Inferência em lógica de primeira ordem

Aula 7 - Capítulo 9

Fundamentos da IA

Mestrado FEI 2005

Redução à inferência proposicional

- A inferência de primeira ordem pode ser realizada convertendo-se a base de conhecimento para a lógica proposicional e utilizando-se a inferência proposicional.

Instanciação Universal (UI)

- Toda instância de uma sentença universalmente quantificada é consequência semantica desta:

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

Para qqr variável *v* e termo *g*

- E.g., $\forall$x *King*(*x*) $\wedge$ *Greedy*(*x*) $\Rightarrow$ *Evil*(*x*) leva a:

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*))

.

.

.

Instanciação Existencial (EI)

- Para qqr sentença α, variável *v*, e constante *k* que não aparece em nenhum outro lugar da base de conhecimento:

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

- E.g., $\exists x\ Crown(x) \wedge OnHead(x, John)$ leva:

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

C_1 é um novo símbolo de constante, chamado constante de Skolem

Redução à inferência proposicional

Suponha uma base de conhecimento KB contendo o seguinte:

$\forall x$ King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x)

King(John)

Greedy(John)

Brother(Richard,John)

- Instanciando a sentença universalmente quantificada, em todos os modos possíveis, temos:

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 BC está "proposicionalizada": contendo símbolos proposicionais:

King(John), Greedy(John), Evil(John), King(Richard), etc.

Redução à inferência proposicional

- Toda BC de 1a ordem pode ser proposicionalizada preservando a consequência lógica
  - (Uma sentença é obtida pela nova BC sse for consequência lógica da BC original)
- Processo: proposicionalizar a BC e aplicar resolução proposicional;
  - Concepção de inferência automática de 1a ordem até 1960!
- Problema: com símbolos funcionais, há infinitos termos proposicionais:
  - e.g., *Father*(*Father*(*Father*(*John*)))

Profundidade de aninhamento

Redução contd.

Teorema de Herbrand (1930): se uma sentença α é consequência lógica de uma base de conhecimento de 1a ordem, então é também consequência de um subconjunto finito da base de conhecimento proposicionalizada

Resultado: para a profundidade de $n = 0$ à ∞:

criar uma base proposicional BC com profundidade n,

verificar consequência lógica α com relação à BC

Só funciona se α é consequência lógica, entra em loop caso contrário

Teorema de Church-Turing (1936): a Consequência lógica em lógica de primeira ordem é semidecidível (i.e. Existem algoritmos que dizem YES para todas as sentenças que são consequência semântica de uma BC, mas NENHUM algoritmo pode dizer FALHA para todas as sentenças que não são consequência da base de conhecimento).

Problemas com a propositionalização

- Proposicionalização gera muitas sentenças irrelevantes.

- E.g.,

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

- a dedução de *Evil*(*John*) deveria ser direta, porém propositionalização produz fatos irrelevantes como *Greedy*(*Richard*);

Unificação

- Obtém-se a inferência desejada imediatamente se existir uma substituição θ tal que *King(x)* e *Greedy(x)* se resolvam com *King(John)* e *Greedy(y)*

  θ = {x/John,y/John} é o que procuramos.

  – Unify(α,β) = θ if αθ = βθ

p	q	θ
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

p	q	θ
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

p	q	θ
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

p	q	θ
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

p	q	θ
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}

Unificação

- Para unificar *Knows(John,x)* e *Knows(y,z)*,

  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- O primeiro unificador é mais geral do que o segundo

- se duas sentenças são unificáveis, há um e somente um unificador mais geral (MGU) -- *up to renaming of variables.*

