

Lógica de Primeira Ordem: Forward Chaining

- renomeia variáveis e compõe substituições.
- Composição de substituições:

$$SUBST(COMPOSE(\theta_1, \theta_2), p) = SUBST(\theta_2, SUBST(\theta_1, p))$$

Lógica de Primeira Ordem: Forward Chaining

procedure FORWARD-CHAIN(KB,p)

if há uma sent em KB que renomeia p **then return**

adiciona p a KB

for each $(p_1 \wedge \dots \wedge p_n) \rightarrow q$ em KB tal que para algum i,

$UNIFY(p_i, p) = \theta$ sucede **do**

FIND-AND-INFER(KB, $[p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_n], q, \theta$)

end

Lógica de Primeira Ordem: Forward Chaining

```
procedure FIND-AND-INFER(KB,premises,conclusion, $\theta$ )  
  if premises = [] then  
    FORWARD-CHAIN(KB,SUBST( $\theta$ ,conclusion))  
  else for each  $p'$  em KB tal que UNIFY( $p'$ ,  
    SUBST( $\theta$ , FIRST(premises))) =  $\theta_2$  do  
    FIND-AND-INFER(KB,REST(premises),conclusion,  
      COMPOSE( $\theta$ , $\theta_2$ ))  
end
```

Lógica de Primeira Ordem: Forward Chaining

- (1)
 $Amer(x) \wedge Arma(z) \wedge Nacao(y) \wedge Hostil(y) \wedge Vende(x, z, y) \Rightarrow Crim(x)$
- (2) $Dono(Nono, x) \wedge Missil(x) \Rightarrow Vende(Oeste, Nono, x)$
- (3) $Missil(x) \Rightarrow Arma(x)$
- (4) $Inimigo(x, EUA) \Rightarrow Hostil(x)$

Lógica de Primeira Ordem: Forward Chaining

- Inicialmente KB contém somente as implicações de Horn, adiciona-se cada fato de cada vez.
- FORWARD-CHAIN(KB,Amer(Oeste)): adiciona fato ao KB, unifica com uma premissa de (1), mas as outras não são conhecidas, então termina.
- FORWARD-CHAIN(KB,Nacao(Nono)): adiciona fato ao KB, unifica com uma premissa de (1), mas as outras não são conhecidas, então termina.
- FORWARD-CHAIN(KB,Inimigo(Nono,EUA)): adiciona fato ao KB, unifica com premissa de (3), com MGU $\{x/\text{Nono}\}$, chama:
 - FORWARD-CHAIN(KB,Hostil(Nono)): adiciona fato ao KB, unifica com uma premissa de (1), mas somente duas são conhecidas, então termina.

Lógica de Primeira Ordem: Forward Chaining

- FORWARD-CHAIN(KB,Dono(Nono,M1)): adiciona fato ao KB, unifica com premissa de (2), com MGU $\{x/M1\}$, mas outra premissa ainda não conhecida, então termina.
- FORWARD-CHAIN(KB,Missil(M1)): adiciona fato ao KB, unifica com premissa de (2) e (3):
 - Por (2) Missil(M1), MGU $\{x/M1\}$, como Dono(Nono,M1) já foi provado, chama:
 - * FORWARD-CHAIN(KB,Vende(Oeste,Nono,M1)): adiciona fato ao KB, unifica com uma premissa de (1), com MGU $\{x/Oeste,y/M1,z/Nono\}$, mas Arma(M1) não é conhecida, então termina.

Lógica de Primeira Ordem: Forward Chaining

- Por (3) Missil(M1), MGU $\{x/M1\}$, chama:
 - FORWARD-CHAIN(KB,Arma(M1)): adiciona fato ao KB, unifica com uma premissa de (1), com MGU $\{y/M1\}$, outras premissas conhecidas com MGU acumulado $\{x/Oeste,y/M1,z/Nono\}$, chama:
 - * FORWARD-CHAIN(KB,Crim(Oeste)): adiciona fato ao KB, terminada a prova!

Lógica de Primeira Ordem: Backward Chaining

- Algoritmo Forward Chaining implementa um procedimento *dirigido a dados*. Pode ser incorporado ao procedimento TELL.
- Pode gerar conclusões irrelevantes.
- Algoritmo mais dirigido à consulta: *Backward Chaining*.
- Encontra todas as soluções para uma determinada pergunta ao KB.
- Pode ser incorporado ao procedimento ASK.

