

# 一階述語論理での推論

## Chapter 9

# 概要

- 一階述語論理の推論を命題論理の推論に
- 単一化
- 一般化された肯定式 (Modus Ponens)
- 前向き連鎖
- 後ろ向き連鎖
- 融合 (Resolution)

# 全称限定子のインスタンス化

- 全称限定子による文のインスタンス化は以下の置換により  
伴意されている: 任意の変数  $v$  と基礎項  $g$  に対して

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

- 例:  $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$  yields:  
 $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$   
 $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$   
 $\text{King}(\text{Father}(\text{John})) \wedge \text{Greedy}(\text{Father}(\text{John})) \Rightarrow \text{Evil}(\text{Father}(\text{John}))$   
.  
.  
.

# 存在限定子のインスタンス化

- For any sentence 任意の文 $\alpha$ と変数 $v$ と知識ベースのどこにも**現れない**定数のシンボル $k$ に対して:

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

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

$$\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$$

$C_1$  が新しい定数のシンボルとして導入される。これは**スコレム (Skolem) 定数**と呼ばれる

# 命題論理への還元

知識ベースが次の文だけを持っているとする:

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

King(John)

Greedy(John)

Brother(Richard, John)

- **全ての可能な**方法で全称限定子のインスタンス化をすると:

King(John)  $\wedge$  Greedy(John)  $\Rightarrow$  Evil(John)

King(Richard)  $\wedge$  Greedy(Richard)  $\Rightarrow$  Evil(Richard)

King(John)

Greedy(John)

Brother(Richard, John)

- 新しい知識ベースは**命題論理化**された: 命題論理でのシンボルは

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

# 命題論理への還元

- 一階述語論理での知識ベースは伴意され (entailment) 命題論理に還元することができる
- (基礎項が新しい知識ベースによって意味を伴われているときかつそのときに限り、元の知識ベースによって伴意されている)
- 考え方: 知識ベースと質問を命題論理化し、反駁を応用し、結果を返す
- 問題: 関数シンボルのとき、無限にたくさんの基礎項が存在する
  - 例:  $Father(Father(Father(John)))$

# 命題論理への還元

定理: Herbrand (1930). 文 $\alpha$ が一階述語論理の知識ベースによって伴意されているとき、それは命題論理化されて知識ベースの**有限の**集合によって意味を伴われる

考え方: For  $n = 0$  to  $\infty$  do  
    create a propositional KB by instantiating with depth- $n$  terms  
    see if  $\alpha$  is entailed by this KB

問題:  $\alpha$ が伴意されてるときは機能するが、そうでないときはループとなる

定理: Turing (1936), Church (1936) 一階述語論理での伴意は**半決定的**である(algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)

# 命題論理化の問題

- 命題論理化はたくさんのお互い関係ない文を生成するように見える
- E.g., from:  
     $\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$   
    King(John)  
     $\forall y \text{ Greedy}(y)$   
    Brother(Richard, John)
- *Evil(John)*は明確であると思えるが、命題論理化は*Greedy(Richard)* のような無関係な事実をたくさん生み出す
- $p$ 個の $k$ 変数の述語と $n$ 定数の場合 $p \cdot n^k$  個のインスタンスが存在する.

# 単一化

- $King(x)$ と $Greedy(x)$ は $King(John)$ と $Greedy(y)$ に一致するという置換 $\theta$ が見出せるならすぐに推論することができる

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

- $Unify(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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

- 標準化された別物は変数の重なりを消去できる, 例:  $Knows(z_{17}, OJ)$

# 単一化

- $King(x)$ と $Greedy(x)$ は $King(John)$ と $Greedy(y)$ に一致するという置換 $\theta$ が見出せるならすぐに推論することができる

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

- $Unify(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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)	

- 標準化された別物は変数の重なりを消去できる, 例: Knows( $z_{17}$ , OJ)

# 単一化

- $King(x)$ と $Greedy(x)$ は $King(John)$ と $Greedy(y)$ に一致するという置換 $\theta$ が見出せるならすぐに推論することができる

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

- $Unify(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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

- 標準化された別物は変数の重なりを消去できる, 例:  $Knows(z_{17}, OJ)$

# 単一化

- $King(x)$ と $Greedy(x)$ は $King(John)$ と $Greedy(y)$ に一致するという置換 $\theta$ が見出せるならすぐに推論することができる

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

- $Unify(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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

- 標準化された別物は変数の重なりを消去できる, 例:  $Knows(z_{17}, OJ)$

# 単一化

- $King(x)$ と $Greedy(x)$ は $King(John)$ と $Greedy(y)$ に一致するという置換 $\theta$ が見出せるならすぐに推論することができる

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

- $Unify(\alpha, \beta) = \theta$  if  $\alpha\theta = \beta\theta$

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

- 標準化された別物は変数の重なりを消去できる, 例:  $Knows(z_{17}, OJ)$

# 単一化

- $Knows(John, x)$ と $Knows(y, z)$ を単一化すると  
 $\theta = \{y/John, x/z\}$  あるいは  $\theta = \{y/John, x/John, z/John\}$
- 最初の単一化は2番目と比べるとより汎用である
- 最も汎用な単一化がある。それは変数の名前を変える上で一意的である  
MGU =  $\{y/John, x/z\}$

