First-Order Logic

AIMA Chapter 8, 9

Inference in first-order logic

通用实例化

Universal instantiation (UI)

object
(Term without variables)

result of substituting a

- For any sentence α, variable *v* and ground term *g*:

$\forall x\ A$

with free x in A

$$\frac{\forall v\ \alpha}{\mathrm{Subst}(\{v/g\}, \alpha)} \longleftarrow \text{Substitute } v \text{ with } g \text{ in } \alpha$$

substitute

$\Rightarrow A : \{x \mapsto \alpha\}$.

- Every instantiation of a universally quantified sentence is entailed by it
- UI can be applied multiple times to add new sentences
- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:
  - $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  - $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  - $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

"可代入"

Existential instantiation (EI)

- For any sentence α, variable v, and constant symbol k that does not appear elsewhere in the knowledge base:

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

$\exists x\ P(x)$

$\Rightarrow P(a)$

- EI can be applied once to replace an existential sentence
- E.g., $\exists x\ Crown(x) \land OnHead(x, John)$ yields:

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

provided C_1 is a new constant symbol, called a Skolem constant

Reduction to propositional inference

- Suppose the KB contains just the following:
  - $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  - King(John)
  - Greedy(John)
  - Brother(Richard,John)

Reduction contd.

- Instantiating the universal sentence in all possible ways, we have:
  - eg King(John) $\wedge$ Greedy(John) $\Rightarrow$ Evil(John)
  - King(Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)
  - King(John)
  - Greedy(John)
  - Brother(Richard,John)

P- logic!

- The new KB is propositionalized: proposition symbols are
  - King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment

体化

  – i.e., a ground sentence is entailed by new KB iff entailed by original KB
- A naïve idea for FOL inference:
  – propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms,
  – e.g., *Father*(*Father*(*Father*(*John*)))

通用实例化

Universal instantiation (UI)

result of substituting a

object

(Term without variables)

- For any sentence α, variable v and ground term g:

$\forall x A$ with free x in A

$$\frac{\forall v\, \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$ ← Substitute v with g in α

substitute

$\Rightarrow A : \{x \mapsto \alpha\}$.

- Every instantiation of a universally quantified sentence is entailed by it
- UI can be applied multiple times to add new sentences
- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:
  - $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  - $King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard)$
  - $King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John))$

"可代入"

Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment 维持
  - i.e., a ground sentence is entailed by new KB iff entailed by original KB
- A naïve idea for FOL inference:
  - propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*)))

Reduction contd.

- Theorem (Herbrand, 1930)
  - If a sentence α is entailed by an FOL KB, it is entailed by a finite subset of the propositionalized KB
- Idea:

  For n = 0 to ∞ do

  1. create a propositional KB by instantiating with depth-n terms
  2. if α is entailed by this KB, return true

Function nesting levels

Does this work?

Entail

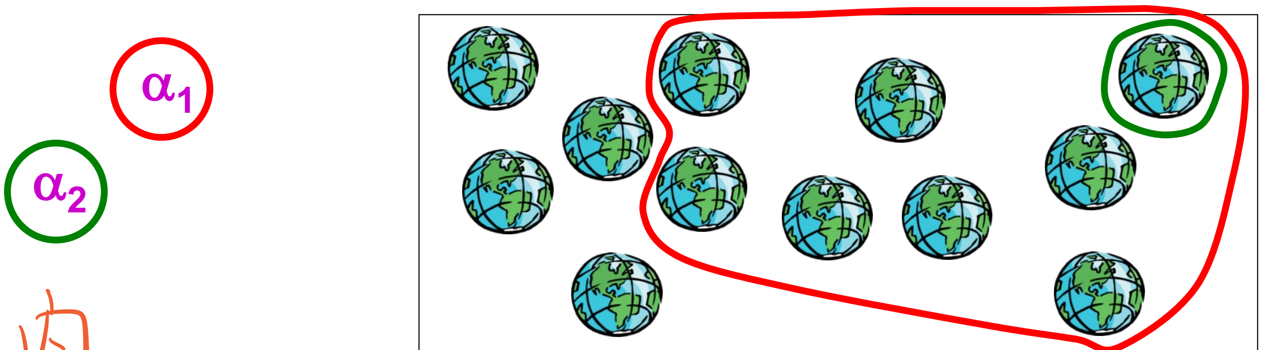
Inference: entailment

使必要

- Entailment: $\alpha \models \beta$ ("α entails β" or "β follows from α") means in every world where α is true, β is also true
  – i.e., the α-worlds are a subset of the β-worlds [models(α) $\subseteq$ models(β)]
- In the example, $\alpha 2 \models \alpha 1$

world = model

Q why



set 内

同时性： $A \cap B$ true, A true

Reduction contd.

- Problem
  - works if α is entailed
  - infinite loops if α is not entailed

- Theorem (Turing, 1936; Church, 1936): entailment for FOL is semi-decidable
  - algorithms exist that say yes to every entailed sentence
  - but no algorithm exists that says no to every non-entailed sentence.

