ \neg Animal(y) \vee Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$

2. Deslocar $\neg$ para dentro: $\neg\forall x, p \equiv \exists x \ \neg p$, $\neg\exists x, p \equiv \forall x \ \neg p$:

$$\forall x \ [\exists y \ \ \neg(\neg Animal(y) \vee Loves(x, y))] \vee [\exists y \ \ Loves(y, x)]$$
$$\forall x \ [\exists y \ \ \neg\neg Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$
$$\forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists y \ \ Loves(y, x)]$$

# Conversão para forma clausal (cont.)

3. Standardizar variáveis: cada quantificadar deve usar uma diferente

$$\forall x \ [\exists y \ \ Animal(y) \wedge \neg Loves(x, y)] \vee [\exists z \ \ Loves(z, x)]$$

4. Skolemizar: uma forma mais geral de instanciação existencial.
   Cada variável existencial é substituída por uma função de Skolem
   das variáveis quantificadas universalmente que a incluem:

$$\forall x \ [Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

5. Remover quantificadores universais:

$$[Animal(F(x)) \wedge \neg Loves(x, F(x))] \vee Loves(G(x), x)$$

6. Distribuir $\wedge$ por $\vee$:

$$[Animal(F(x)) \vee Loves(G(x), x)] \wedge [\neg Loves(x, F(x)) \vee Loves(G(x), x)]$$

# A Curiosidade matou o Gato?

1. Toda a gente que ama todos os animais é amada por alguém.

   $$\forall_x \left[\forall_y \left(Animal(y) \Rightarrow Loves(x, y)\right)\right] \Rightarrow \left[\exists_y \; Loves(x, y)\right]$$

2. Qualquer pessoa que mata um animal não é amada por ninguém.

   $$\forall_x \left[\exists_z \left(Animal(z) \wedge Kills(x, z)\right)\right] \Rightarrow \left[\neg\exists_y \; Loves(y, x)\right]$$