# 単一化のアルゴリズム

```
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
```

# 単一化のアルゴリズム

```
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  $\theta$ 
```

# 一般化された肯定式( Modus Ponens)

$$\frac{p_1', p_2', \dots, p_n', (p_1 \wedge p_2 \wedge \dots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i \theta \text{ for all } i$$

$p_1'$ is <i>King(John)</i>	$p_1$ is <i>King(x)</i>
$p_2'$ is <i>Greedy(y)</i>	$p_2$ is <i>Greedy(x)</i>
$\theta$ is $\{x/\text{John}, y/\text{John}\}$	$q$ is <i>Evil(x)</i>
$q \theta$ is <i>Evil(John)</i>	

- 確定節(肯定的なリテラルはひとつだけ)で表された知識ベースで使われる一般化された肯定式
- 全ての変数は全称限定子とされる

# 一般化された肯定式の健全性

- 全ての*i*に対して $p_i'\theta = p_i\theta$ ならば、次が成り立つことを証明する必要がある

$$p_1', \dots, p_n', (p_1 \wedge \dots \wedge p_n \Rightarrow q) \vdash q\theta$$

- 補助定理: 任意の文*p*に対して全称限定子のインスタンス化により $p \vdash p\theta$ である

1.  $(p_1 \wedge \dots \wedge p_n \Rightarrow q) \vdash (p_1 \wedge \dots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \dots \wedge p_n\theta \Rightarrow q\theta)$
2.  $p_1', \dots, p_n' \vdash p_1' \wedge \dots \wedge p_n' \vdash p_1'\theta \wedge \dots \wedge p_n'\theta$
3. 1と2から,  $q\theta$  は通常肯定式から導かれる

# 例：知識ベース

- アメリカ人が敵国に武器を売るとは犯罪であると法律で規定されている。敵国Nonoがミサイルを持っている。その全てがアメリカ人であるColonel Westによって売られたものである。
- Col. Westが犯罪者であることを証明せよ

# 例：知識ベース

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

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

Nono ... has some missiles, i.e.,  $\exists x Owns(Nono,x) \wedge Missile(x)$ :

$Owns(Nono,M_1) \text{ and } Missile(M_1)$

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

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

Missiles are weapons:

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

An enemy of America counts as "hostile":

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

West, who is American ...

$American(West)$

The country Nono, an enemy of America ...

