

# INFERÊNCIA EM LÓGICA DE PRIMEIRA ORDEM CAP 9

---

Parcialmente adaptado de  
<http://aima.eecs.berkeley.edu>

# 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

# Perspectiva Histórica

450 A.C.	Estóicos	logica proposicional, inferência (possivelmente)
322 A.C.	Aristóteles	“silogismos” (regras de inferência), quantificadores
1565	Cardano	teoria da probabilidade (logica 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 aritmetica
1960	Davis/Putnam	algoritmo “eficaz” para logica proposicional
1965	Robinson	algoritmo “eficaz” para LPO – resolução

# Métodos de Inferência para LPO

- Proposicionalização
- Resolução
- Sequentes
- Dedução Natural
- Tableaux
- Conexão de Matrizes
- Reescrita de termos

# Instanciação Universal (IU)

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

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

- Para qualquer variável  $v$  e termo básico (concreto)  $g$ .
- E.g.  $\forall x \text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$  origina:
  - $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  - $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  - $\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$
  - ...

# Instanciação Existencial (IE)

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

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

- E.g.  $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$  origina:
  - $\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$
  - 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.

# Instanciações Universal e Existencial

- 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:
  - $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  - $\text{King}(\text{John})$
  - $\text{Greedy}(\text{John})$
  - $\text{Brother}(\text{Richard}, \text{John})$
- Instanciando a frase universal de **todas as maneiras possíveis**, ficamos com:
  - $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  - $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  - $\text{King}(\text{John})$
  - $\text{Greedy}(\text{John})$
  - $\text{Brother}(\text{Richard}, \text{John})$
- A nova KB foi proposicionalizada: os símbolos proposicionais são:
  - $\text{King}(\text{John}), \text{Greedy}(\text{John}), \text{Evil}(\text{John}), \text{King}(\text{Richard}),$  etc.

# Redução à inferência proposicional

- **Proposição:** Uma frase básica é consequência da nova KB sse é consequência da KB original
- **Proposição:** 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.,  $Father(Father(Father(John)))$

# Redução à inferência proposicional

- **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 é consequência, pode não terminar se  $\alpha$  não é consequência
- **Teorema:** Turing (1936), Church (1936), **consequência em LPO é semidecidível**
  - Existem algoritmos que respondem “sim” a todas as frases que são consequência lógica, mas não existem algoritmos que também respondam “não” a todas as frases que não são consequência lógica.

# Problemas com a proposicionalização

- A proposicionalização pode gerar inúmeras frases irrelevantes.
  - E.g., de
    - $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
    - $\text{King}(\text{John})$
    - $\forall y \text{ Greedy}(y)$
    - $\text{Brother}(\text{Richard}, \text{John})$
  - é obvio que  $\text{Evil}(\text{John})$ , mas a proposicionalização produz muitos factos, por exemplo  $\text{Greedy}(\text{Richard})$ , que são irrelevantes
- Com  $p$  predicados  $k$ -ários e  $n$  constantes, existem  $p \cdot n^k$  instanciações!

# Unificação

- Podemos imediatamente obter a conclusão se conseguirmos encontrar uma substituição tal que  $King(x)$  e  $Greedy(x)$  concordem com  $King(John)$  e  $Greedy(y)$ 
  - $\theta = \{x/John, y/John\}$  funciona
- $Unificar(\alpha, \beta) = \theta$  se  $\alpha\theta = \beta\theta$

$\alpha$	$\beta$	$\theta$
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	
$Knows(John, x)$	$Knows(x, OJ)$	

# Unificação

- Podemos imediatamente obter a conclusão se conseguirmos encontrar uma substituição tal que  $King(x)$  e  $Greedy(x)$  concordem com  $King(John)$  e  $Greedy(y)$ 
  - $\theta = \{x/John, y/John\}$  funciona
- $Unificar(\alpha, \beta) = \theta$  se  $\alpha\theta = \beta\theta$

$\alpha$	$\beta$	$\theta$
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	<i>fail</i>

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

# Unificação

- Para unificar  $Knows(John,x)$  e  $Knows(y,z)$ :
  - $\theta = \{y/John, x/z\}$  ou  $\theta = \{y/John, x/John, z/John\}$
- A primeira substituição é **mais geral** do que a segunda.
- Existe sempre um **unificador mais geral** (most general unifier – **MGU**) que é único a menos de renomeação de variáveis.
  - $MGU = \{y/John, x/z\}$

# Algoritmo de Unificação

```
function UNIFY( $x, y, \theta$ ) returns a substitution to make  $x$  and  $y$  identical
  inputs:  $x$ , a variable, constant, list, or compound
            $y$ , a variable, constant, list, or compound
            $\theta$ , the substitution built up so far

  if  $\theta = \text{failure}$  then return failure
  else if  $x = y$  then return  $\theta$ 
  else if VARIABLE?( $x$ ) then return UNIFY-VAR( $x, y, \theta$ )
  else if VARIABLE?( $y$ ) then return UNIFY-VAR( $y, x, \theta$ )
  else if COMPOUND?( $x$ ) and COMPOUND?( $y$ ) then
    return UNIFY(ARGS[ $x$ ], ARGS[ $y$ ], UNIFY(OP[ $x$ ], OP[ $y$ ],  $\theta$ ))
  else if LIST?( $x$ ) and LIST?( $y$ ) then
    return UNIFY(REST[ $x$ ], REST[ $y$ ], UNIFY(FIRST[ $x$ ], FIRST[ $y$ ],  $\theta$ ))
  else return failure
```

# Algoritmo de Unificação (cont.)

```
function UNIFY-VAR(var, x,  $\theta$ ) returns a substitution
  inputs: var, a variable
            x, any expression
             $\theta$ , the substitution built up so far

  if  $\{var/val\} \in \theta$  then return UNIFY(val, x,  $\theta$ )
  else if  $\{x/val\} \in \theta$  then return UNIFY(var, val,  $\theta$ )
  else if OCCUR-CHECK?(var, x) then return failure
  else return add  $\{var/x\}$  to  $\theta$ 
```

# Modus Ponens Generalizado (MPG)

$$\frac{p'_1, \dots, p'_n, \quad p_1 \wedge \dots \wedge p_n \Rightarrow q}{q\theta} \quad \text{onde } p'_i\theta = p_i\theta \text{ para todo } i$$

- $p'_1$  é *King(John)*                       $p_1$  é *King(x)*
- $p'_2$  é *Greedy(y)*                       $p_2$  é *Greedy(x)*
- $q$  é *Evil(x)*
- $\theta$  é  $\{x/\text{John}, y/\text{John}\}$                $q\theta$  é *Evil(John)*
- MPG utilizado com KB de **cláusulas definidas** (**exatamente** um literal positivo).
- Todas as variáveis estão implicitamente quantificadas universalmente.