MGU = { y/John, x/z }

O algoritmo de unificação

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

   if θ = failure then return failure
   else if x = y then return θ
   else if VARIABLE?(x) then return UNIFY-VAR(x, y, θ)
   else if VARIABLE?(y) then return UNIFY-VAR(y, x, θ)
   else if COMPOUND?(x) and COMPOUND?(y) then
       return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
   else if LIST?(x) and LIST?(y) then
       return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
   else return failure
```

O algoritmo de unificação

function UNIFY-VAR(var, x, θ) **returns** a substitution
  inputs: var, a variable
    x, any expression
    θ, the substitution built up so far

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

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' is *King*(*John*)	p_1 is *King*(*x*)
p_2' is *Greedy*(*y*)	p_2 is *Greedy*(*x*)
θ is {x/John,y/John}	q is *Evil*(*x*)
q θ is *Evil*(*John*)	

- processo é utilizar MPG com bases de dados compostas somente com cláusulas de Horn.
- Todas as variáveis são universalmente quantificadas.

Exemplo

- "*The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American*."

- Provar automaticamente que Col. West é um criminoso.

Exemplo

... it is a crime for an American to sell weapons to hostile nations:

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

… all of its missiles were sold to it by Colonel West

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Forward chaining algorithm

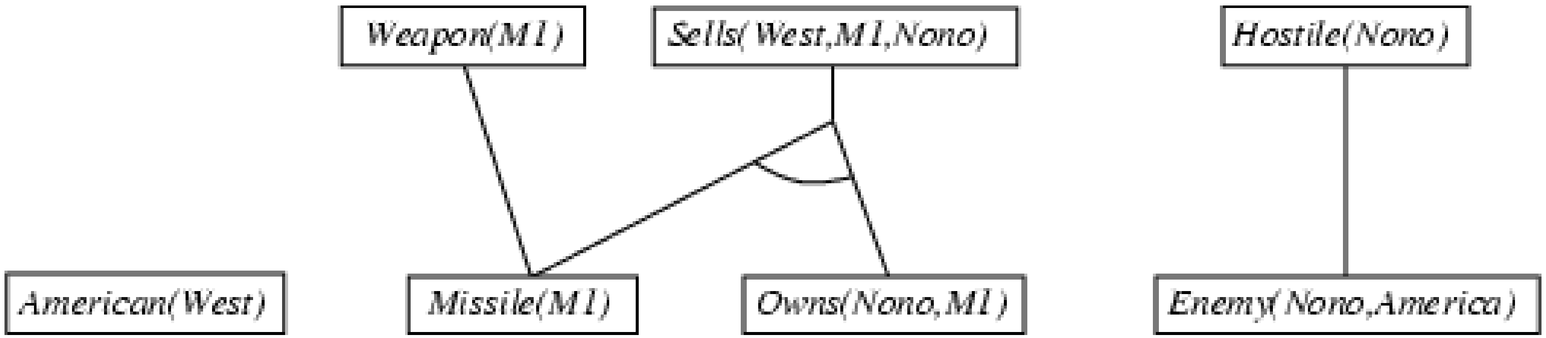
function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