$Enemy(Nono,America)$

# 前向き連鎖のアルゴリズム

```
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
```

# 前向き連鎖での証明

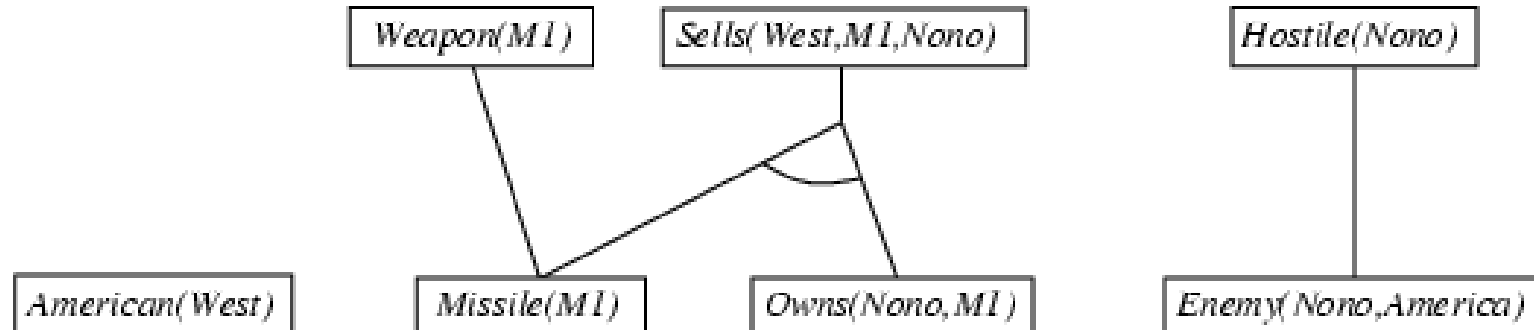
*American(West)*

*Missile(MI)*

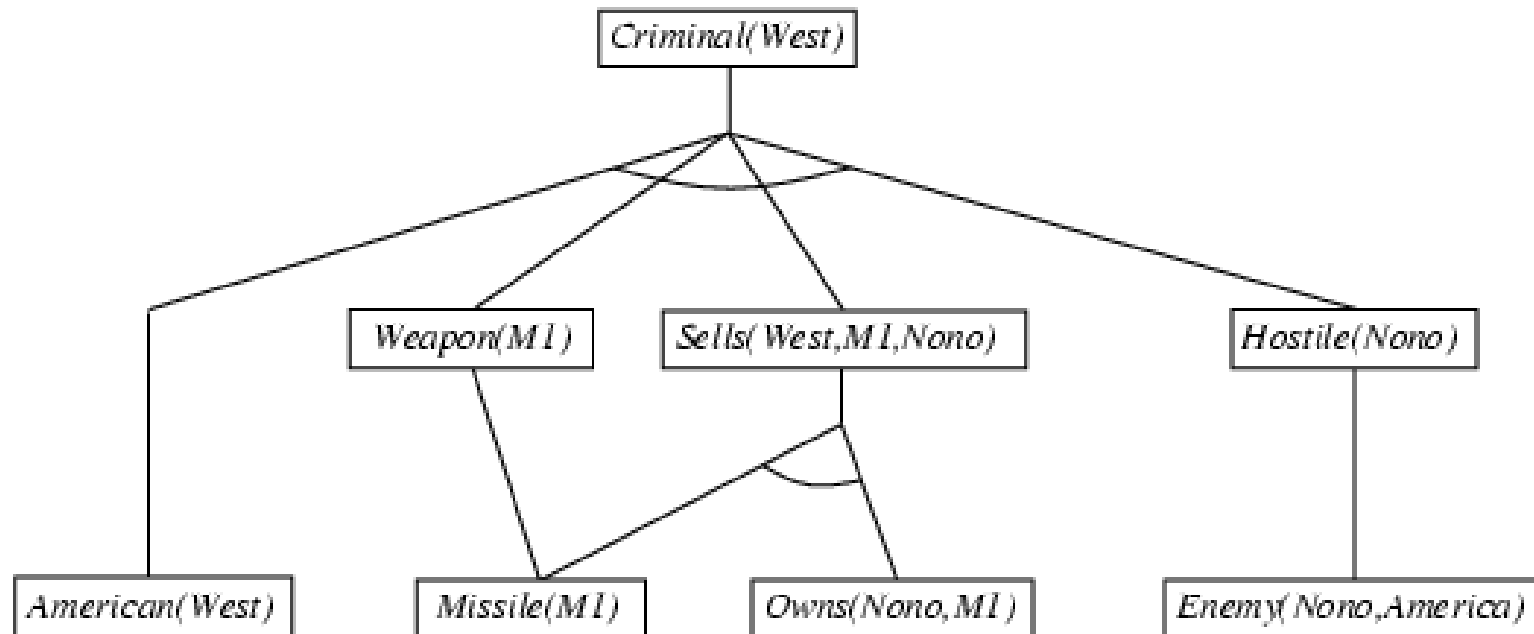
*Owms(Nono, MI)*

*Enemy( Nono, America)*

# 前向き連鎖での証明



# 前向き連鎖での証明



# 前向き連鎖の性質

- 一階述語論理の確定節は健全で完全である
- Datalog = 一階述語論理の確定節+ 関数なし
- 前向き連鎖は有限の繰り返しでDatalogに対して終了する
- $\alpha$  が伴意されていないとき一般的に終了しないかもしれない
- これは避けがたい: 伴意されての確定節は半決定的である

# 前向き連鎖の効率

増分の前向き連鎖: 前提が繰り返し $k-1$ で加えられなければ、繰り返し $k$ でルールに一致する必要はない

⇒ 前提が新たに加えられた肯定のリテラルを含んでいれば、それぞれのルールに一致する

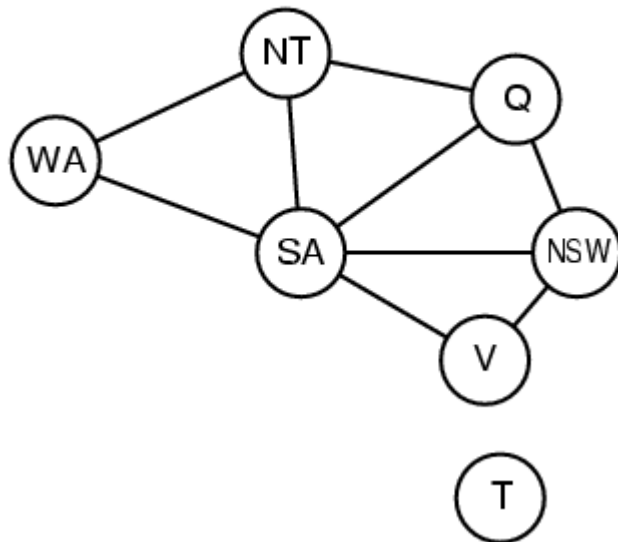
一致それ自身は高くつく:

Database indexing データベースのインデックス は知られている事実 $O(1)$ の検索を必要とする

– e.g., query *Missile(x)* retrieves *Missile(M<sub>1</sub>)*

前向き連鎖は演繹的データベースで広く使われている

# 困難な一致の例



$Diff(wa,nt) \wedge Diff(wa,sa) \wedge Diff(nt,q) \wedge$   
 $Diff(nt,sa) \wedge Diff(q,nsw) \wedge Diff(q,sa) \wedge$   
 $Diff(nsw,v) \wedge Diff(nsw,sa) \wedge Diff(v,sa) \Rightarrow$   
 $Colorable()$

