Artificial Intelligence (ENCS434)

First Order Logic

Dr. Mustafa Jarrar

University of Birzeit

mjarrar@birzeit.edu
www.jarrar.info

Acknowledgement

The slides in this lecture are based on the slides developed by Prof. Enrico Franconi*

See the online course on Description Logics http://www.inf.unibz.it/~franconi/dl/course/

(But notice that I introduced some modifications.)

Reading

All slides + everything I say

Chapter 8 and Chapter 9

Outline

- FOL, First Order Logic
  - Motivation (why FOL)
  - Syntax
  - Semantics
- FOL Inference Methods
  - Enumeration Method
  - Inference rules
  - Resolution
  - Forward and backward Chaining

Motivation

- We can already do a lot with propositional logic.
- But it is unpleasant that we cannot access the *structure of atomic sentences.*
- Atomic formulas of propositional logic are *too atomic . they are just statement.*
- which my be <u>true</u> or <u>false</u> but which have no internal structure.
- In *First Order Logic (FOL) the atomic formulas are interpreted as statements* about ***relationships between objects***.

Predicates and Constants

Let's consider the statements:

- *Mary is female*
- *John is male*
- *Mary and John are siblings*

In propositional logic the above statements are atomic propositions:

- Mary-is-female
- John-is-male
- Mary-and-John-are-siblings

In FOL atomic statements use predicates, with constants as argument:

- Female(mary)
- Male(john)
- Siblings(mary, john)

Variables and Quantifiers

Let's consider the statements:

- *Everybody is male or female*
- *A male is not a female*

In FOL predicates may have variables as arguments, whose value is bounded by quantifiers:

- $\forall x.\ Male(x) \vee Female(x)$
- $\forall x.\ Male(x) \rightarrow \neg Female(x)$

Deduction (why?):

- *Mary is not male*
- $\neg$ Male(mary)

Functions

Let's consider the statement:

- *The father of a person is male*

In FOL objects of the domain may be denoted by functions applied to (other) objects:

- $\forall x. Male(father(x))$

Syntax of FOL: atomic sentences

Countably infinite **supply of symbols (*signature*):**

variable symbols: $x, y, z, \ldots$

n-ary function symbols: $f, g, h, \ldots$

individual constants: $a, b, c, \ldots$

n-are predicate symbols: $P, Q, R, \ldots$

Terms:	$t \rightarrow x$	variable
	$\mid\ a$	constant
	$\mid\ f(t_1, \ldots, t_n)$	function application

Ground terms: terms that do not contain variables

Formulas: $\phi \rightarrow P(t_1, \ldots, t_n)$ atomic formulas

E.g., Brother(kingJohn; richardTheLionheart)

>(length(leftLegOf(richard)), length(leftLegOf(kingJohn)))

Outline

- FOL, First Order Logic
  - Motivation (why FOL)
  - Syntax
  - Semantics
- FOL Inference Methods
  - Enumeration Method
  - Inference rules
  - Resolution
  - Forward and backward Chaining

Syntax of Propositional Logic

Formulas: $\phi, \psi \rightarrow$		$P(t_1,\ldots,t_n)$	atomic formulas
	$\mid$	$\perp$	false
	$\mid$	T	true
	$\mid$	$\neg\phi$	negation
	$\mid$	$\phi \wedge \psi$	conjunction
	$\mid$	$\phi \vee \psi$	disjunction
	$\mid$	$\phi \rightarrow \psi$	implication
	$\mid$	$\phi \leftrightarrow \psi$	equivalence

(Ground) **atoms** and (ground) **literals.**

E.g. Sibling(kingJohn, richard) $\rightarrow$ Sibling(richard, kingJohn)

$>(1, 2) \vee \leq(1, 2)$

$>(1, 2) \wedge \neg>(1, 2)$

Syntax of First Order Logic

Formulas: $\phi, \psi \rightarrow P(t_1,\ldots,t_n)$ atomic formulas

$\mid$	$\bot$	false
$\mid$	T	true
$\mid$	$\neg\phi$	negation
$\mid$	$\phi \wedge \psi$	conjunction
$\mid$	$\phi \vee \psi$	disjunction
$\mid$	$\phi \rightarrow \psi$	implication
$\mid$	$\phi \leftrightarrow \psi$	equivalence
$\mid$	$\forall x.\phi$	universal quantification
$\mid$	$\exists x.\phi$	*existential* quantification

E.g. Everyone in Italy is smart: $\forall$x. In(x, Italy) $\rightarrow$ Smart(x)

Someone in France is smart: $\exists$x. In(x, France) $\wedge$ Smart(x)

Summary of Syntax of FOL

Terms

- variables
- constants
- functions

Literals

- atomic formula
  - relation (predicate)
- negation

Well formed formulas

- truth-functional connectives
- existential and universal quantifiers

Outline

- FOL, First Order Logic
  - Motivation (why FOL)
  - Syntax
  - Semantics
- FOL Inference Methods
  - Enumeration Method
  - Inference rules
  - Resolution
  - Forward and backward Chaining

Semantics of FOL: intuition

- Just like in propositional logic, a (complex) FOL formula may be true (or false) with respect to a given interpretation.