3. Jack ama todos os animais

   $$\forall_x \; Animal(x) \Rightarrow Loves(Jack, x)$$

4. O Jack ou a Curiosidade mataram o gato, que se chama Tuna.

   $$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$$

5. Todos os gatos são animais

   $$\forall_x \; Cat(x) \Rightarrow Animal(, x)$$

6. A curiosidade matou o gato?

   $$\neg Kills(Curiosity, Tuna)$$

# Conversão para FNC

1. Toda a gente que ama todos os animais é amada por alguém.

   $Animal(F(x)) \vee Loves(G(x), x)$

   $\neg Loves(x, F(x)) \vee Loves(G(x), x)$

2. Qualquer pessoa que mata um animal não é amada por ninguém.

   $\neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kills(x, z)$

3. Jack ama todos os animais

   $\neg Animal(x) \vee Loves(Jack, x)$

4. O Jack ou a Curiosidade mataram o gato, que se chama Tuna.

   $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

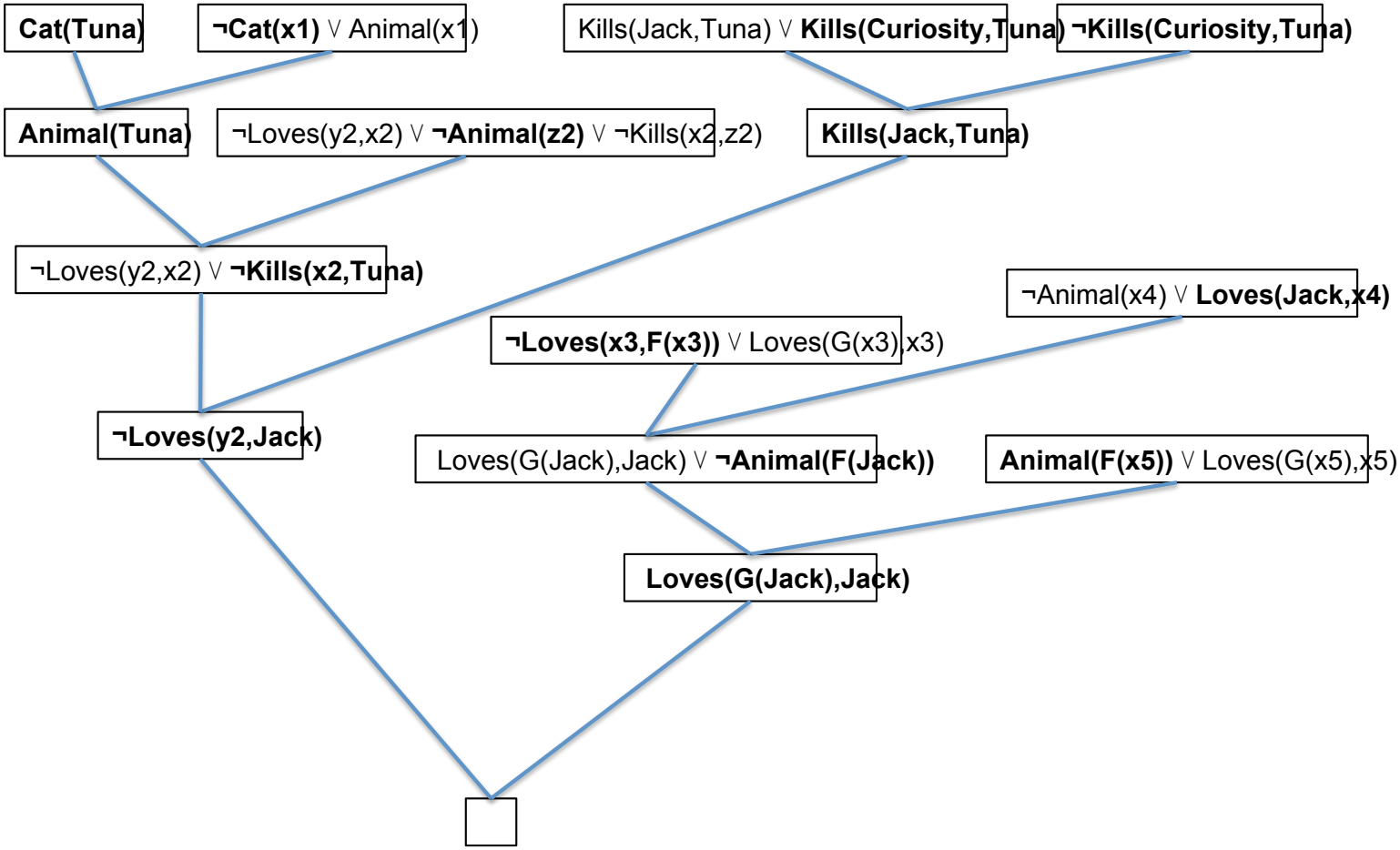
5. Todos os gatos são animais

   $\neg Cat(x) \vee Animal(, x)$

6. A curiosidade matou o gato?

   $\neg Kills(Curiosity, Tuna)$

# Prova que a curiosidade matou o gato!



# Tratamento da igualdade

A igualdade introduz problemas extra no algoritmo de inferência. Existem duas grandes classes de aproximações para lidar com o predicado de igualdade:

1. Através da inclusão dos axiomas para a igualdade.
2. Recorrendo a regras de inferência adicionais.

# Axiomas para a igualdade

Axiomas básicos:

$$\forall\, x \;\; x = x$$
$$\forall\, x \;\; \forall\, y \;\; x = y \;\Rightarrow\; y = x$$
$$\forall\, x \;\; \forall\, y \;\; \forall\, z \;\; x = y \wedge y = z \;\Rightarrow\; x = z$$

Para cada predicado $P/n$ e para cada $1 \leq i \leq n$

$$\forall\, x_1, \ldots, x_i, \ldots, x_n \;\; \forall\, y$$
$$x_i = y \;\Rightarrow\; (P(x_1, \ldots, x_i, \ldots, x_n) \equiv P(x_1, \ldots, y, \ldots, x_n))$$

Para cada símbolo de função $f/n$ e para cada $1 \leq i \leq n$

$$\forall\, x_1, \ldots, x_i, \ldots, x_n \;\; \forall\, y$$
$$x_i = y \;\Rightarrow\; (f(x_1, \ldots, x_i, \ldots, x_n) = f(x_1, \ldots, y, \ldots, x_n))$$

Recorre-se depois ao método de resolução binária com factorização.