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 θ 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(θ, 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', α)
      if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

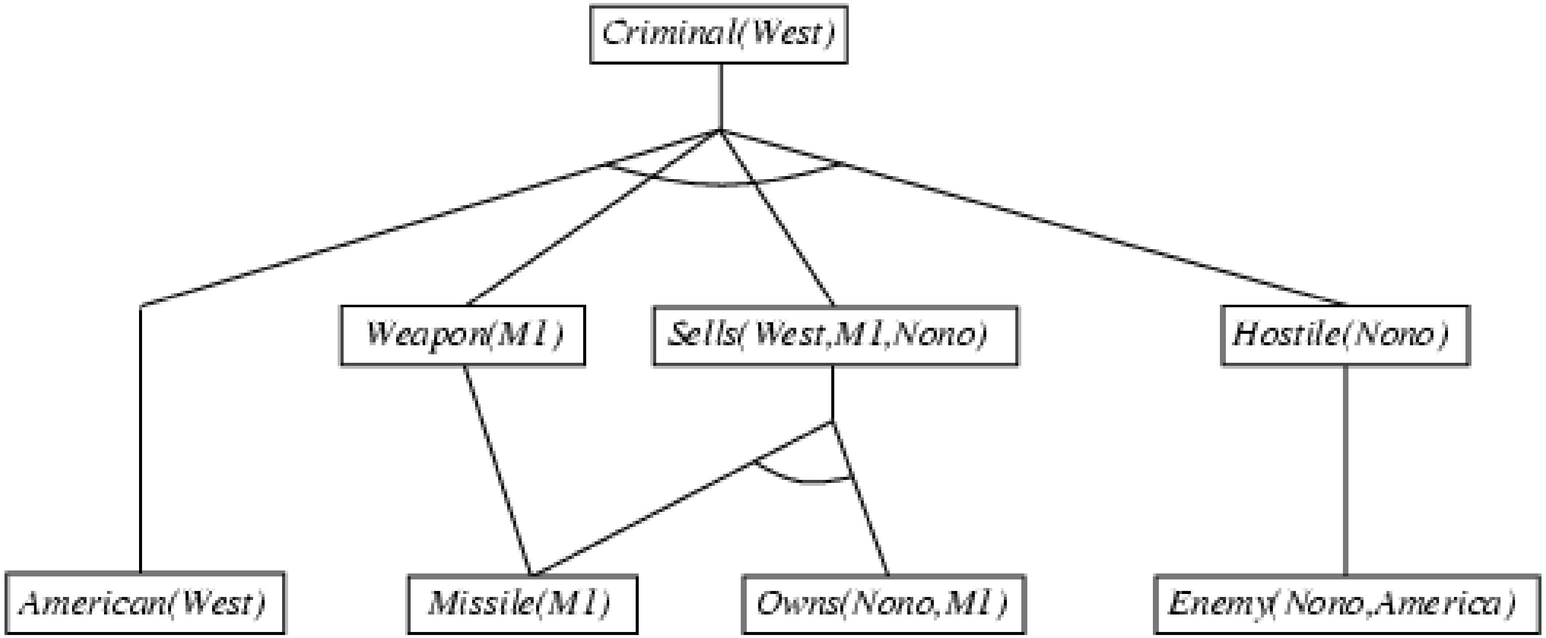
Forward chaining proof

$\boxed{American(West)}$ $\boxed{Missile(M1)}$ $\boxed{Owns(Nono,M1)}$ $\boxed{Enemy(Nono,America)}$

Forward chaining proof



Forward chaining proof



Propriedades do encadeamento para frente

- É correto e completo para cláusulas de Horn (sem símbolos funcionais -- cláusulas definidas);
- Pode não terminar se α não for consequência lógica.
  - E isso é definitivo (Teorema de Church-Turing)!

Backward chaining algorithm

- É chamado com uma lista de objetivos contendo inicialmente um único elemento, a consulta original, e retorna o conjunto de todas as substituições que satisfazem à consulta.