Problems with propositionalization

- Propositionalization generates many irrelevant sentences
- E.g., from:
  - $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  - $King(John)$
  - $\forall y\ Greedy(y)$

The query *Evil*(*John*) seems obviously true. But propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant.

Unification

- Given:
  - $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
  - $King(John)$
  - $\forall y\ Greedy(y)$

Only variables can be substituted

- If we can find the substitution $\theta = \{x/John, y/John\}$, then we get
  - $King(John) \wedge Greedy(John) \Rightarrow Evil(John)$
  - $King(John)$
  - $Greedy(John)$

  and we can answer the query *Evil(John)* immediately
- Unification finds substitutions that make different expressions identical
  - E.g., *King(x)* vs. *King(John)*; *Greedy(x)* vs. *Greedy(y)*

Unification

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

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)	

Unification

Unify(α, β).

Unify (p, q).

- Unify(α,β) = θ if αθ = βθ

p (α)	q (β)	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)	

Unification

- Unify(α,β) = θ if αθ = βθ

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)	

Unification

- Unify(α,β) = θ if αθ = βθ

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)	

John John

Unification

- Unify(α,β) = θ if αθ = βθ

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}

Can be avoided by standardizing apart: eliminate overlap of variables, e.g., Knows(z,OJ)

x ?

What to eliminate

MGU most genera

Unification

- To unify Knows(John,x) and Knows(y,z),
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}
  - The first unifier is more general than the second.

- There is a single most general unifier (MGU) that is unique up to renaming of variables.
  - MGU = { y/John, x/z }

Uni

FOL Inference

- Horn logic (the FOL case)
  - Forward chaining
  - Backward chaining

在命题逻辑里，“苏格拉底是哲学家”、“帕雷托是哲学家”只能简单标记为 p 及 q。

而在一阶逻辑，我们先将符号 $\mathrm{Phil}(x)$ 解释为“x 是哲学家”，然后以 s 代表苏格拉底；b 代表帕雷托，则 $\mathrm{Phil}(s)$ 对应到 p；$\mathrm{Phil}(b)$ 对应到 q。也就是我们赋予**断言符号**“$\mathrm{Phil}(x)$”**语义的解释**，而这个解释默认一个“所有人类的群体”(也就是我们一阶逻辑语义的论域)，将变量 x 解释为从这个群体取出来讨论的一个人。

事实上断言符号可以包含不只一个的变量，比如说我们可以把 $\mathrm{Cp}(x, y)$ 解释为“*x*与*y*是夫妻”，这样

$$\mathrm{Cp}(s, y)$$

就可以解释为“苏格拉底和*y*是夫妻”。

进一步的，断言符号和变量还可以和逻辑符号组合成更复杂的叙述。例如将断言符号 $\mathrm{Schol}(x)$ 解释为 x 为学者。则“若*x*为哲学家，则*x*为学者”可以表示为

$$\mathrm{Phil}(x) \Rightarrow \mathrm{Schol}(x)$$

而“对所有*x*，若*x*为哲学家，则*x*为学者”之类的叙述，我们写为

$$\forall x(\mathrm{Phil}(x) \Rightarrow \mathrm{Schol}(x))$$

也就是自左方开始阅读，以 $\forall x$ 代表“对所有*x*”；将叙述理解为“对所有的*x*，$\forall x$ 右方的叙述为真。”而 $\forall x$ 这个符号我们称为 x 的全称量词

但全称量词直观上，会有“若所有*x*是哲学家，那*x*的长子也会是哲学家”这样的叙述，为此我们将$\mathrm{Son}(x)$ 解释为“*x*的长子”，这样前面的叙述可以写为

$$\forall x \mathrm{Phil}(x) \Rightarrow \mathrm{Phil}[\mathrm{Son}(x)]$$

这种解释为成与 x 有唯一对应的那个对象的符号，称为**函数符号**。而事实上这段直观为真的叙述，经过适当的扩展以后就是一阶逻辑其中的一条公理。

而对于“有*x*是哲学家”，我们引入另一种量词而表记为:

$$\exists x(\mathrm{Phil}(x))$$

自左方开始阅读，以 $\exists x$代表“存在*x*”；也就是解释为“有*x*使 $\exists x$ 右方的叙述为真”。而 $\exists x$ 被称为 x 的存在量词。全称量词和存在量词一起被简称为**量词**