$Diff(Blue,Red) \quad Diff(Blue,Green)$   
 $Diff(Green,Red) \quad Diff(Green,Blue)$   
 $Diff(Red,Blue) \quad Diff(Red,Green)$

- $Colorable()$  は推論されるときかつそのときに限り制約充足問題は解を持つ
- 制約充足問題は特別な場合3SAT(3充足可能性判定問題)を含んでいるので NP困難である

# 後向き連鎖のアルゴリズム

```
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

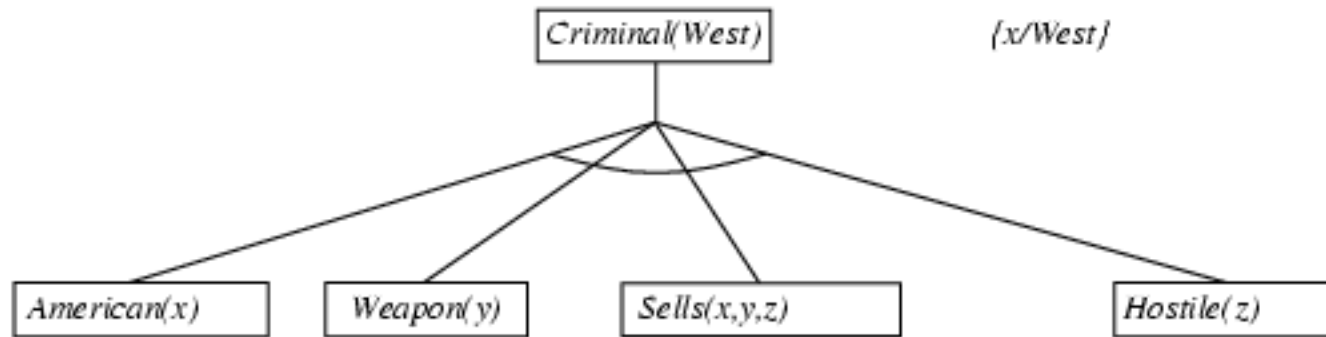
  if goals is empty then return { $\theta$ }
   $q' \leftarrow \text{SUBST}(\theta, \text{FIRST}(\textit{goals}))$ 