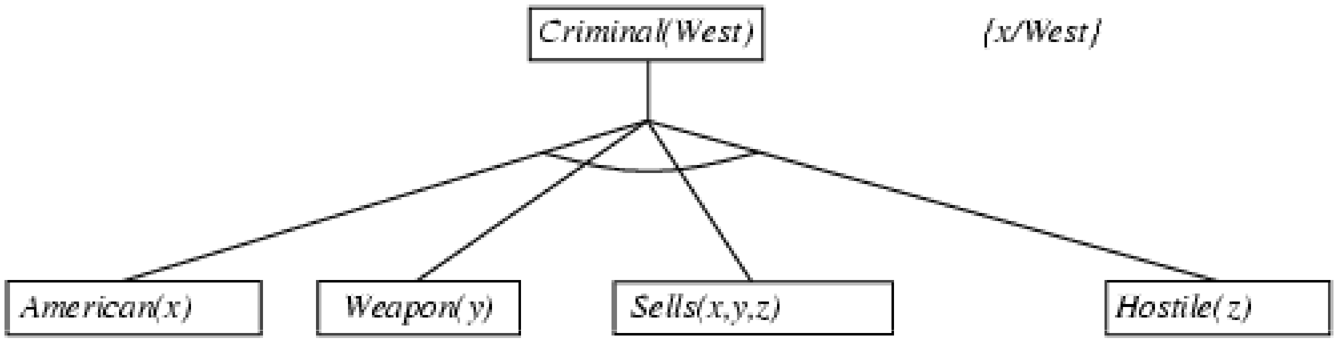
Backward chaining algorithm

- O algoritmo recebe o primeiro objetivo e encontra toda cláusula da base de conhecimento cujo literal positivo (ou cabeça) se unifica com objetivo. Cada clausula desse tipo cria uma nova chamada recursiva, na qual a premissa (ou corpo) da cláusula é adicionada à pilha de objetivos.

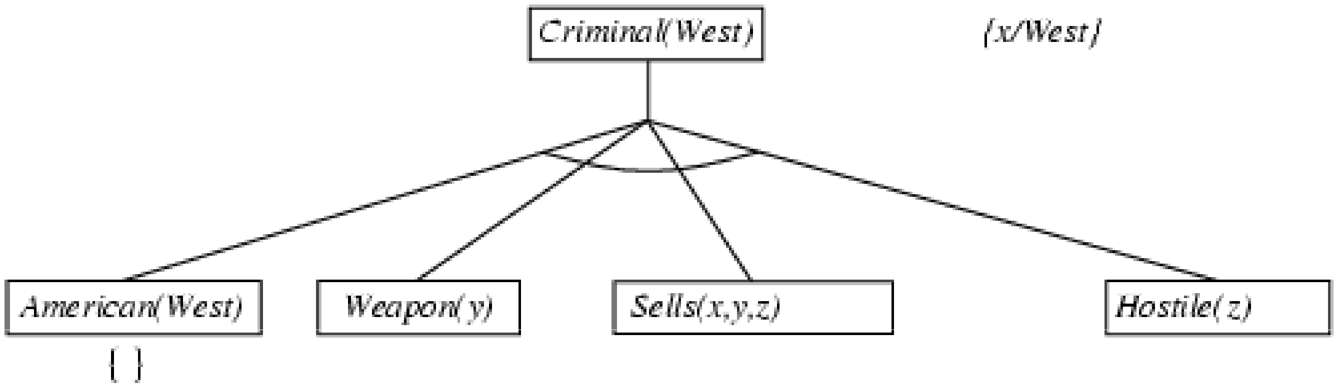
Backward chaining example

$\boxed{Criminal(West)}$

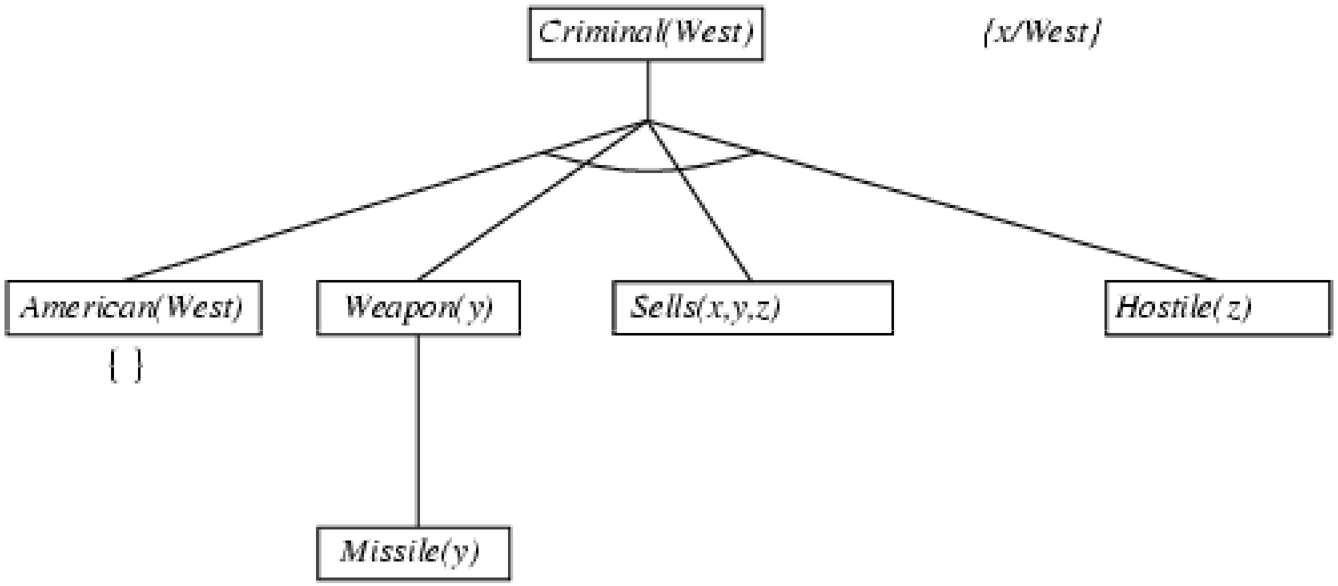
Backward chaining example



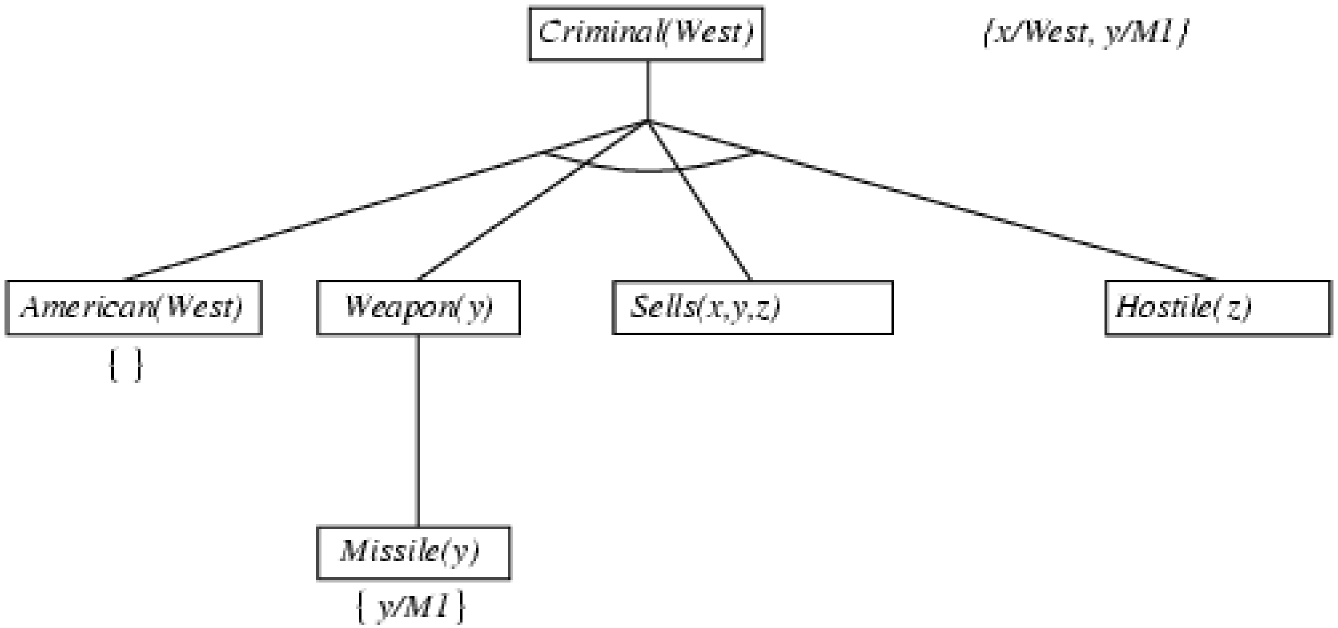
Backward chaining example



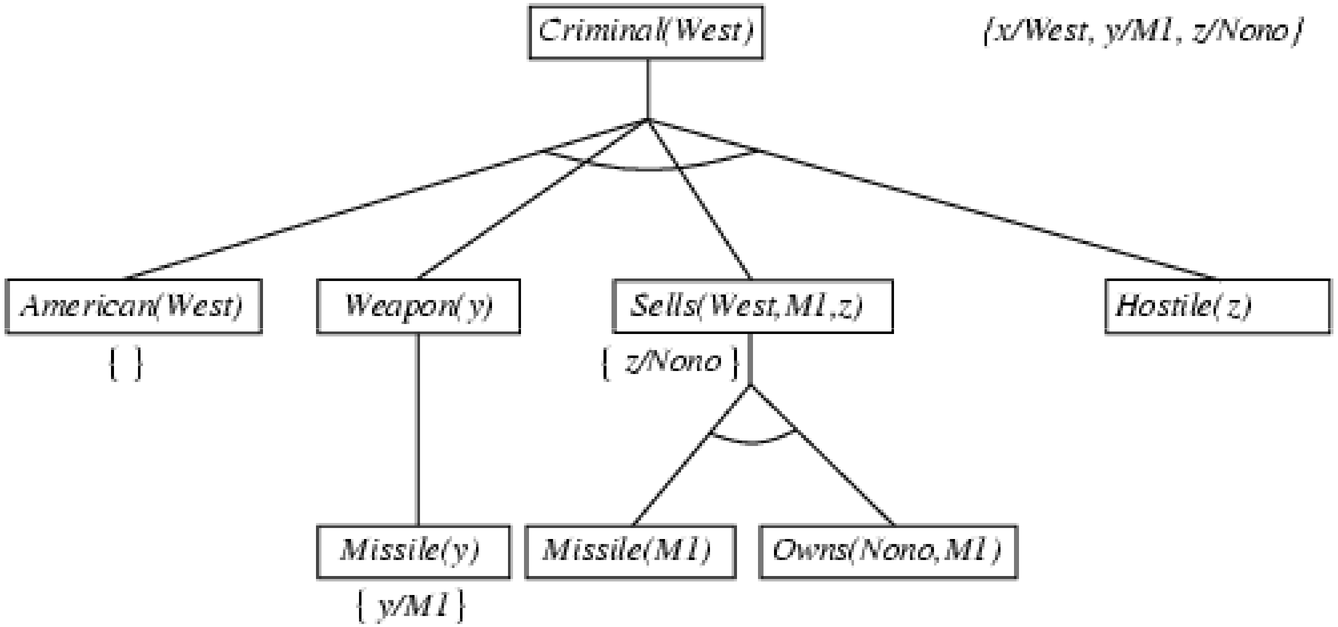
Backward chaining example



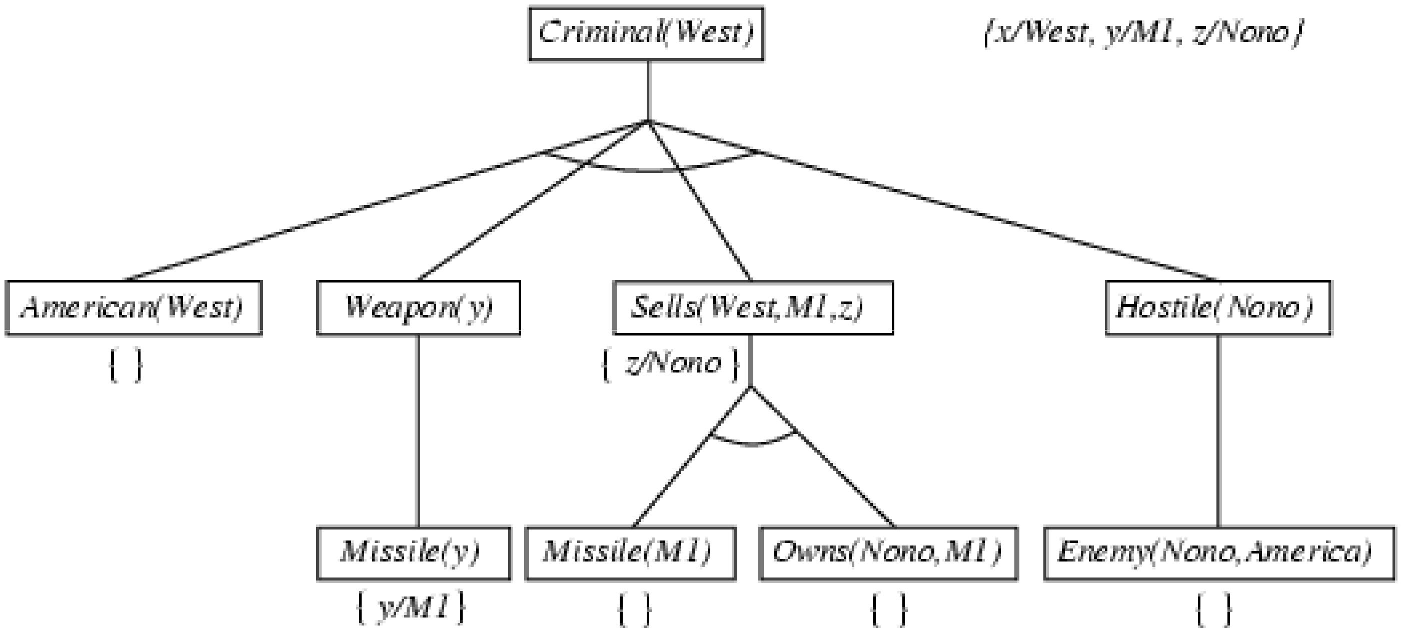
Backward chaining example



Backward chaining example



Backward chaining example



Propriedades do backward chaining