Lógica de Primeira Ordem: Backward Chaining

function BACK-CHAIN(KB, q) **returns** a set of substitutions

BACK-CHAIN-LIST($KB, [q], \{\}$)

function BACK-CHAIN-LIST($KB, qlist, \theta$) **returns** a set of substitutions

inputs: KB , a knowledge base

$qlist$, a list of conjuncts forming a query (θ already applied)

θ , the current substitution

static: $answers$, a set of substitutions, initially empty

if $qlist$ is empty **then return** $\{\theta\}$

$q \leftarrow \text{FIRST}(qlist)$

for each q'_i **in** KB such that $\theta_i \leftarrow \text{UNIFY}(q, q'_i)$ succeeds **do**

 Add COMPOSE(θ, θ_i) to $answers$

end

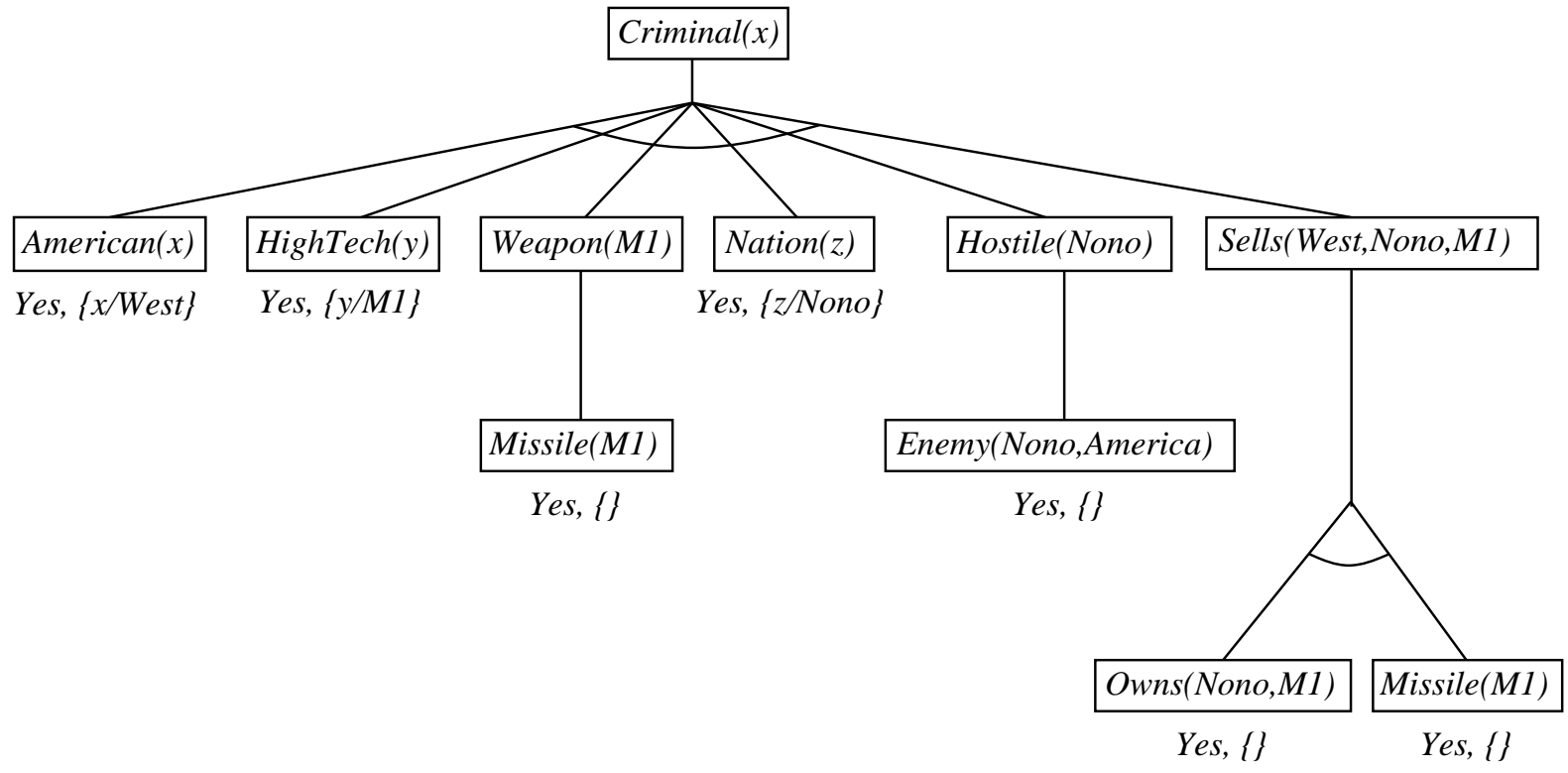
for each sentence $(p_1 \wedge \dots \wedge p_n \Rightarrow q'_i)$ **in** KB such that $\theta_i \leftarrow \text{UNIFY}(q, q'_i)$ succeeds **do**