# Demodulação

Para quaisquer termos $x, y$ e $z$ tal que $UNIFY(x, z) = \theta$ e $m_n[z]$ é um literal contendo $z$:

$$\frac{x = y, \qquad m_1 \vee \ldots \vee m_n[z]}{m_1 \vee \ldots \vee m_n[SUBST(\theta, y)]}$$

A regra da Demodulação é incompleta.

Exemplo:

$$\frac{0 + z_1 = z_1 \qquad P(0 + (0 + 2)) \vee Q(3)}{P(0 + 2) \vee Q(3)}$$

# Paramodulação

Para quaisquer termos $x, y$ e $z$ tal que $UNIFY(x, z) = \theta$:

$$\frac{l_1 \vee \ldots \vee l_k \vee x = y, \qquad m_1 \vee \ldots \vee m_n[z]}{SUBST(\theta, l_1 \vee \ldots \vee l_k \vee m_1 \vee \ldots \vee m_n[y])}$$

A regra da Paramodulação é completa quando combinada com factorização, resolução binária e axiomas de reflexividade para variáveis e funções.

Exemplo:

$$\frac{P(x_1) \vee f(x_1, h(y_1)) = g(x_1, y_1), \qquad Q(h(f(h(x_2), h(a))))}{P(h(x_2)) \vee Q(h(g(h(x_2), a)))}$$

Com

$$x = f(x_1, h(y_1))$$
$$y = g(x_1, y_1)$$
$$z = f(h(x_2), h(a))$$

# Estratégias de resolução

♢ Preferência pelas cláusulas unitárias – prefere resoluções envolvendo pelo menos uma cláusula contendo só um literal (cláusula unitária)

♢ Resolução Unitária – só efectua resoluções em que pelo menos uma das cláusulas é unitária. Método incompleto.

Para o caso de cláusulas de Horn, o método é completo. Assemelha-se ao encadeamento para a frente.

♢ Conjunto de suporte – Identifica-se inicialmente um conjunto de cláusulas (o conjunto de suporte – set of support). Qualquer resolução combina uma cláusula do conjunto de suporte com outra cláusula, juntando a resolvente ao conjunto de suporte.

Se não houver cuidado, o método pode ser incompleto. Escolhe-se normalmente como conjunto de suporte inicial a negação da fórmula que se pretende demonstrar.

# Estratégias de resolução

♢ Resolução de entrada (input resolution)

Combina sempre uma das cláusulas de entrada (na base de conhecimento ou interrogação) com outra cláusula. Completa para cláusulas de Horn.

♢ Resolução linear

Método completo em que se permite resolver $P$ com $Q$ desde que $P$ esteja na base de conhecimento ou $P$ é uma antecessor de $Q$ na árvore de prova. Método completo.

# Demonstadores de Teoremas (Otter)

OTTER (Organized Techniques for Theorem Proving and Effective Research) obriga a que o utilizador divida o seu conhecimento em 4 partes:

1. Conjunto de suporte.
2. Axiomas utilizáveis, aqueles que estão fora do conjunto de suporte. Capturam o conhecimento do domínio.
3. Conjunto de equações designados "demoduladores' utilizados para simplificar os termos para uma forma canónica (e.g. $x + 0 = 0$).
4. Parâmetros que controlam a estratégia de controlo.

# Algoritmo do OTTER

**procedure** OTTER(*sos*, *usable*)
  **inputs**: *sos*, a set of support—clauses defining the problem (a global variable)
    *usable*, background knowledge potentially relevant to the problem
  **repeat**
    clause ← the lightest member of *sos*
    move *clause* from *sos* to *usable*
    PROCESS(INFER(*clause*, *usable*), *sos*)
  **until** *sos* = [] **or** a refutation has been found

---

**function** INFER(*clause*, *usable*) **returns** clauses
  resolve *clause* with each member of *usable*
  **return** the resulting clauses after applying FILTER

Efectua uma procura pelo melhor primeiror recorrendo à noção de peso de uma cláusula. Cláusulas unitárias são as mais leves.