# Exemplo de Base de Conhecimento

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

# Exemplo de Base de Conhecimento

- ... 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.,  $\exists x Owns(Nono,x) \wedge Missile(x)$ :  
 $Owns(Nono,M1) \text{ and } Missile(M1)$  (por IE)
- ... todos os seus mísseis foram-lhe vendidos pelo Coronel West  
 $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
- Mísseis são armas:  
 $Missile(x) \Rightarrow Weapon(x)$
- Um inimigo da America é “hostil”:  
 $Enemy(x,America) \Rightarrow Hostile(x)$
- West, é Americano ...  
 $American(West)$
- O país Nono, é inimigo da América...  
 $Enemy(Nono,America)$
- Recordar que todas as variáveis estão implicitamente quantificadas universalmente

# 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
       $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-APART}(r)$ 
      for each  $\theta$  such that  $(p_1 \wedge \dots \wedge p_n)\theta = (p'_1 \wedge \dots \wedge p'_n)\theta$ 
        for some  $p'_1, \dots, p'_n$  in  $KB$ 
           $q' \leftarrow \text{SUBST}(\theta, q)$ 
          if  $q'$  is not a renaming of a sentence already in  $KB$  or new then do
            add  $q'$  to new
             $\phi \leftarrow \text{UNIFY}(q', \alpha)$ 
            if  $\phi$  is not fail then return  $\phi$ 
    add new to  $KB$ 
  return false
```

# Prova por encadeamento para a frente

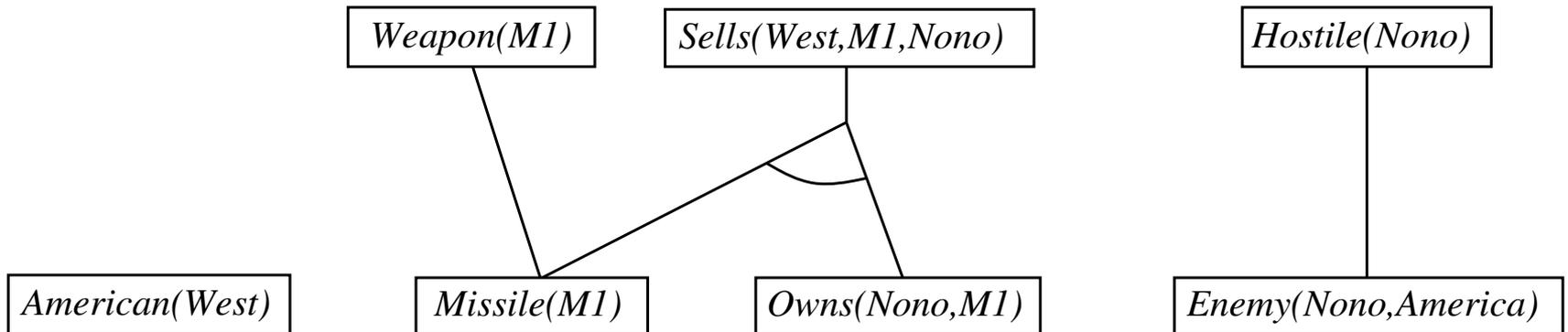
*American(West)*

*Missile(M1)*

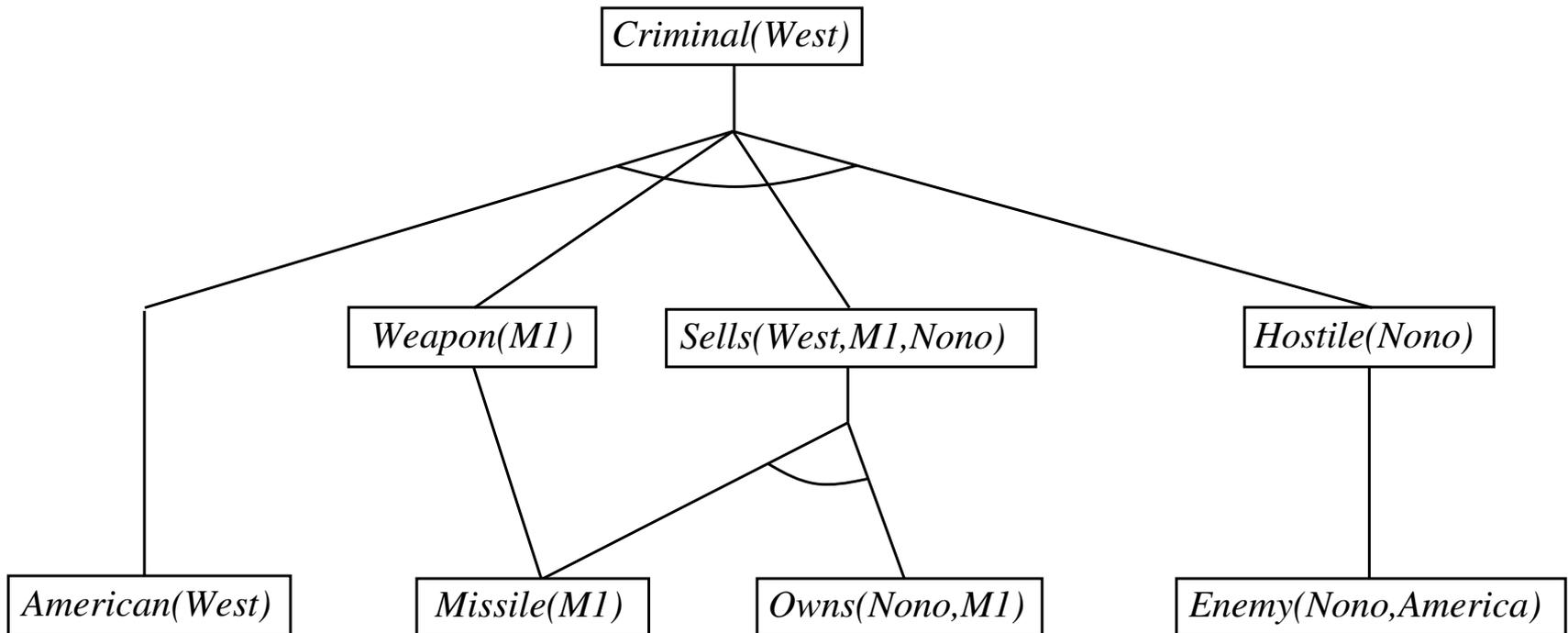
*Owns(Nono,M1)*

*Enemy(Nono,America)*

# Prova por encadeamento para a frente



# Prova por encadeamento para a frente



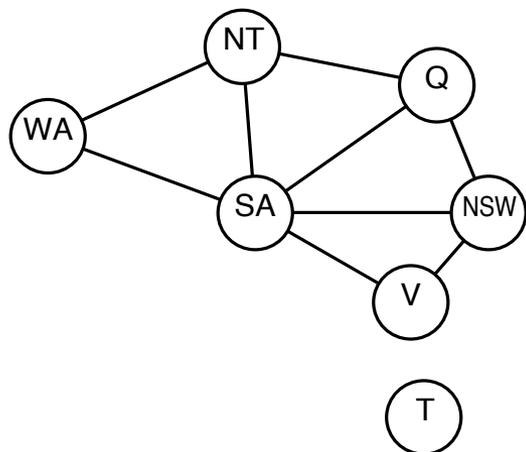