$answers \leftarrow \text{BACK-CHAIN-LIST}(KB, \text{SUBST}(\theta_i, [p_1 \dots p_n]), \text{COMPOSE}(\theta, \theta_i)) \cup answers$

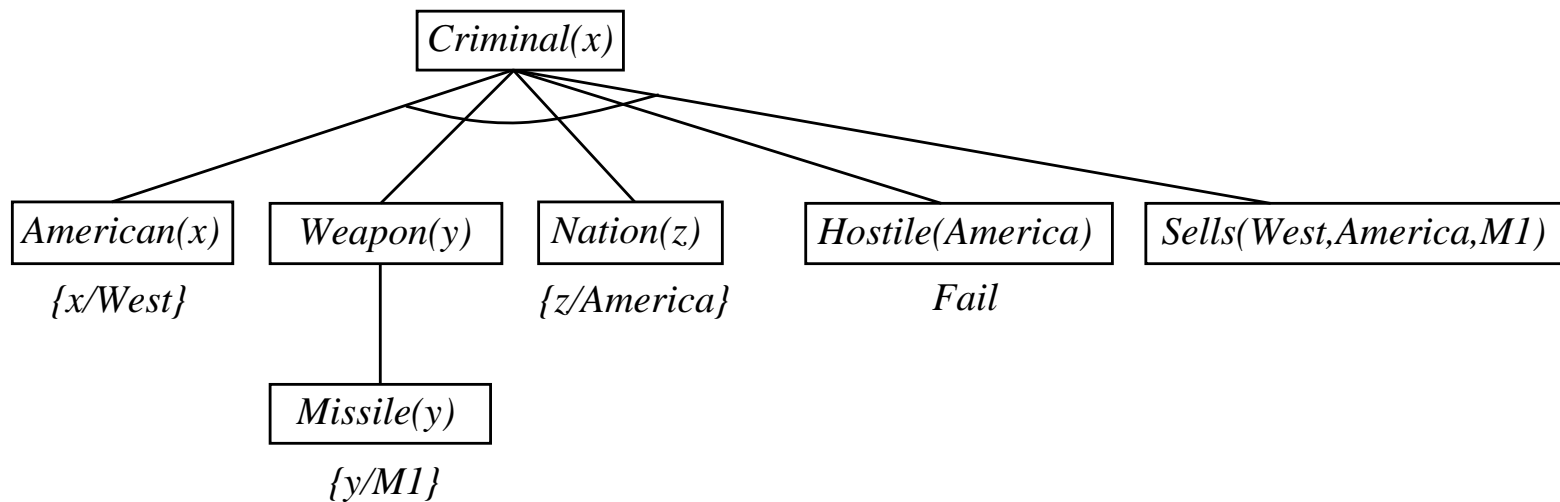
end

return the union of BACK-CHAIN-LIST($KB, \text{REST}(qlist), \theta$) for each $\theta \in answers$

Lógica de Primeira Ordem: Backward Chaining



Lógica de Primeira Ordem: Backward Chaining



Lógica de Primeira Ordem: Completude

- Qq procedimento de prova usando Modus Ponens é *incompleto*.
- Mas se uma sentença for consequência lógica do KB, então existem algoritmos que são completos. Se a sentença não for consequência lógica do KB, então o procedimento pode entrar em loop (*Teorema da Completude de Gödel*): **if** $KB \models \alpha$ **then** $KB \vdash_R \alpha$.
- Problema da validade de lógica de primeira ordem é *semi-decidível*.
- Problema com Modus Ponens: sentenças com literais negados.
- $\forall x P(x) \Rightarrow Q(x)$
- $\forall x \neg P(x) \Rightarrow R(x)$
- $\forall x Q(x) \Rightarrow S(x)$
- $\forall x R(x) \Rightarrow S(x)$

Lógica de Primeira Ordem: Algoritmo de Resolução

- Relembrando regra de resolução de lógica proposicional:
- $\frac{\alpha \vee \beta, \neg \beta \vee \gamma}{\alpha \vee \gamma}$ ou $\frac{\neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{\neg \alpha \Rightarrow \gamma}$
- Duas interpretações: raciocínio por casos ou transitividade da implicação.
- Modus Ponens não permite derivar novas implicações. Somente deriva conclusões atômicas.
- Regra de resolução mais poderosa.
- Regra de resolução generalizada: procedimento de inferência completo para lógica de primeira ordem.