- O espaço é linear com relação ao tamanho da prova.
- Incompleta devido a loops infinitos (busca em profundidade)

$\Rightarrow$ Pode ser consertado verificando se cada novo objetivo já foi verificado antes.

- Ineficiente, pois subgoals podem ser resolvidos várias vezes

$\Rightarrow$ Pode ser consertado armazenando resultados anteriores (às expensas de memória)

- Método utilizado em programação em lógica

Logic programming: Prolog

- Algorítmo = Lógica + Controle

- Base: backward chaining com Horn clauses + parâmetros de controle
  Extensamente utilizado na Europe, Japan (base para 5th Generation project)

- Programa = conjunto de cláusulas
  - `head :- literal`$_1$`, … literal`$_n$.
  - `criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- Busca em profundidade;
- Predicados pré-definidos
  - aritmética: e.g., `X is Y*Z+3`
  - `com efeitos colaterais:`(e.g., input and output predicates, assert/retract predicates)
- Hipótese de mundo fechado ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` é verdade se `dead(joe)` falha.

Prolog

- Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- query: `append(A,B,[1,2]) ?`

- answers:

```
A=[]     B=[1,2]
A=[1]    B=[2]
A=[1,2]  B=[]
```

Resolução

- Versão de primeira ordem:

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

  onde `Unifica`(ℓ_i, $\neg m_j$) = θ.

- As duas clausulas são separadamente padronizadas, assim não possuem variáveis iguais
- As cláusulas devem estar em forma normal conjuntiva (CNF).
- Exemplo,

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

  with θ = {x/Ken}

- Resolução à CNF(KB $\wedge$ $\neg\alpha$) é completo lógica de primeira ordem.

Exemplo: Prova por resolução