而直观上，“并非所有*x*不是哲学家”，和“有*x*是哲学家”是等价的；且“并非有*x*不是学者”也跟“所有的*x*是学者”在直观上也是等价的。所以只要有“否定”这个逻辑概念，那一阶逻辑就只需要一种量词。据此，并且为了使逻辑公理的叙述更加简洁，一般会以全称量词为基础，作以下的符号定义($\neg$ 解释为“否定”，而 $\mathcal{A}$ 代表一段"叙述"):

$$\exists x \mathcal{A} := \neg[\forall x(\neg \mathcal{A})]$$

而将存在量词定义为全称量词的衍伸符号。

在通常的一阶逻辑语义解释上，需要一个量词所提及对象所组成的非空集合，称为论域。例如，$\exists x \mathrm{Phil}(x)$为真的意思为，若论域里有对象使断言Phil为真。也就是严格来说，通常的一阶逻辑语义需要一套集合论基础，但一般的严谨的公理化集合论都是以一阶逻辑的语言来叙述（如策梅洛-弗兰克尔集合论）。这不免让一阶逻辑语义沦为"具有集合论公理的一阶逻辑理论的一套元定理"，用来有效的检验一段"叙述"(合式公式)是否为这个一阶逻辑理论的定理。

Horn clauses in FOL

$\neg p \vee \neg q \cdots \vee \neg t \vee u$ only positive

- $p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$
  - $p_1, p_2, \ldots, p_n$, q are atomic sentences
  - All variables assumed to be universally quantified
- E.g., *human*(x) $\Rightarrow$ *mortal*(x)

Generalized Modus Ponens (GMP)

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

where $p_i'\theta = p_i\,\theta$ for all i

Example: King(John), Greedy(y), (King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x))

到

p_1' is *King*(*John*) p_1 is *King*(*x*)

p_2' is *Greedy*(*y*) p_2 is *Greedy*(*x*)

Therefore, θ is {x/John, y/John}

q is *Evil*(*x*), so $q\theta$ is *Evil*(*John*)

Horn Def

A Horn clause is a clause (a disjunction of literals) with at most one positive, i.e. unnegated, literal.

Conversely, a disjunction of literals with at most one negated literal is called a **dual-Horn clause**.

A Horn clause with exactly one positive literal is a **definite clause**[2] or a **strict Horn clause**;[3] a definite clause with no negative literals is a **unit clause**,[4] and a unit clause without variables is a **fact**;[5]. A Horn clause without a positive literal is a **goal clause**. Note that the empty clause, consisting of no literals (which is equivalent to *false*) is a goal clause. These three kinds of Horn clauses are illustrated in the following propositional example:

Type of Horn clause	Disjunction form	Implication form	Read intuitively as
Definite clause	$\neg p \lor \neg q \lor \ldots \lor \neg t \lor u$	$u \leftarrow p \land q \land \ldots \land t$	assume that, if p and q and ... and t all hold, then also u holds
Fact	u	$u \leftarrow true$	assume that u holds
Goal clause	$\neg p \lor \neg q \lor \ldots \lor \neg t$	$false \leftarrow p \land q \land \ldots \land t$	show that p and q and ... and t all hold[note 1]

All variables in a clause are implicitly universally quantified with the scope being the entire clause. Thus, for example:

$\neg\ human(X) \lor mortal(X)$

stands for:

$\forall X(\neg\ human(X) \lor mortal(X))$

which is logically equivalent to:

$\forall X\ (human(X) \rightarrow mortal(X))$

Logic programming [edit]

Horn clauses are also the basis of logic programming, where it is common to write definite clauses in the form of an implication:

$(p \land q \land \ldots \land t) \rightarrow u$

In fact, the resolution of a goal clause with a definite clause to produce a new goal clause is the basis of the SLD resolution inference rule, used in implementation of the logic programming language Prolog.

In logic programming, a definite clause behaves as a goal-reduction procedure. For example, the Horn clause written above behaves as the procedure:

to show u, show p and show q and ... and show t.

To emphasize this reverse use of the clause, it is often written in the reverse form:

$u \leftarrow (p \land q \land \ldots \land t)$

In Prolog this is written as:

```
u :- p, q, ..., t.
```

In logic programming, computation and query evaluation are performed by representing the negation of a problem to be solved as a goal clause. For example, the problem of solving the existentially quantified conjunction of positive literals:

$\exists X\,(p \land q \land \ldots \land t)$

is represented by negating the problem (denying that it has a solution), and representing it in the logically equivalent form of a goal clause:

$\forall X\,(false \leftarrow p \land q \land \ldots \land t)$

In Prolog this is written as:

```
:- p, q, ..., t.
```

Solving the problem amounts to deriving a contradiction, which is represented by the empty clause (or "false"). The solution of the problem is a substitution of terms for the variables in the goal clause, which can be extracted from the proof of contradiction. Used in this way, goal clauses are similar to conjunctive queries in relational databases, and Horn clause logic is equivalent in computational power to a universal Turing machine.