Lógica de Primeira Ordem: Algoritmo de Resolução

- Regra de Resolução generalizada (disjunções):

$$\frac{p_1 \vee \dots \vee p_j \vee \dots \vee p_m \quad q_1 \vee \dots \vee q_k \vee \dots \vee q_n}{SUBST(\theta, (p_1 \vee \dots \vee p_{j-1} \vee p_{j+1} \dots \vee p_m \vee q_1, \vee \dots \vee q_{k-1} \vee q_{k+1} \vee \dots \vee q_n))}$$

- Regra de Resolução generalizada (implicações):

$$\frac{p_1 \wedge \dots \wedge p_j \wedge \dots \wedge p_{n_1} \Rightarrow r_1 \vee \dots \vee r_{n_2} \quad s_1 \wedge \dots \wedge s_{n_3} \Rightarrow q_1 \vee \dots \vee q_k \vee \dots \vee q_{n_4}}{SUBST(\theta, (p_1 \wedge \dots \wedge p_{j-1} \wedge p_{j+1} \wedge p_{n_1} \wedge s_1 \wedge \dots \wedge s_{n_3} \Rightarrow r_1 \vee \dots \vee r_{n_2} \vee q_1 \vee \dots))}$$

Lógica de Primeira Ordem: Algoritmo de Resolução

- Formas canônicas para resolução: *forma conjuntiva normal* ou **forma implicativa normal**.

$$\neg P(w) \vee Q(w) \quad P(w) \Rightarrow Q(w)$$

$$P(x) \vee R(x) \quad True \Rightarrow P(x) \vee R(x)$$

$$\neg Q(y) \vee S(y) \quad Q(y) \Rightarrow S(y)$$

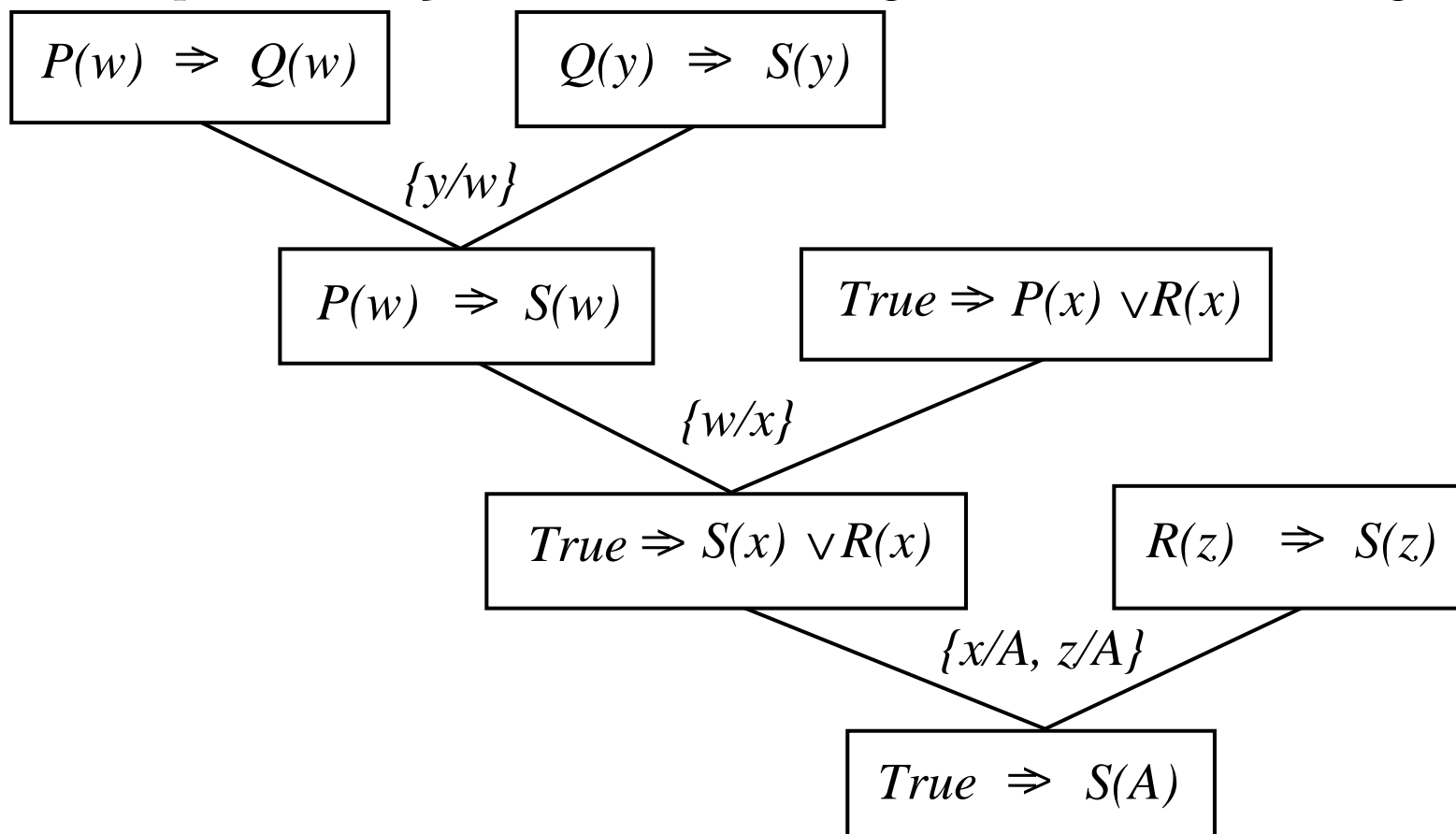
$$\neg R(z) \vee S(z) \quad R(z) \Rightarrow S(z)$$