- An interpretation specifies referents for

  constant symbols → ***objects***

  predicate symbols → ***relations***

  function symbols → ***functional relations***

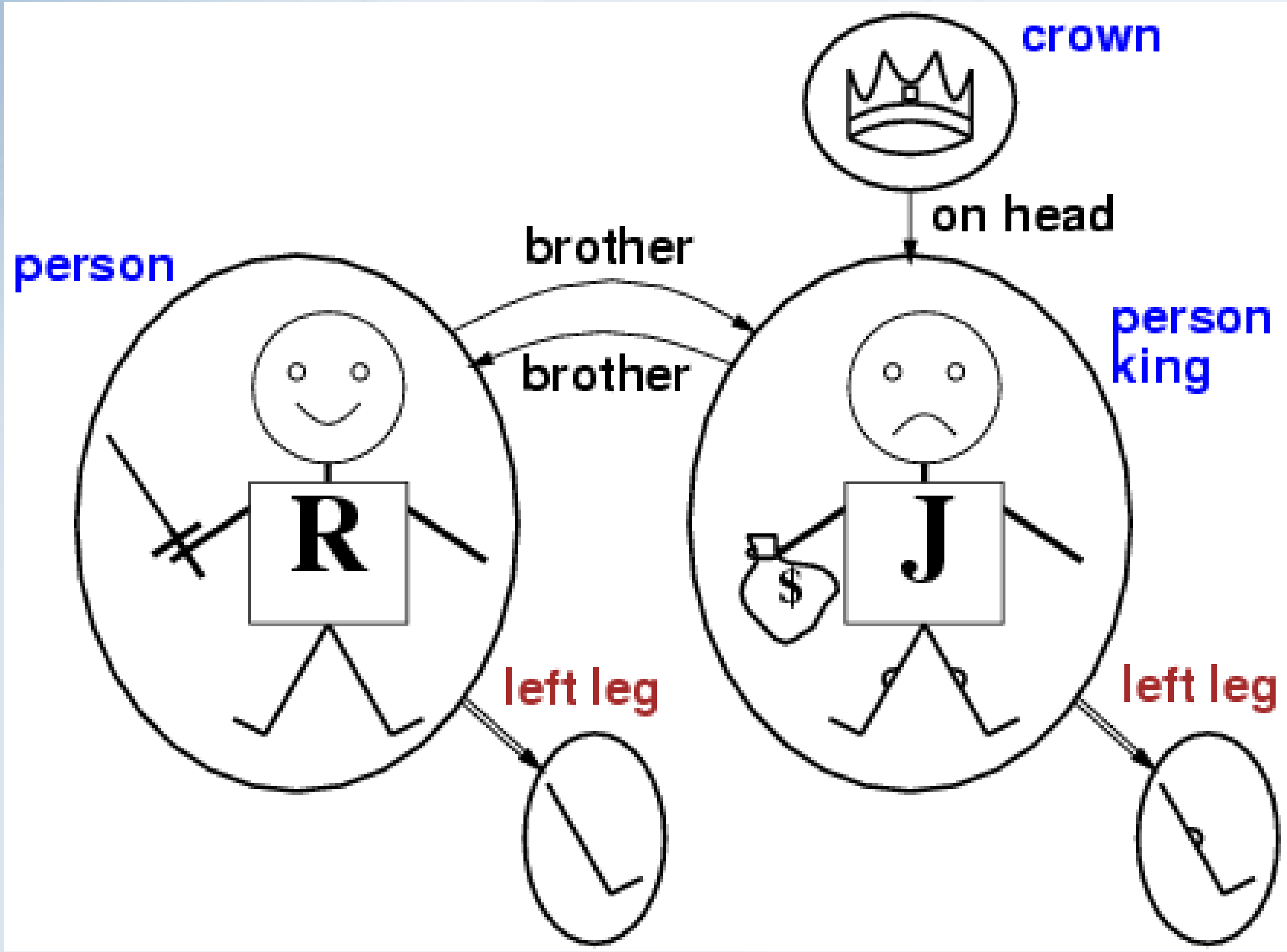
- An atomic sentence $P(t_1,\ldots, t_n)$ is true in a given interpretation

  iff the *objects referred to by* $t_1,\ldots, t_n$

  are in the *relation referred to by the predicate* P.