# Propriedades do encadeamento para a frente

- Correcto para cláusulas definidas de primeira-ordem
  - Demonstração semelhante à do caso proposicional
- **Datalog** = cláusulas definidas de primeira-ordem + **sem funções** (e.g. 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 do encadeamento para a frente

- Observação simples: não é necessário utilizar uma regra na iteração  $k$  se a premissa não foi adicionada na iteração  $k-1$ 
  - Estratégia: 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.
  - E.g., a consulta `Missile(x)` devolve `Missile(M1)`
- 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


$$\text{Diff}(wa,nt) \wedge \text{Diff}(wa,sa) \wedge \text{Diff}(nt,q) \wedge$$
$$\text{Diff}(nt,sa) \wedge \text{Diff}(q,nsw) \wedge \text{Diff}(q,sa) \wedge$$
$$\text{Diff}(nsw,v) \wedge \text{Diff}(nsw,sa) \wedge \text{Diff}(v,sa) \Rightarrow$$
$$\text{Colorable}()$$
$$\text{Diff}(\text{Red},\text{Blue}) \quad \text{Diff}(\text{Red},\text{Green})$$
$$\text{Diff}(\text{Green},\text{Red}) \quad \text{Diff}(\text{Green},\text{Blue})$$
$$\text{Diff}(\text{Blue},\text{Red}) \quad \text{Diff}(\text{Blue},\text{Green})$$

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

# Encadeamento para trás

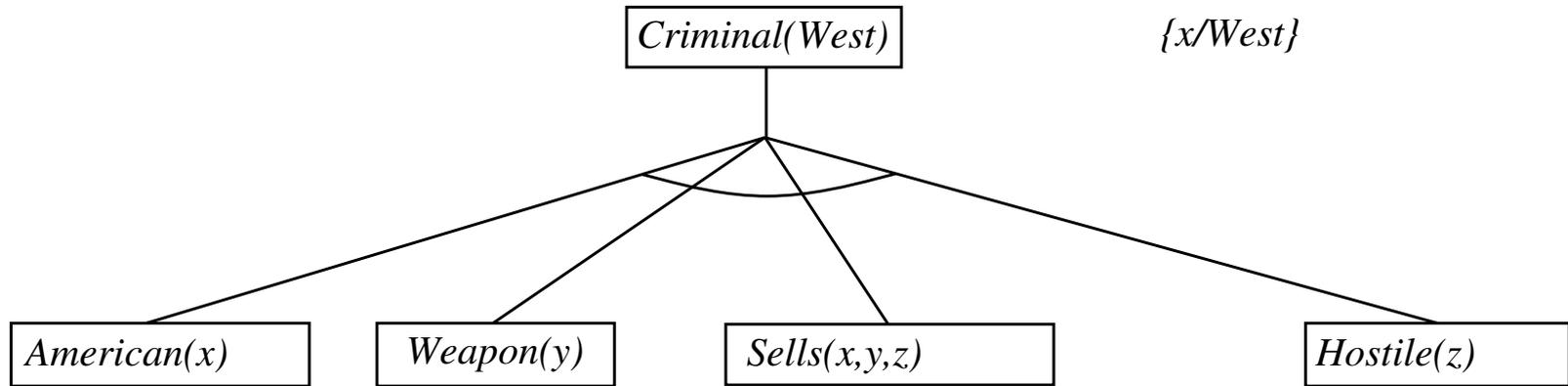
```
function FOL-BC-ASK(KB, goals,  $\theta$ ) returns a set of substitutions
  inputs: KB, a knowledge base
            goals, a list of conjuncts forming a query ( $\theta$  already applied)
             $\theta$ , the current substitution, initially the empty substitution { }
  local variables: answers,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