The Prolog notation is actually ambiguous, and the term "goal clause" is sometimes also used ambiguously. The variables in a goal clause can be read as universally or existentially quantified, and deriving "false" can be interpreted either as deriving a contradiction or as deriving a successful solution of the problem to be solved.[further explanation needed]

Van Emden and Kowalski (1976) investigated the model-theoretic properties of Horn clauses in the context of logic programming, showing that every set of definite clauses **D** has a unique minimal model **M**. An atomic formula **A** is logically implied by **D** if and only if **A** is true in **M**. It follows that a problem **P** represented by an existentially quantified conjunction of positive literals is logically implied by **D** if and only if **P** is true in **M**. The minimal model semantics of Horn clauses is the basis for the stable model semantics of logic programs.[7]

Clause (logic)

From Wikipedia, the free encyclopedia

For other uses, see Clause (disambiguation).

In logic, a **clause** is a propositional formula formed from a finite collection of literals (atoms or their negations) and logical connectives. A clause is true either whenever at least one of the literals that form it is true (a disjunctive clause, the most common use of the term), or when all of the literals that form it are true (a conjunctive clause, a less common use of the term). That is, it is a finite disjunction[1] or conjunction of literals, depending on the context. Clauses are usually written as follows, where the symbols l_i are literals:

$$l_1 \lor \cdots \lor l_n$$

- The US 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.

- Prove that Col. West is a criminal

Example knowledge base contd.

... 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)*

Q why decompose like this?

Missiles are weapons:

Missile(x) ⇒ *Weapon(x)*

An enemy of America counts as "hostile“:

Enemy(x,America) ⇒ *Hostile(x)*

West, who is American …

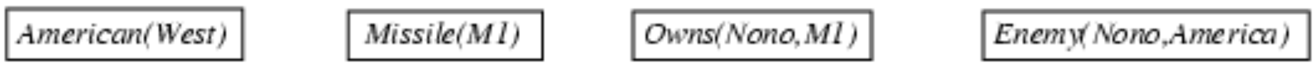
American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Forward chaining proof

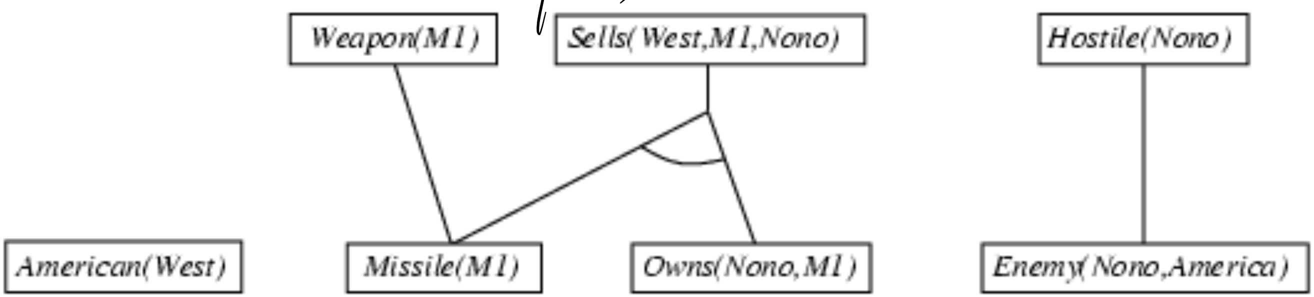
- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



Forward chaining proof

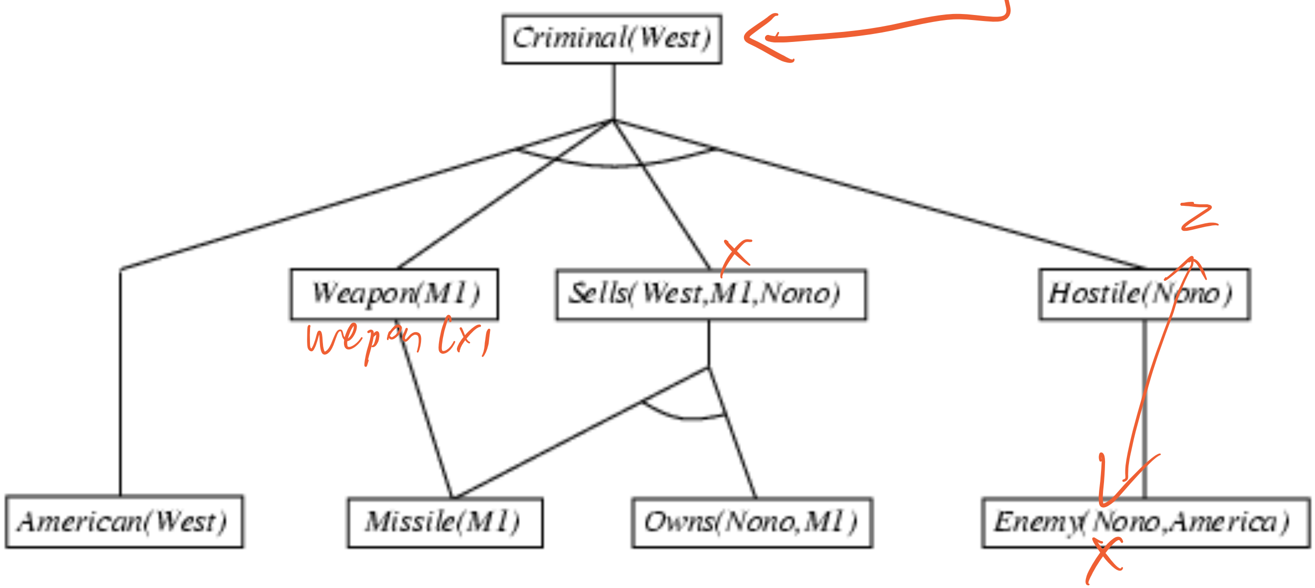
- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*