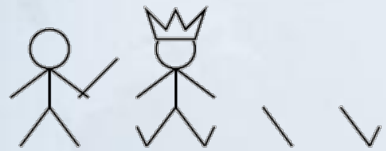
- An interpretation in which a formula is true is called a ***model*** *for the formula.*

Models for FOL: Example



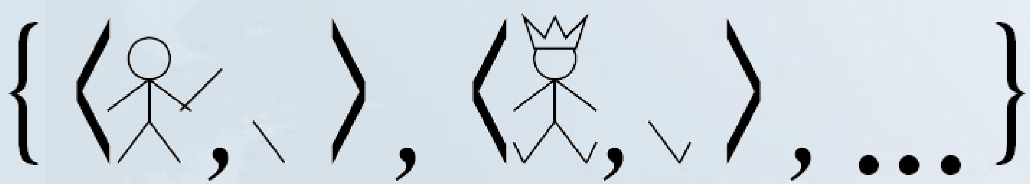
Models for FOL: Example

Objects



Relations: sets of tuples of objects

Functional relations: all tuples of objects + "value" object



Semantic of FOL: Interpretations

Interpretation: $I = \langle \Delta, .^I \rangle$ where Δ is an arbitrary non-empty set and I is a function that maps:

- Individual constants to elements of Δ :

  $a^I \in \Delta$

- n-ary predicate symbols to relation over Δ :

  $P^I \subseteq \Delta^n$

- n-ary function symbols to functions over Δ :

  $f^I \in [\Delta^n \rightarrow \Delta]$

Semantic of FOL: Satisfaction

Interpretation of ground terms:

$$(f(t_1,\ldots,t_n))^I = f^I(t^I_1,\ldots,t^I_n) \quad (\in \Delta)$$

Satisfaction of ground atoms $P(t_1,\ldots,t_n)$:

$$I \models P(t_1,\ldots,t_n) \quad \text{iff} \quad \langle t^I_1,\ldots,t^I_n \rangle \in P^I$$

Examples

$\Delta = \{d_1, \ldots, d_n, n > 1\}$

$a^I = d_1$

$b^I = d_2$

$Block^I = \{d_1\}$

$Red^I = \Delta$

$\Delta = \{1,2,3,\ldots\}$

$1^I = 1$

$2^I = 2$

….

$Even^I = \{2,4,6,\ldots\}$

$Succ^I = \{(1 \rightarrow 2), \mathbf{(2 \rightarrow 3)}, \ldots\}$

Examples

$\Delta = \{d_1, \ldots, d_n, n > 1\}$

$a^I = d_1$

$b^I = d_2$

$\text{Block}^I = \{d_1\}$

$\text{Red}^I = \Delta$

$\Delta = \{1, 2, 3, \ldots\}$

$1^I = 1$

$2^I = 2$

….

$\text{Even}^I = \{2, 4, 6, \ldots\}$

$\text{Succ}^I = \{(1 \rightarrow 2), \mathbf{(2 \rightarrow 3), \ldots}\}$

$I \models \text{Red}(b)$

$I \mid\neq \text{Block}(b)$

$I \mid\neq \text{Even}(3)$

$I \models \text{Even}(\text{succ}(3))$

Semantics of FOL: Variable Assignments

V set of all variables. Function α: $V \rightarrow \Delta$.

Notation: $\alpha[x/d]$ means assign d to x

Interpretation of terms *under c*

$$x^{I,\alpha} = \alpha(x)$$

$$a^{I,\alpha} = a^{I}$$

$$(f(t_1,..,t_n))^{I,\alpha} = f^{I}(t_1^{I,\alpha},\ldots,t_n^{I,\alpha})$$

Satisfiability of atomic formulas:

$$I,\alpha \models P(t_1,\ldots,t_n) \quad \text{iff} \quad \langle t_1^{I,\alpha},\ldots,t_n^{I,\alpha}\rangle \in P^{I}$$