- Resolução é uma generalização de Modus Ponens:

$$\frac{\alpha, \alpha \Rightarrow \beta}{\beta} \Leftrightarrow \frac{True \Rightarrow \alpha, \alpha \Rightarrow \beta}{True \Rightarrow \beta}$$

Lógica de Primeira Ordem: Algoritmo de Resolução

- Provas por Resolução: forward chaining ou backward chaining.

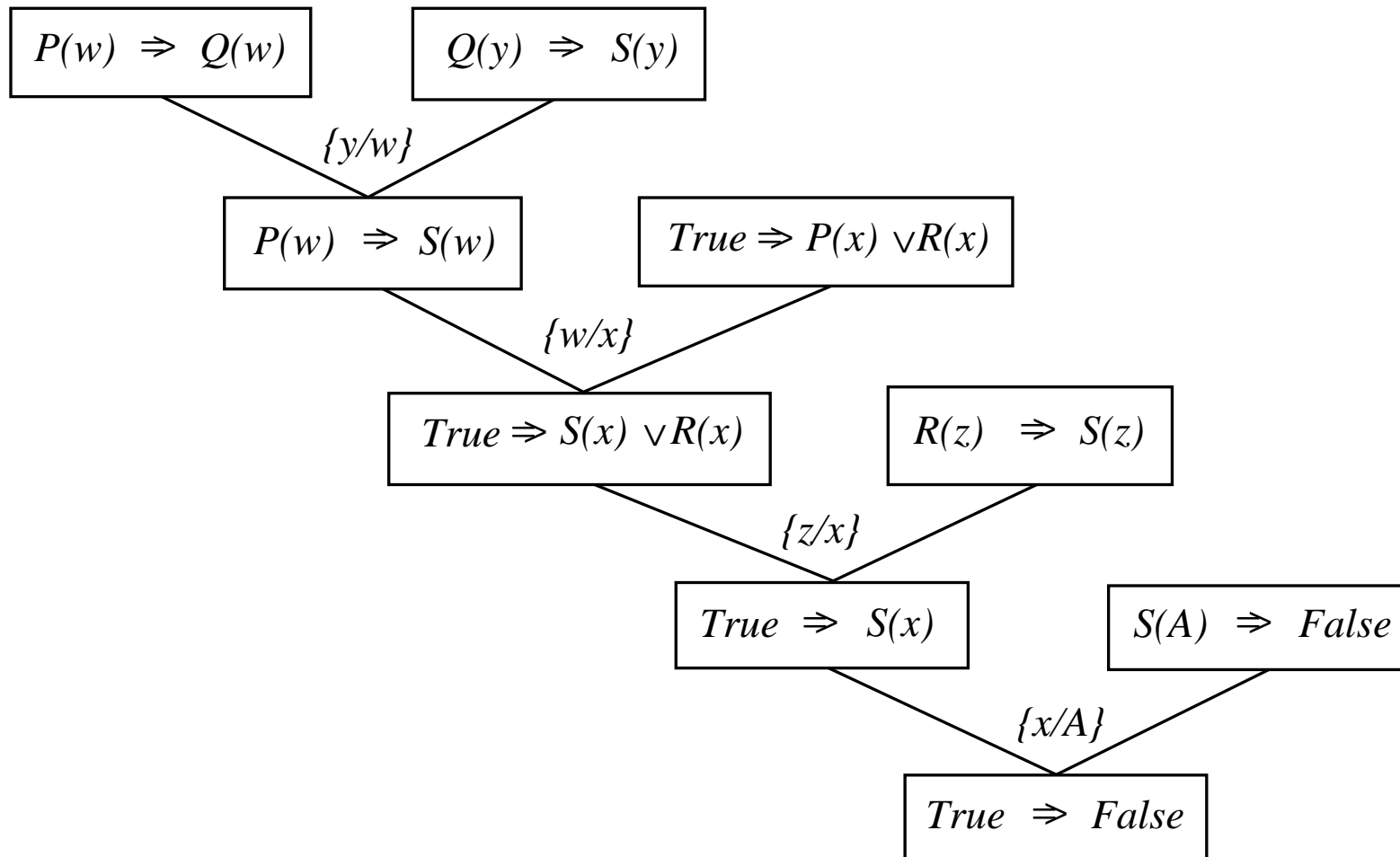


Lógica de Primeira Ordem: Algoritmo de Resolução

- Prova usando resolução ainda é não completa se não tivermos nada no KB. Ex: $p \vee \neg p$. é válida, mas resolução não pode resolver se KB estiver vazio.
- Procedimento de inferência completo usando resolução: *refutação* (redução ao absurdo ou prova por contradição).
- Para provar P, assume-se que P é falso adicionando-se $\neg P$ ao KB. Em outras palavras: $(KB \wedge \neg P \Rightarrow \text{Falso}) \Leftrightarrow (KB \Rightarrow P)$.
- resolução usando refutação é completo e correto.

Lógica de Primeira Ordem: Algoritmo de Resolução

- resolução usando refutação:



Lógica de Primeira Ordem: Conversão para Forma Normal

- Qq sentença em lógica de primeira ordem pode ser convertida em forma implicativa (ou conjuntiva) normal.
- Passos para conversão:
 1. Eliminação de implicações.
 2. Mover negações para dentro dos parênteses.
 3. Renomear variáveis (padronização separada).
 4. Mover quantificadores para a esquerda.
 5. “Skolemização”: $\forall x Pessoa(x) \Rightarrow \exists y Cor(y) \wedge Tem(x, y)$.
 6. Distribuição de \wedge sobre \vee : $(a \wedge b) \vee c$.
 7. Agrupamento de conjunções e disjunções aninhadas:
 $(a \vee b) \vee c$.
 8. Conversão de disjunções para forma implicativa:
 $(\neg a \vee \neg b \vee c \vee d)$.

Lógica de Primeira Ordem: Exemplo de Prova

Joao tem um cachorro.

Todo dono de cachorro ama animais.

Ninguém que ama animais mata animais.

Joao ou a Curiosidade matou o gato, cujo nome é Atum.

A Curiosidade matou o gato?