(missile->weapon)



Forward chaining proof

- $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x,America) \Rightarrow Hostile(x)$
- $Owns(Nono,M_1)$, $Missile(M_1)$, $American(West)$, $Enemy(Nono,America)$



Properties of forward chaining

子句

- Sound and complete for first-order Horn clauses

forward chaining

- FC terminates for first-order Horn clauses with no functions (Datalog) in finite number of iterations
- In general, FC may not terminate if α is not entailed
  – This is unavoidable: entailment with Horn clauses is also semi-decidable

Backward chaining example

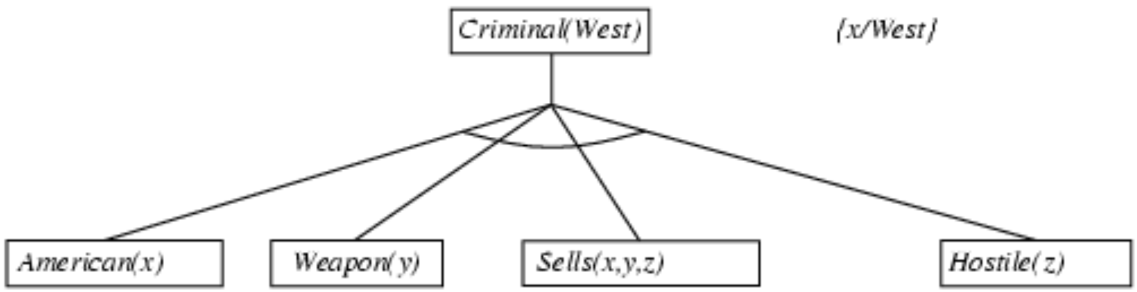
- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*

$\boxed{Criminal(West)}$

存在对应关系的

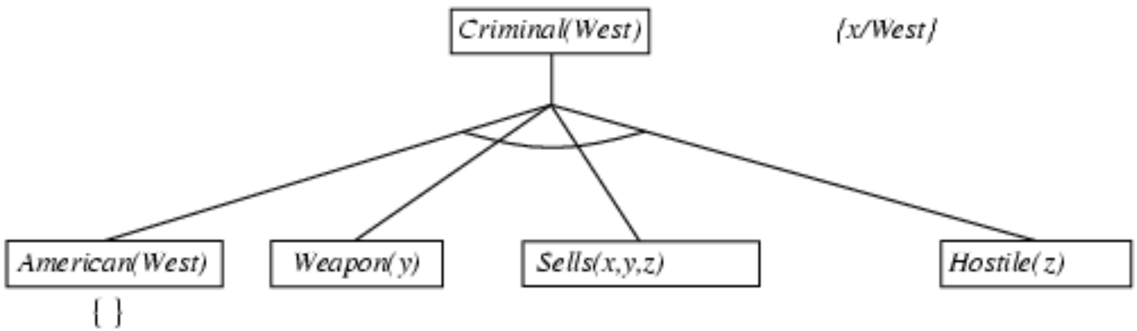
Backward chaining example

- *American(x)* $\wedge$ *Weapon(y)* $\wedge$ *Sells(x,y,z)* $\wedge$ *Hostile(z)* $\Rightarrow$ *Criminal(x)*
- *Missile(x)* $\wedge$ *Owns(Nono,x)* $\Rightarrow$ *Sells(West,x,Nono)*
- *Missile(x)* $\Rightarrow$ *Weapon(x)*
- *Enemy(x,America)* $\Rightarrow$ *Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