  for each r in KB where  $\text{STANDARDIZE-APART}(r) = (p_1 \wedge \dots \wedge p_n \Rightarrow q)$ 
    and  $\theta' \leftarrow \text{UNIFY}(q, q')$  succeeds
     $\textit{ans} \leftarrow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

$$\text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p))$$

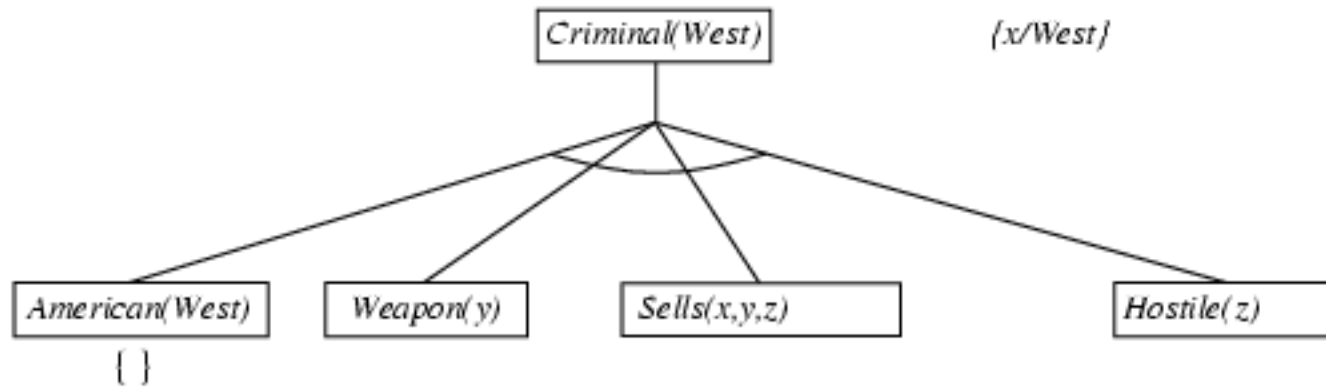
# 後向き連鎖の例

*Criminal(West)*

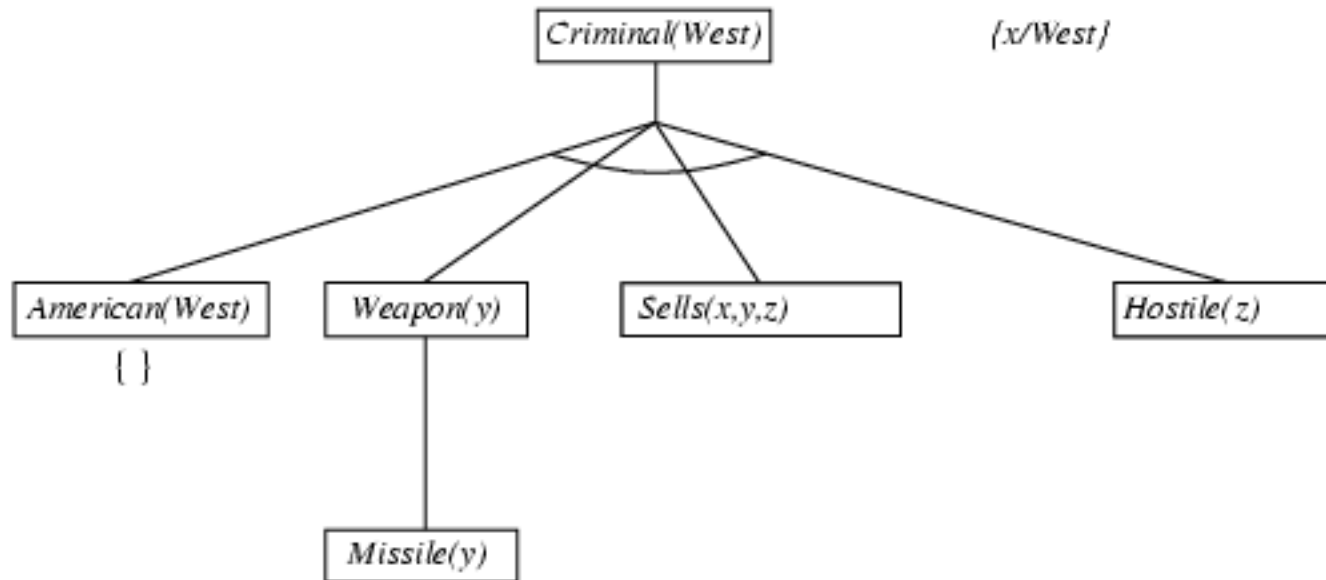