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

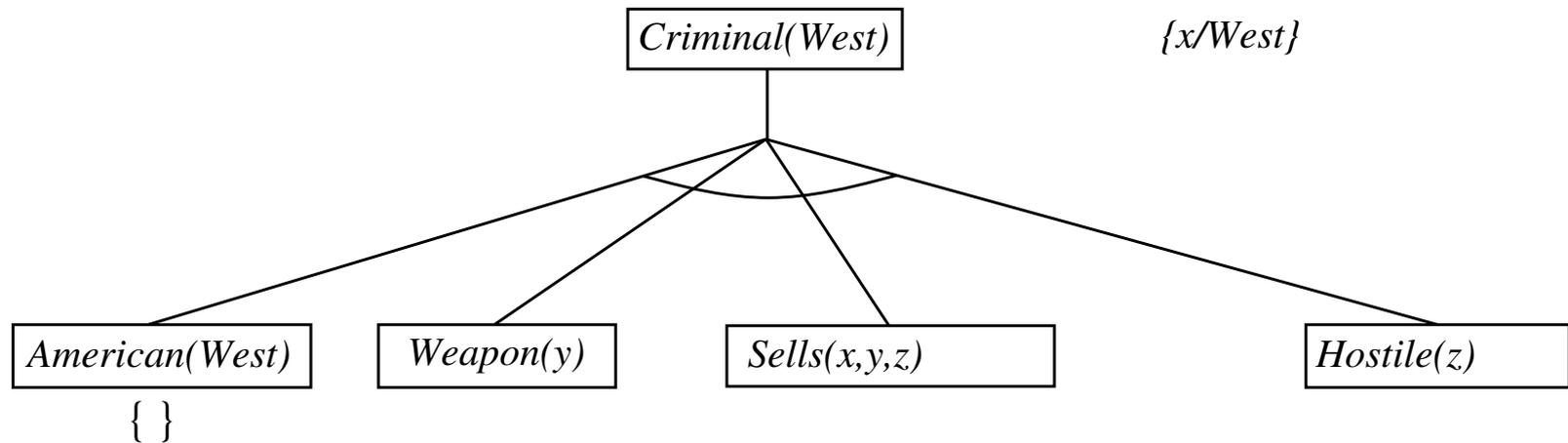
# Prova por encadeamento para trás

*Criminal(West)*

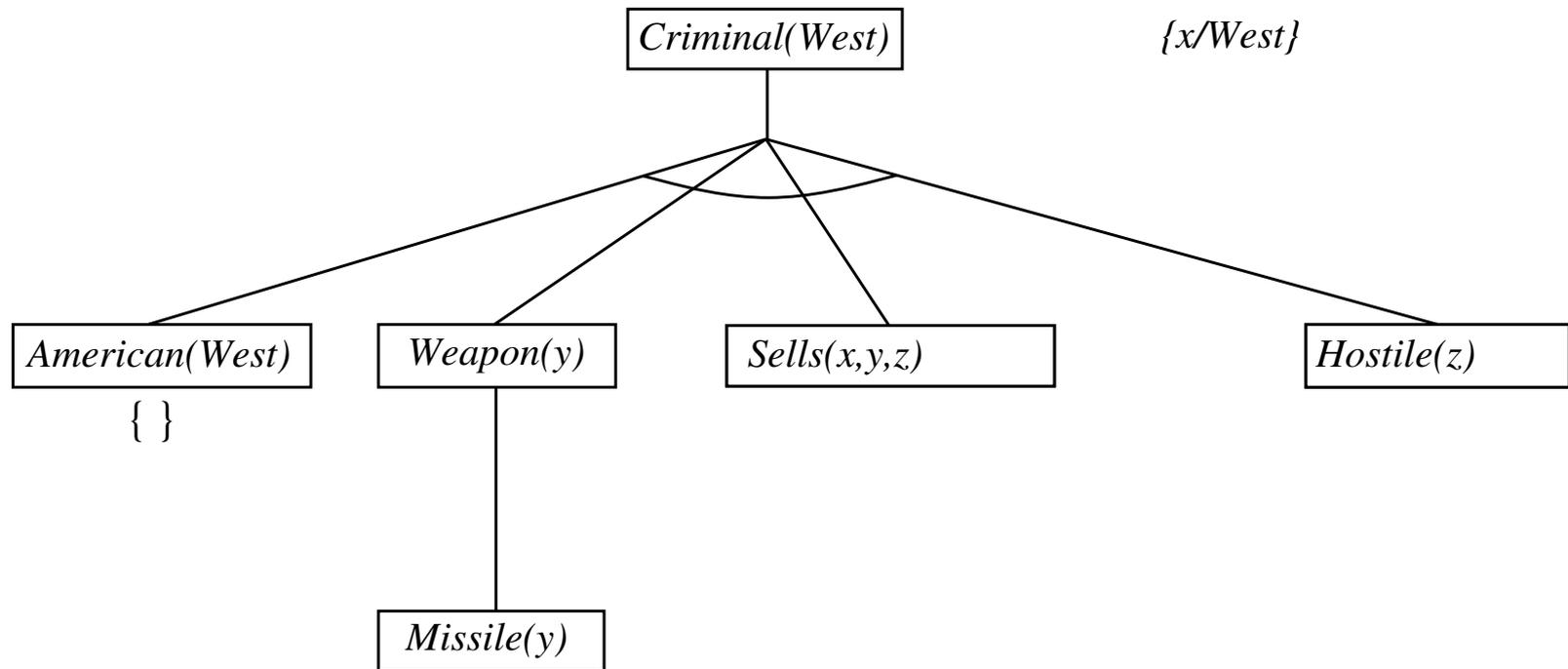
# Prova por encadeamento para trás



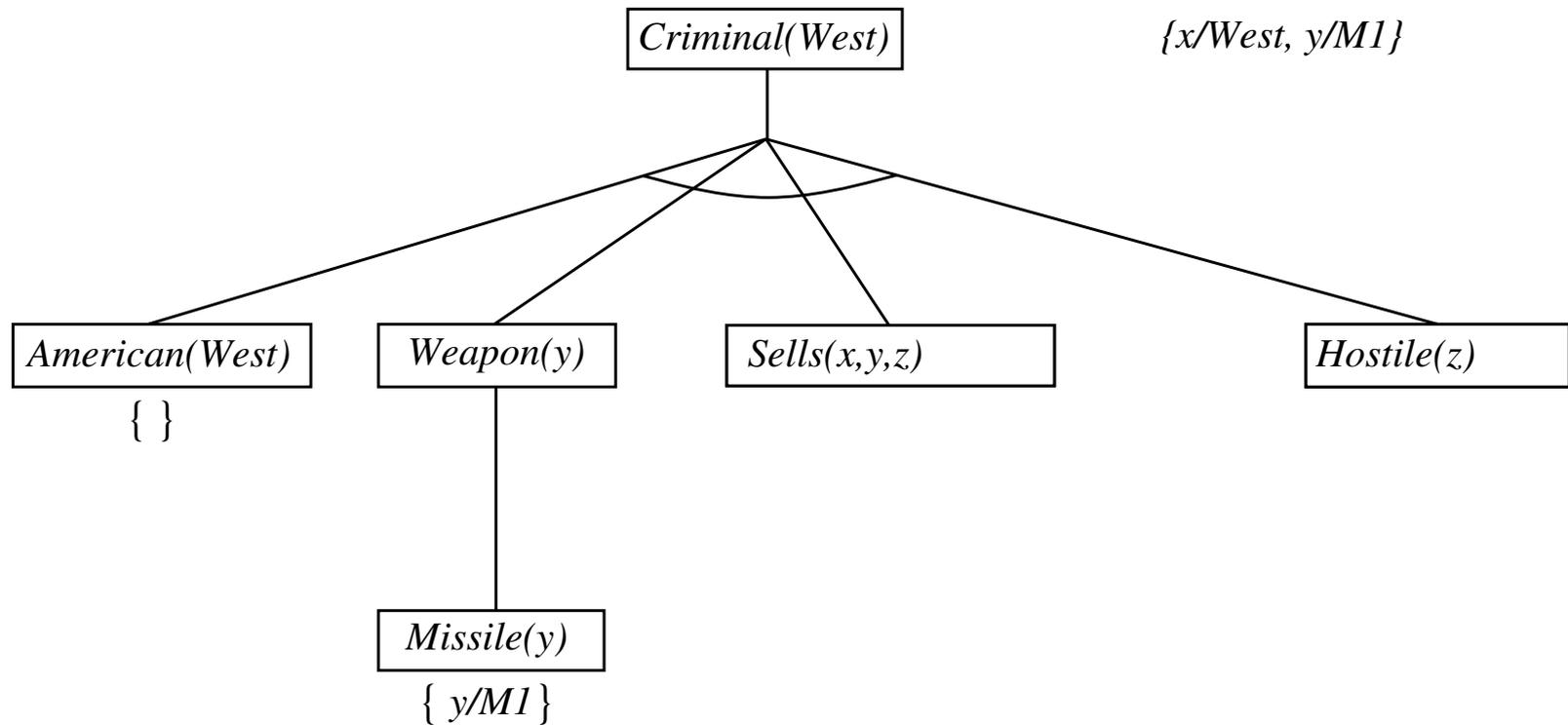
# Prova por encadeamento para trás



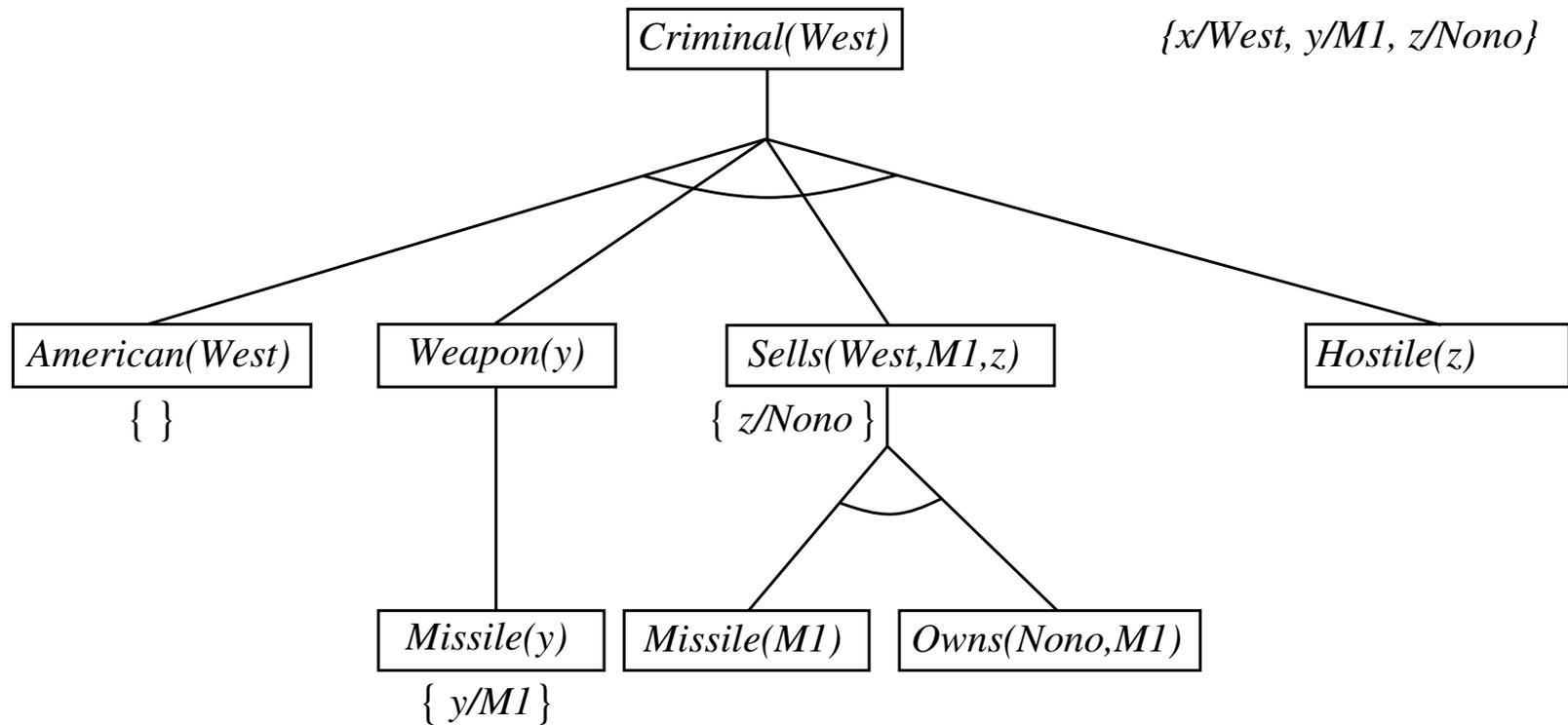
# Prova por encadeamento para trás



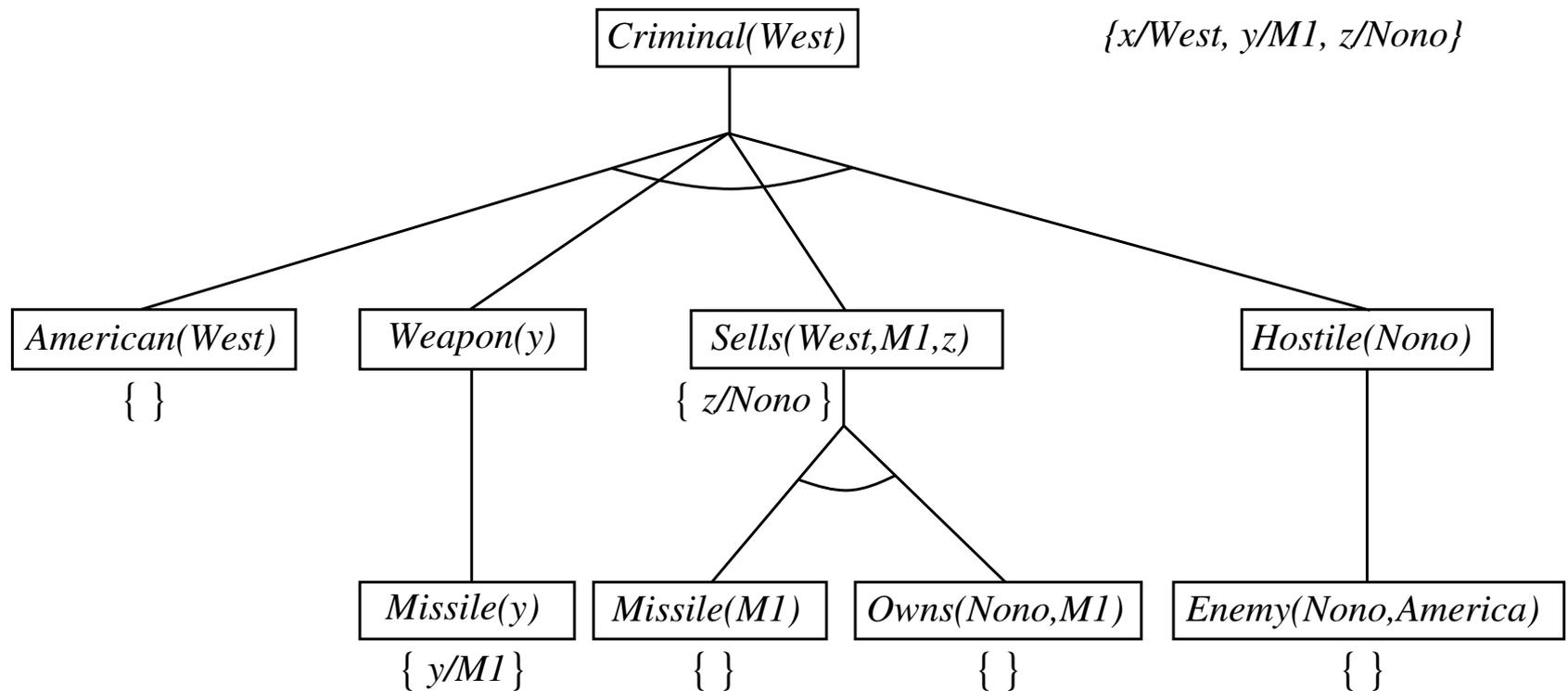
# Prova por encadeamento para trás



# Prova por encadeamento para trás



# Prova por encadeamento para trás



# Propriedades do encadeamento para trás

- Pesquisa da prova recursivamente em profundidade primeiro: espaço linear no tamanho da prova
- Incompleto devido a ciclos infinitos
  - Verificação do objectivo corrente com todos os outros na pilha
- Ineficiente devido às subconsultas repetidas (de sucesso e de falha)
  - Memorização dos resultados anteriores (espaço extra!)
- Amplamente utilizado (sem melhoramentos!) na programação em lógica

# Programação em Lógica

## Programação em Lógica

- Identificar problema
- Coligir Informação
- <pausa para café>
- Codificar informação na KB
- Representar instância com factos
- Efectuar consultas
- Encontrar factos errados

## Programação Procedimental

- Identificar problema
- Coligir informação
- Descobrir solução
- Programar solução
- Representar instância com dados
- Aplicar programa aos dados
- Depurar erros procedimentais

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

# Programação em Lógica: Prolog

- Essência: encadeamento para trás com cláusulas de Horn
- Programa = conjunto de cláusulas = `head:-literal1,...,literaln.`  
`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`
- Pressuposto do Mundo Fechado (“negação por falha”)
  - e.g., dado `alive(X):-not dead(X).`
  - `alive(joe)` sucede se `dead(joe)` falha