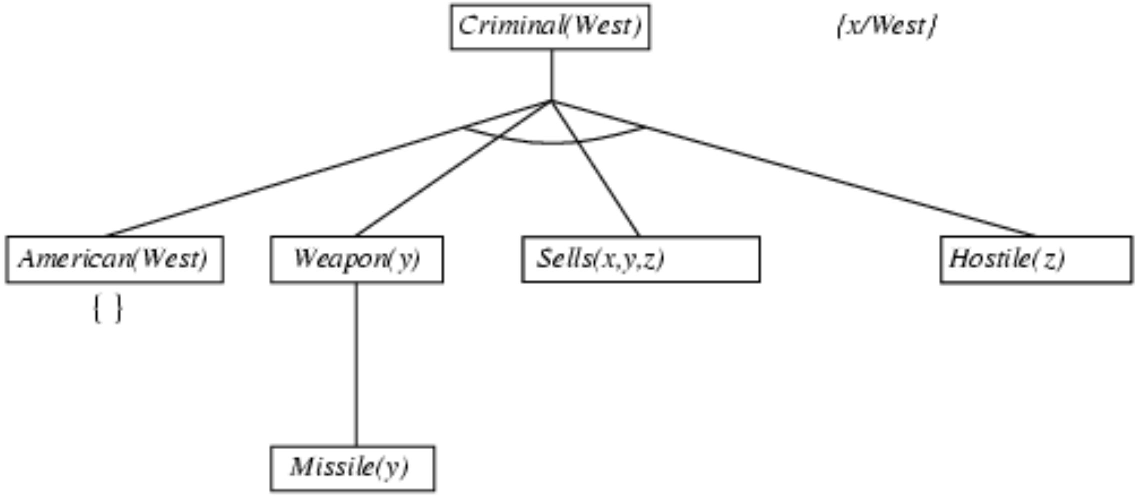
Backward chaining example

- *American(x)* $\wedge$ *Weapon(y)* $\wedge$ *Sells(x,y,z)* $\wedge$ *Hostile(z)* $\Rightarrow$ *Criminal(x)*
- *Missile(x)* $\wedge$ *Owns(Nono,x)* $\Rightarrow$ *Sells(West,x,Nono)*
- *Missile(x)* $\Rightarrow$ *Weapon(x)*
- *Enemy(x,America)* $\Rightarrow$ *Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