A. $\exists x Cach(x) \wedge Dono(Joao, x)$

B. $\forall x(\exists y Cach(y) \wedge Dono(x, y)) \Rightarrow AmaAnimais(x)$

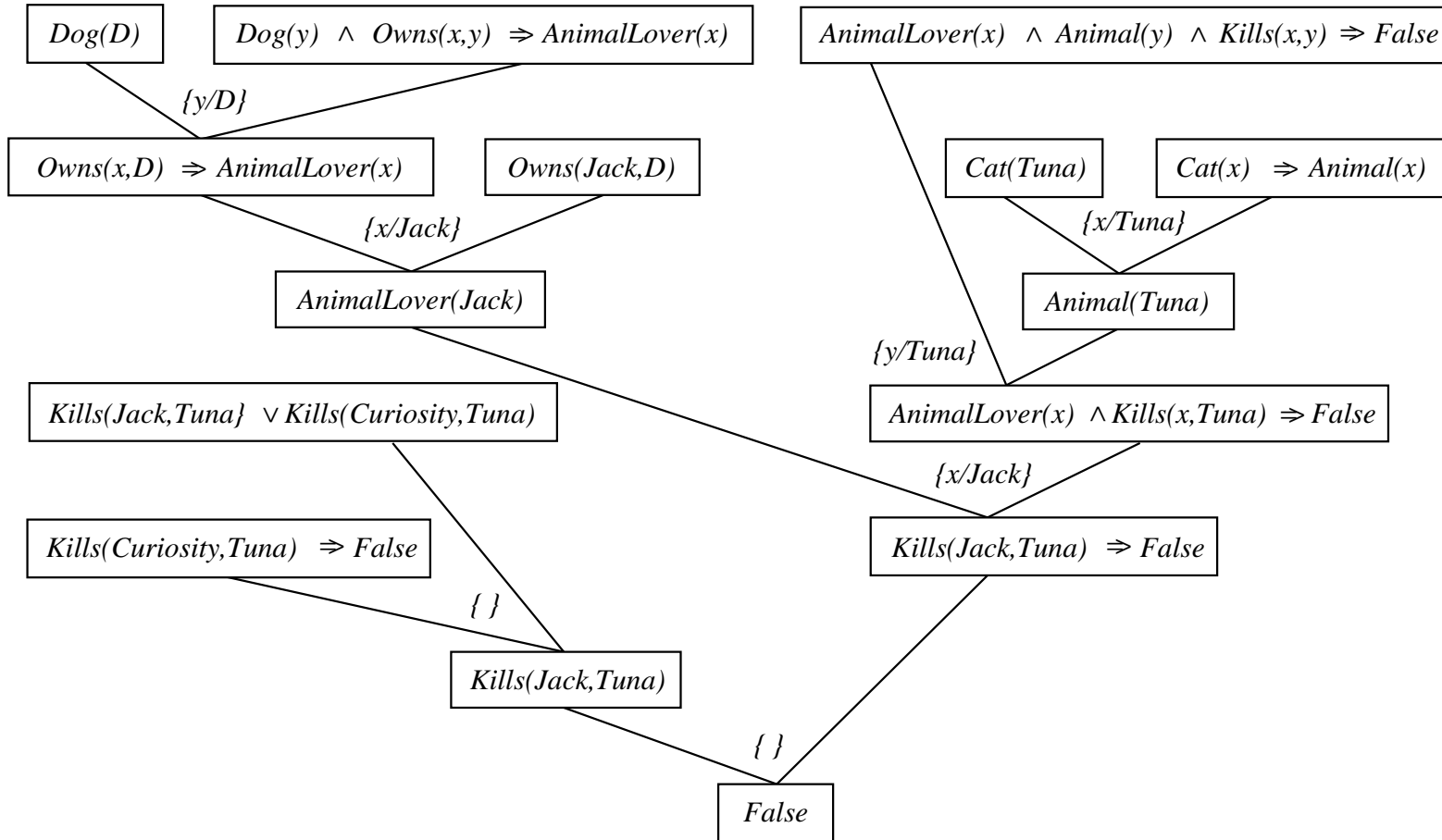
C. $\forall x AmaAnimais(x) \Rightarrow \forall y Animal(y) \Rightarrow \neg Mata(x, y)$

D. $Mata(Joao, Atum) \vee Mata(Curiosidade, Atum)$

E. $Gato(Atum)$

F. $\forall x Gato(x) \Rightarrow Animal(x)$

Lógica de Primeira Ordem: Exemplo de Prova



Lógica de Primeira Ordem: Igualdade de Termos

- $P(A)$ não unifica com $P(B)$ mesmo que exista um fato no KB dizendo que $A=B$.
- Unificação faz somente testes **sintáticos** para verificar se dois termos são iguais.
- duas formas de resolver o problema: *axiomatização da igualdade* ou nova regra de inferência: *demodulação/paramodulação*.

Lógica de Primeira Ordem: Estratégias de Resolução

- Principal preocupação: **eficiência** do método.
- Preferência por sentenças unitárias (atômicas).
- Conjunto de Suporte.
- Resolução de Entrada, **Resolução Linear**.
- Subsumption: elimina sentenças que são mais específicas do que qq sentença existente no KB.