# Exemplo 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).
|
?- weapon(Y1), sells(west,Y1,Z1), hostile(Z1).
|
?- missile(Y1), sells(west,Y1,Z1), hostile(Z1).
|
?- sells(west,m1,Z1), hostile(Z1).
|
?- missile(m1), owns(nono,m1), hostile(nono).
|
?- owns(nono,m1), hostile(nono).
|
?- hostile(nono).
|
?- enemy(nono,america).
|
?-
```

Who = west

Y1 = m1

Z1 = nono

# Resolução binária

- **Forma Normal Conjuntiva (FNC – universal)**
  - **Conjunção de disjunções de literais com variáveis quantificadas universalmente**
- **Regra de inferência Resolução:**

$$\frac{l_1 \vee \dots \vee l_k, \quad m_1 \vee \dots \vee m_n}{(l_1 \vee \dots \vee l_{i-1} \vee l_{i+1} \vee \dots \vee l_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n) \theta}$$

em que  $Unificar(l_i, \neg m_j) = \theta$ . E.g.,

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

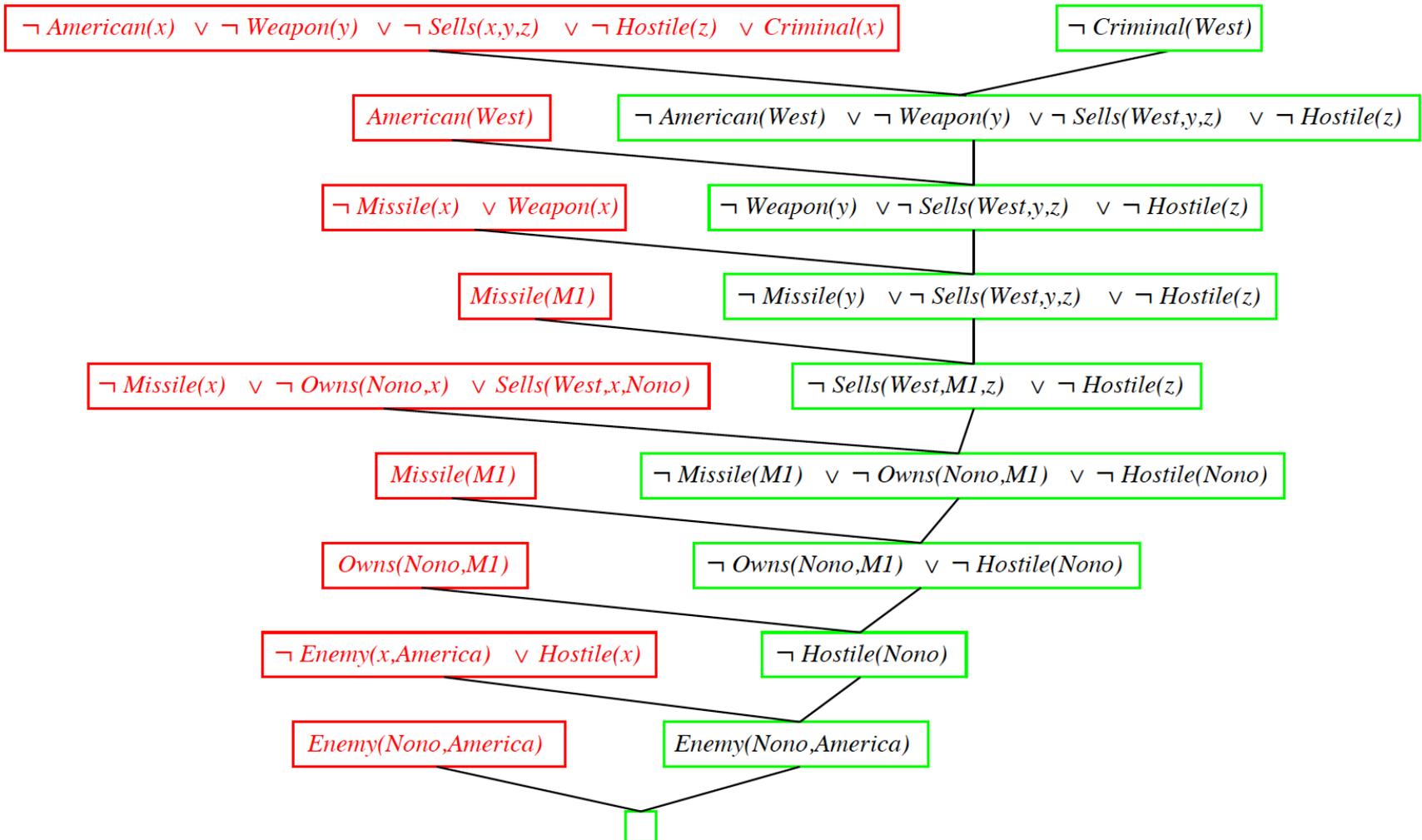
em que  $\theta = \{x/Ken\}$ .

- A regra da resolução binária, por si só, não é completa.

# Factorização

- Seja  $C'$  um subconjunto de literais com o mesmo sinal de uma cláusula  $C$  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.  $P(x1, f(x1), f(x1))$  factor de 1.
  4.  $\neg P(A, z1, z1)$  factor de 2:
  5.  $\square$  resolvente de 3. e 4.
- Aplicar resolução a  $\text{CNF}(\text{KB} \wedge \neg\alpha)$
- **Resolução Binária + Factorização  $\Rightarrow$  completo para LPO**

# Prova por resolução: cláusulas definidas



# Conversão para a FNC

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

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

1. Eliminar bicondicionais e implicações

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

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

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

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

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

# Conversão para a FNC

3. Standardizar variáveis: cada quantificador deve usar uma diferente.

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

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.

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

5. Remover quantificadores universais.

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

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

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

# A Curiosidade matou o Gato?

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

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

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

$$\forall x [\exists z (\textit{Animal}(z) \wedge \textit{Kills}(x, z))] \Rightarrow [\neg \exists y \textit{Loves}(y, x)]$$