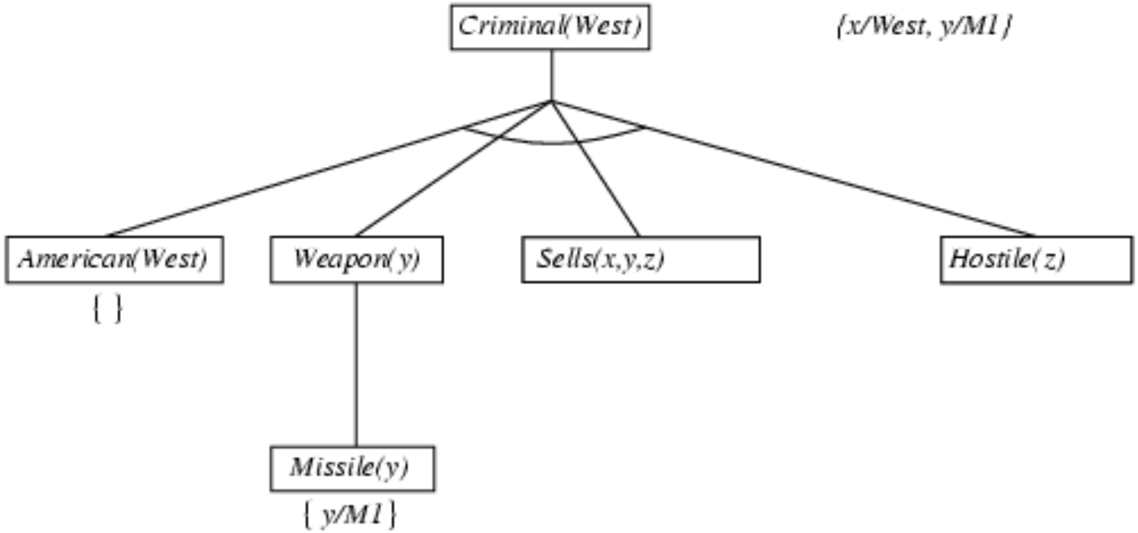
Backward chaining example

- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



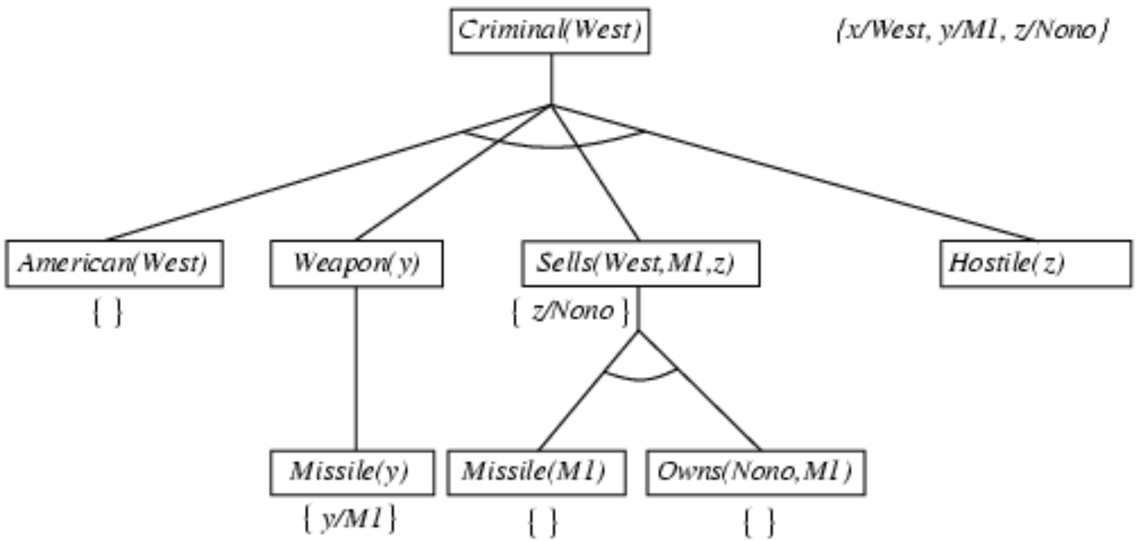
Backward chaining example

- *American(x) $\wedge$ Weapon(y) $\wedge$ Sells(x,y,z) $\wedge$ Hostile(z) $\Rightarrow$ Criminal(x)*
- *Missile(x) $\wedge$ Owns(Nono,x) $\Rightarrow$ Sells(West,x,Nono)*
- *Missile(x) $\Rightarrow$ Weapon(x)*
- *Enemy(x,America) $\Rightarrow$ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



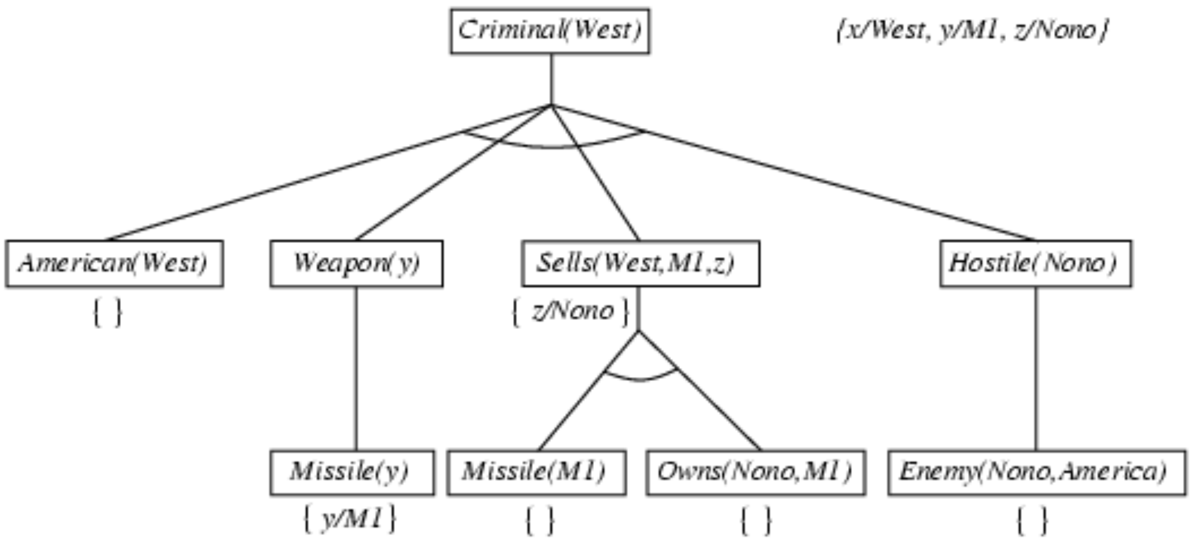
Backward chaining example

- *American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)*
- *Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)*
- *Missile(x) ⇒ Weapon(x)*
- *Enemy(x,America) ⇒ Hostile(x)*
- *Owns(Nono,M_1), Missile(M_1), American(West), Enemy(Nono,America)*



Backward chaining example

- $American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
- $Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
- $Missile(x) \Rightarrow Weapon(x)$
- $Enemy(x,America) \Rightarrow Hostile(x)$
- $Owns(Nono,M_1)$, $Missile(M_1)$, $American(West)$, $Enemy(Nono,America)$



Backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Avoid infinite loops by checking current goal against every goal on stack
- Avoid repeated subgoals by caching previous results
- Widely used for logic programming

Logic programming

- Ordinary programming
  - Identify problem
  - Assemble information
  - Figure out solution
  - Encode solution
  - Encode problem instance as data
  - Apply program to data
- Logic programming
  - Identify problem
  - Assemble information
  - <coffee break> ☺
  - Encode info in KB
  - Encode problem instances as facts
  - Ask queries (run SAT solver)

Logic programming: Prolog

- Was widely used in Europe, Japan (basis of 5th Generation project)
- Basis: backward chaining with Horn clauses
  - Program = set of Horn clauses
  - Inference: depth-first, left-to-right backward chaining
- Additions:
  - 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")

Resolution

- Full first-order version:

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

  where `Unify`($\ell_i, \neg m_j$) = θ.

  α

  – The two clauses are assumed to be standardized apart so that they share no variables.

- Example:

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

  with θ = {x/Ken}

conjunction of clause .