# 後向き連鎖の例



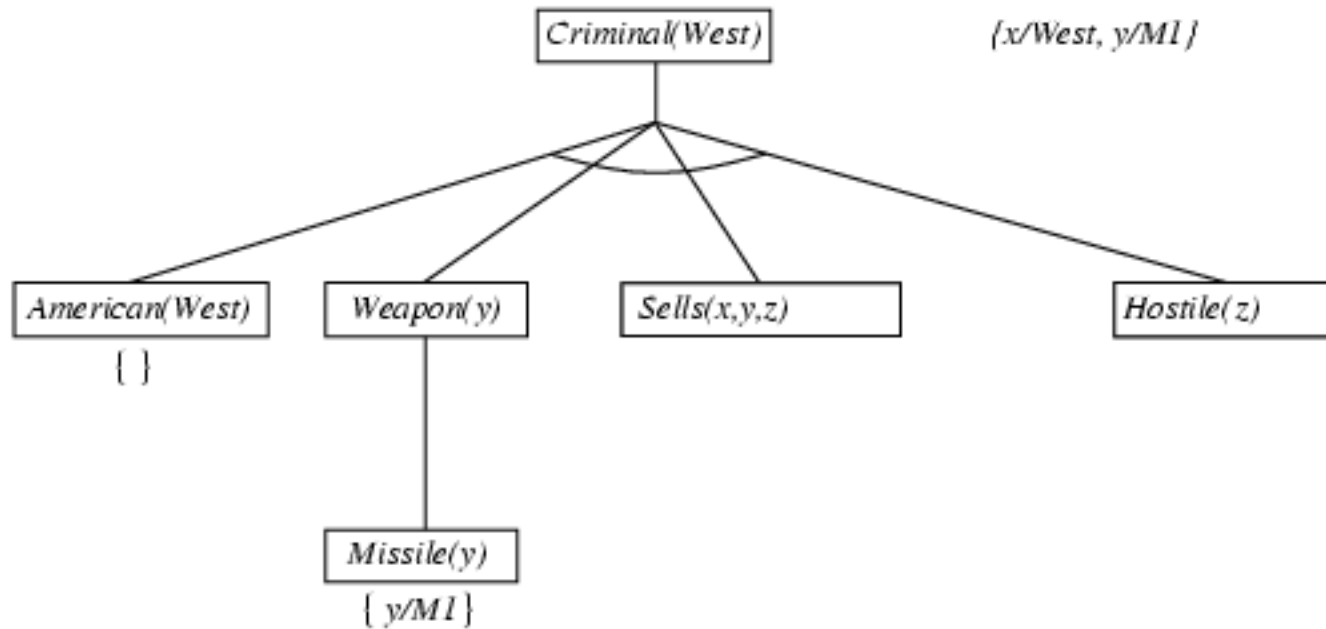
# 後向き連鎖の例



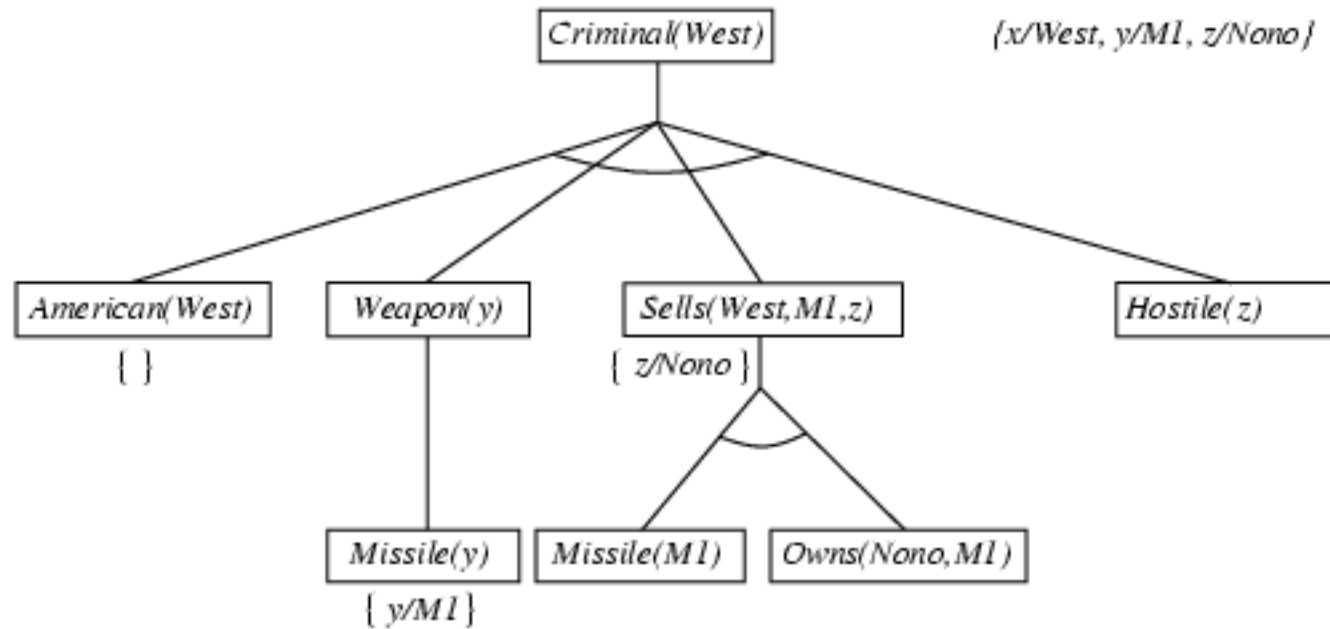
# 後向き連鎖の例



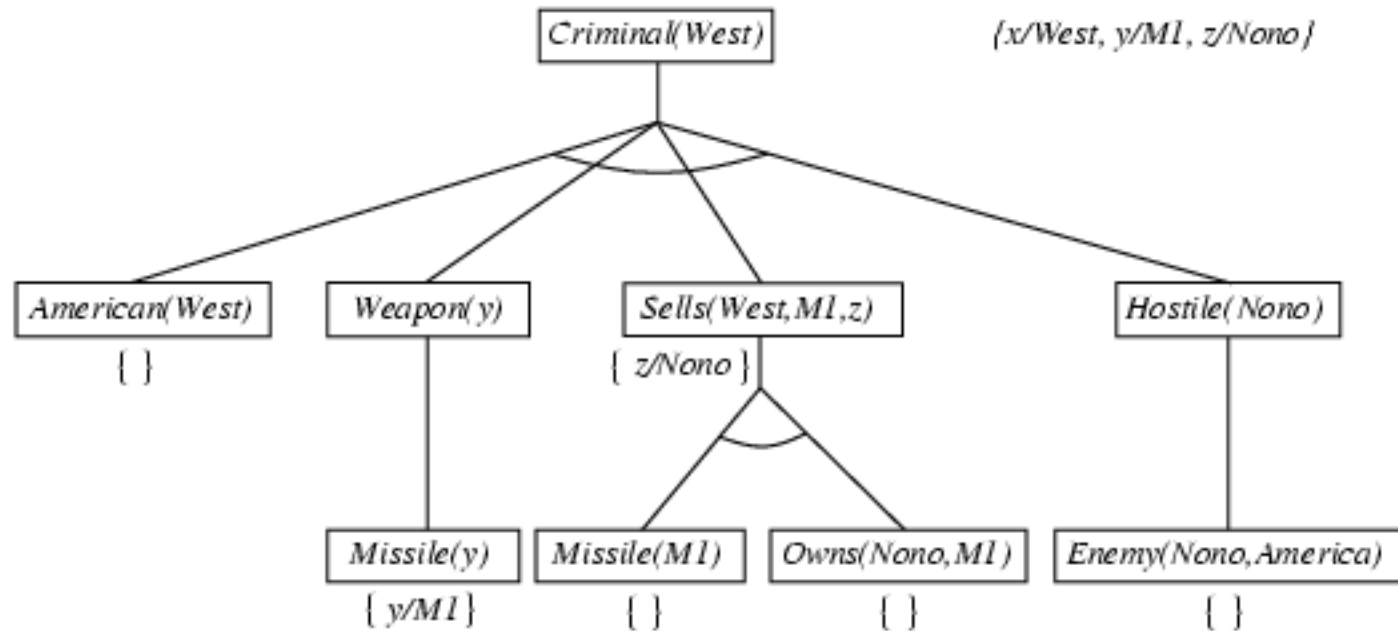
# 後向き連鎖の例



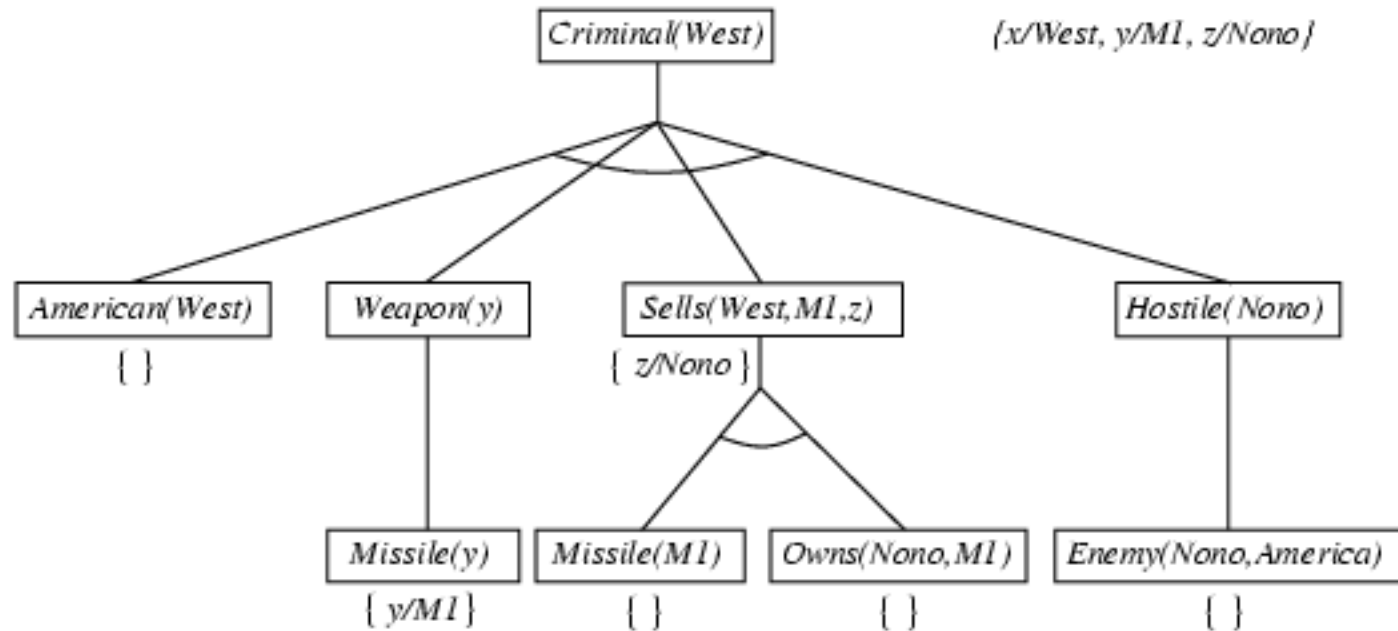
# 後向き連鎖の例



# 後向き連鎖の例



# 後向き連鎖の例



# 後向き連鎖の性質

- 深さ優先再帰証明探索: 空間は証明の大きさに線形である
- 無限ループのため不完全
  - ⇒ スタック上の全てのゴールに対して現在のゴールを調べればそうでなくなる
- 繰り返される部分ゴールのため非効率 (both success and failure)
  - ⇒ 前の結果を蓄積しておくことでそうでなくなる (余分な空間)
- 論理プログラムで広く使われている

# 論理型プログラム: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses + bells & whistles  
Widely used in Europe, Japan (basis of 5th Generation project)  
Compilation techniques  $\Rightarrow$  60 million LIPS
- Program = set of clauses = head :- literal<sub>1</sub>, ... literal<sub>n</sub>.  
`criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails

# 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=[]
```