- Jack ama todos os animais.

$$\forall x \textit{Animal}(x) \Rightarrow \textit{Loves}(\textit{Jack}, x)$$

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

$$\textit{Kills}(\textit{Jack}, \textit{Tuna}) \vee \textit{Kills}(\textit{Curiosity}, \textit{Tuna})$$

- Todos os gatos são animais

$$\forall x \textit{Cat}(x) \Rightarrow \textit{Animal}(x)$$

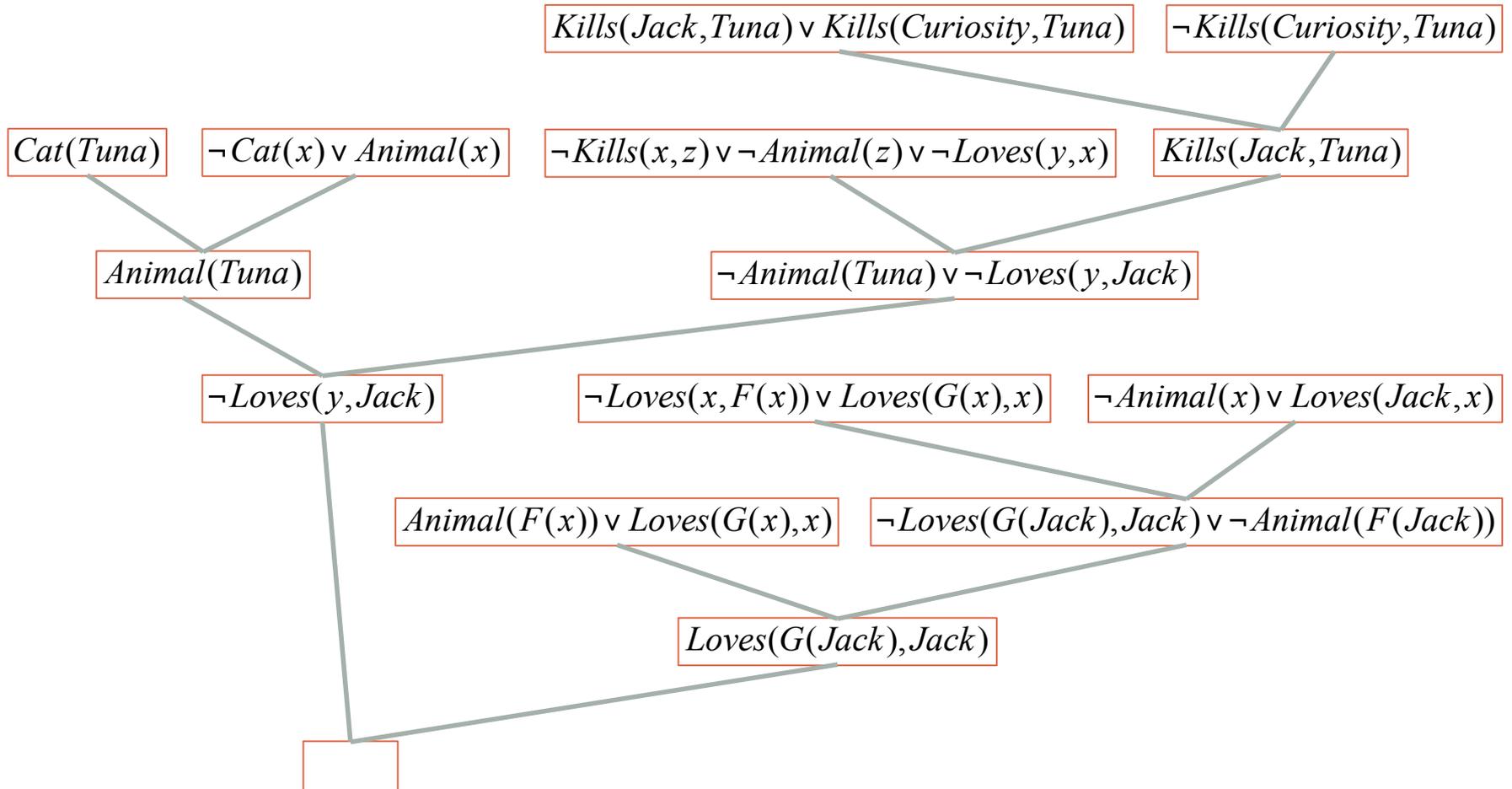
- A curiosidade matou o gato?

$$\neg \textit{Kills}(\textit{Curiosity}, \textit{Tuna})$$

# Conversão para a FNC

- Toda a gente que ama todos os animais é amada por alguém.  
 $Animal(F(x)) \vee Loves(G(x), x) \quad \neg Loves(x, F(x)) \vee Loves(G(x), x)$
- Qualquer pessoa que mata um animal não é amada por ninguém.  
 $\neg Kills(x, z) \vee \neg Animal(z) \vee \neg Loves(y, x)$
- Jack ama todos os animais.  
 $\neg Animal(x) \vee Loves(Jack, x)$
- O Jack ou a Curiosidade mataram o gato, que se chama Tuna.  
 $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
- Todos os gatos são animais  
 $\neg Cat(x) \vee Animal(x)$
- A curiosidade matou o gato?  
 $\neg Kills(Curiosity, Tuna)$

# Prova que a Curiosidade matou o Gato



# Tratamento da igualdade

- A igualdade introduz problemas adicionais no algoritmo de inferência. Existem duas grandes classes de abordagens 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  $1 \leq i \leq n$

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

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

$$\forall x_1, \dots, x_i, \dots, x_n \forall y x_i=y \Rightarrow (f(x_1, \dots, x_i, \dots, x_n) \equiv f(x_1, \dots, y, \dots, x_n))$$

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

# Demodução

- Para quaisquer termos  $x, y$  e  $z$  tal que  $\text{UNIFY}(x, z) = \theta$  e  $m_i$  é um literal contendo  $z$ :

$$\frac{x = y, \quad m_1 \vee \cdots \vee m_n [z]}{m_1 \vee \cdots \vee m_n [\text{SUBST}(\theta, y)]}$$

- A regras da **Demodulação** é incompleta.
- Exemplo:

$$\frac{0 + z = z, \quad P(0 + (0 + 2)) \vee Q(3)}{P(0 + 2) \vee Q(3)}$$

# Paramodulação

- Para quaisquer termos  $x, y$  e  $z$  tal que  $\text{UNIFY}(x, z) = \theta$

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

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

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

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  $P$  seja um antecessor de  $Q$  na árvore de prova. Método completo.

# 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 unificador 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.

# Sumário

- 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.