- Inference algorithm: applying resolution steps to CNF(KB ∧ ¬α)
- Resolution is sound and complete for FOL

Conversion to CNF

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

1. Eliminate biconditionals and implications

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

2. Move $\neg$ inwards: $\neg \forall x\, p \equiv \exists x\, \neg p$, $\neg \exists x\, p \equiv \forall x\, \neg p$

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

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

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

1. 过 ∀ ∃
2. 去括号

DeMorgan

3. Standardize variables: each quantifier should use a different variable

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

Conversion to CNF contd.

4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

   只留一个变量!

5. Drop universal quantifiers:

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

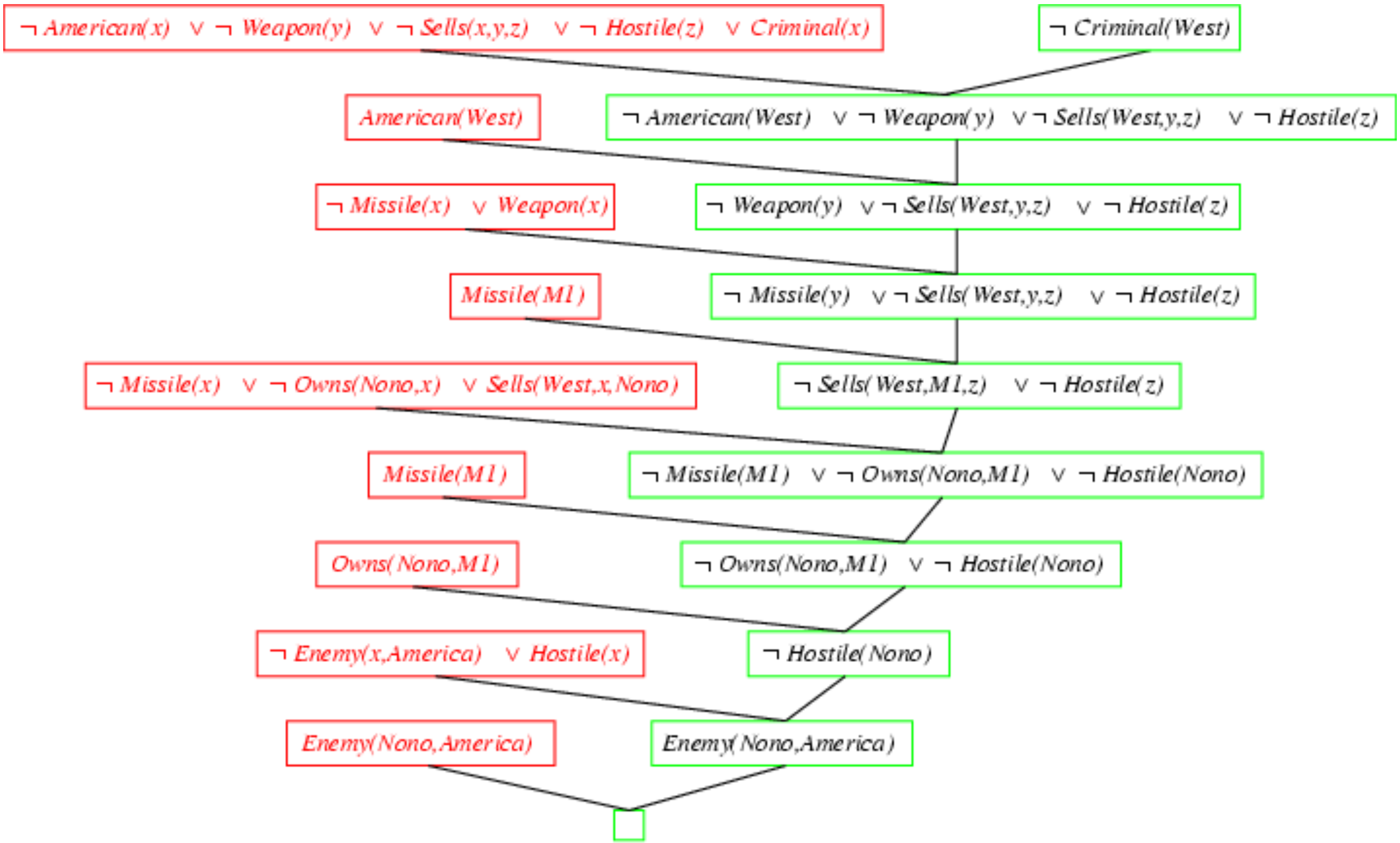
   $\forall$ Q : $\exists$

6. Distribute $\vee$ over $\wedge$:

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

   into $(\vee) \wedge (\vee)$

Resolution proof: Horn clauses



Summary

- First-order logic:
  - objects and relations are semantic primitives
  - syntax: constants, functions, predicates, quantifiers
- Inference
  - Unification
  - Forward/backward chaining
  - Resolution
- Semantic web
  - Application of predicate logic to WWW