$FILTER$ remove cláusulas consideradas não interessantes.

# Algoritmo do OTTER

**procedure** PROCESS(*clauses*, *sos*)

**for each** *clause* **in** *clauses* **do**

*clause* ← SIMPLIFY(*clause*)

merge identical literals

discard clause if it is a tautology

*sos* ← [*clause* || *sos*]

**if** *clause* has no literals **then** a refutation has been found

**if** *clause* has one literal **then** look for unit refutation

**end**

O Prover9, sucessor do OTTER, recorre a outras regras de pós-processamento e de simplificação que podem ser encontradas em `http://www.mcs.anl.gov/~mccune/prover9`.

# Exemplo de utilização do Prover9

Demonstrar que em qualquer grupo a solução da equação $a * x = b$ é única.

```
formulas(assumptions).

  e * x = x                    # label(left_identity).
  x' * x = e                   # label(left_inverse).
  (x * y) * z = x * (y * z)    # label(associativity).

end_of_list.

formulas(goals).

 (all x all y (((a * x = b) & (a * y = b)) -> (x = y))) # label(usol).

end_of_list.
```

# Pré-processamento da teoria

```
formulas(sos).
2 e * x = x # label(left_identity).  [assumption].
3 x' * x = e # label(left_inverse).  [assumption].
4 (x * y) * z = x * (y * z) # label(associativity).  [assumption].
5 a * c1 = b # label(usol).  [deny(1)].
6 a * c2 = b # label(usol).  [deny(1)].
7 c2 != c1 # label(usol) # answer(usol).  [deny(1)].
end_of_list.

formulas(demodulators).
2 e * x = x # label(left_identity).  [assumption].
3 x' * x = e # label(left_inverse).  [assumption].
4 (x * y) * z = x * (y * z) # label(associativity).  [assumption].
5 a * c1 = b # label(usol).  [deny(1)].
6 a * c2 = b # label(usol).  [deny(1)].
end_of_list.
```

# Prova obtida com o Prover9

```
2 e * x = x # label(left_identity).  [assumption].
3 x' * x = e # label(left_inverse).  [assumption].
4 (x * y) * z = x * (y * z) # label(associativity).  [assumption].
5 a * c1 = b # label(usol).  [deny(1)].
6 a * c2 = b # label(usol).  [deny(1)].
7 c2 != c1 # label(usol) # answer(usol).  [deny(1)].
8 x' * (x * y) = y.  [para(3(a,1),4(a,1,1)),rewrite(2(2)),flip(a)].
14 a' * b = c1.  [para(5(a,1),8(a,1,2))].
15 c2 = c1.  [para(6(a,1),8(a,1,2)),rewrite(14(4)),flip(a)].
16 $F # answer(usol).  [resolve(15,a,7,a)].
```

# Sumário

♢ Raciocínio em lógica de primeira ordem é semidecidível

♢ Regras de Instanciação Universal e Instanciação Existencial permitem reduzir inferência em LPO à inferência em lógica proposicional.

♢ Algoritmo de unificação permite encontrar o unifcador mais geral entre 1 ou mais termos/átomos.

♢ A regra de Modus Ponens Generalizado é completa para cláusulas de Horn, mas semidecidível. Para o caso restrito Datalog, o problema da consequência lógica é decidível.

♢ Encadeamento para a frente pode ser utilizado em bases de dados dedutivas, sendo completo para programas Datalog.

♢ Encadeamento para trás é utilizado em sistemas de programação em lógica, tal como o Prolog, sofrendo de problemas de inferências redundantes e possibilidade de entrar em ciclo. Tabulação evita estes problemas.

♢ Regra da resolução binária com factorização é completa para a refutação em LPO.

♢ Igualdade requer introdução de axiomas extra ou utilização de regras de inferência adicionais (e.g. paramodulação).

♢ Existem diversas estratégias para reduzir o espaço de procura em sistemas de resolução, sem sacrificar completude. Estes sistemas podem ser utilizados para demonstrar teoremas e para verificar e sintetizar software e hardware.