# 融合: 簡単なまとめ

- 一階述語版:

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

ここで  $\text{Unify}(l_i, \neg m_j) = \theta$ .

- 二つの節は標準化された別物とする。従って、変数は共有していない
- 例

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

---

$$\text{Unhappy}(\text{Ken})$$

$\theta = \{x/\text{Ken}\}$ である

- 連言正規形( $\text{KB} \wedge \neg\alpha$ )に融合のステップを適応する。一階述語論理に対しては完全である

# 連言正規形への変換

- Everyone who loves all animals is loved by someone:

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

- 1. 同値 $\Leftrightarrow$ と含意 $\Rightarrow$ を消去する

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

- 2.  $\neg$  を内に移動する:  $\neg \forall x p \equiv \exists x \neg p$ ,  $\neg \exists x p \equiv \forall x \neg p$

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

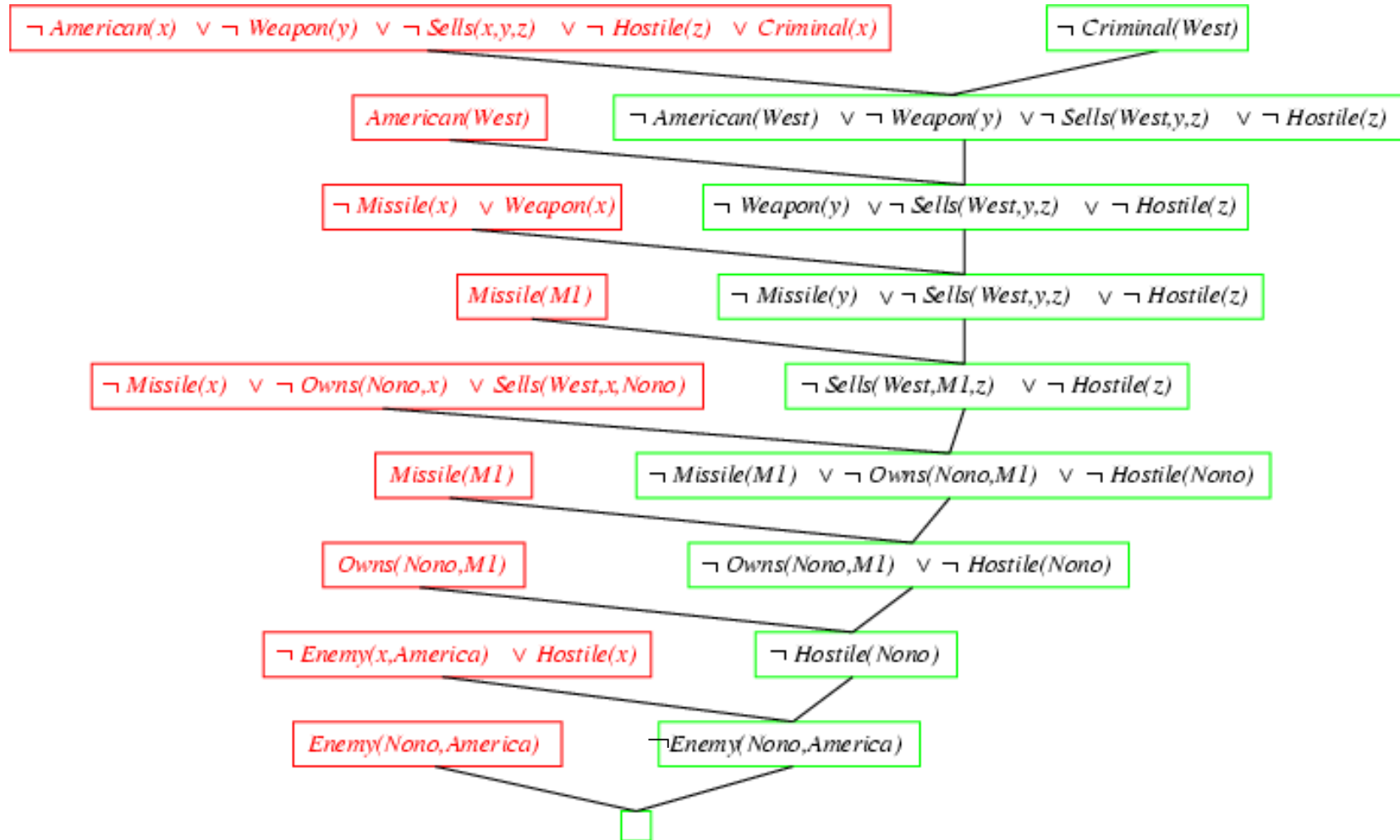
$$\forall x [\exists y \neg \neg \textit{Animal}(y) \wedge \neg \textit{Loves}(x,y)] \vee [\exists y \textit{Loves}(y,x)]$$

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

# 連言正規形への変換

3. 標準化された変数: それぞれの限定子は異なるものを使用する  
 $\forall x [\exists y \text{ Animal}(y) \wedge \neg \text{Loves}(x,y)] \vee [\exists z \text{ Loves}(z,x)]$
4. スコーレム化: 存在限定子のインスタンス化でのより一般的な形式  
全称限定子の変数で表されたスコーレム関数によって存在限定子の変数を置き換える:  
 $\forall x [\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$
5. 全称限定子を落とす:  
 $[\text{Animal}(F(x)) \wedge \neg \text{Loves}(x,F(x))] \vee \text{Loves}(G(x),x)$
6.  $\vee$  の外に  $\wedge$  を分配する:  
 $[\text{Animal}(F(x)) \vee \text{Loves}(G(x),x)] \wedge [\neg \text{Loves}(x,F(x)) \vee \text{Loves}(G(x),x)]$

# 融合による証明: 確定節



確定節: 正リテラルをちょうど一つ含む節