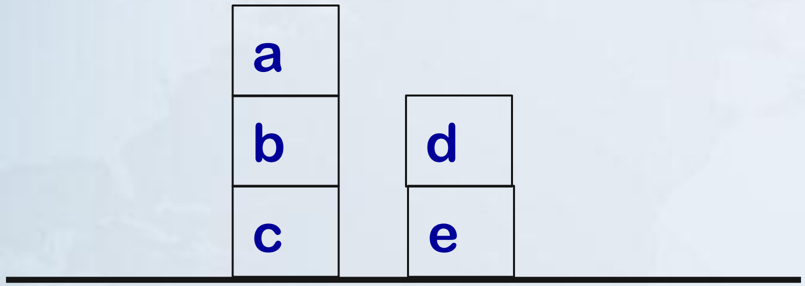
Variable Assignment example

$$\alpha = \{(x \to d_1), (y \to d_2)\}$$

$$I, \alpha \models \text{Red}(x)$$

$$I, \alpha[y/d_1] \models \text{Block}(y)$$

Interpretation (Example)



$Block^{I} = \{\langle a\rangle,\langle b\rangle,\langle c\rangle,\langle d\rangle,\langle e\rangle\}$

$Above^{I} = \{\langle a,b\rangle,\langle b,c\rangle,\langle d,e\rangle\}$

$Clear^{I} = \{\langle a\rangle,\langle d\rangle\}$

$Table^{I} = \{\langle c\rangle,\langle e\rangle\}$

$I \models$ **Block(a)**

$I \not\models$ **Block(f)**

$I \not\models$ **Above(b,e)**

$I \models$ **Above(b,c)**

Semantics of FOL: Satisfiability of formulas

A formula ϕ is satisfied by (*is true in) an interpretation* I *under a variable*

assignment α,

$I, \alpha \models \phi$:

$I, \alpha \models P(t_1,\ldots,t_n)$	iff	$\langle t_1^{I,\alpha},\ldots,t_n^{I,\alpha}\rangle \in P^I$
$I, \alpha \models \neg\phi$	iff	$I, \alpha \not\models \phi$
$I, \alpha \models \phi \wedge \psi$	iff	$I, \alpha \models \phi$ and $I \alpha \models \psi$
$I, \alpha \models \phi \vee \psi$	iff	$I, \alpha \models \phi$ or $I \alpha \models \psi$
$I, \alpha \models \forall x.\phi$	iff	for all $d \in \Delta$: $I, \alpha[x/d] \models \phi$
$I, \alpha \models \exists x.\phi$	iff	there exits a $d \in \Delta$: $I, \alpha[x/d] \models \phi$

Satisfiability and Validity

An interpretation I is a **model** of ϕ under α, if

$$I, \alpha \models \phi$$

Similarly as in propositional logic, a formula ϕ can be **satisfiable, unsatisfiable, falsifiable** or **valid** -the definition is in terms of the pair (I,α).

A formula ϕ is

- **satisfiable**, if there is some (I, α) that satisfies ϕ,
- **un satisfiable**, if ϕ is not satisfiable,
- **falsifiable**, if there is some (I, α) that does not satisfy ϕ,
- **valid** (i.e., a **tautology**), if every (I, α) is a model of ϕ.

Equivalence

Analogously, two formulas are **logically** equivalent ($\phi \equiv \psi$), if for all I, α we have:

$$I, \alpha \models \phi \quad \text{iff} \quad I, \alpha \models \psi$$

Entailment

Entailment is defined similarly as in propositional logic.

The formula ϕ is logically implied by a formula ψ, if ϕ is true in all models of ψ

(symbolically, $\psi \models \phi$):

$$\psi \models \phi \text{ iff } I \models \text{ for all models } I \text{ of } \psi$$

Properties of quantifiers

$(\forall x . \forall y . \phi)$ is the same as $(\forall y . \forall x . \phi)$

$(\exists x . \exists y . \phi)$ is the same as $(\exists y . \exists x . \phi)$

$(\exists x . \forall y . \phi)$ is **not** the same as $(\forall y . \exists x . \phi)$

$\exists x . \forall y . Loves(x,y)$ "There is a person who loves everyone in the world"

$\forall y . \exists x . Loves(x,y)$ "Everyone in the world is loved by at least one person"

Quantifier duality: each can be expressed using the other

$\forall x . Likes(x, Falafel)$ $\neg \exists x . \neg Likes(x, Falafel)$

$\exists x . Likes(x, Salad)$ $\neg \forall x \neg Likes(x, Salad)$

Equivalences

$$(\forall x.\,\phi) \wedge \psi \equiv \forall x.\,(\phi \wedge \psi)$$
$$(\forall x.\,\phi) \vee \psi \equiv \forall x.\,(\phi \vee \psi)$$
$$(\exists x.\,\phi) \wedge \psi \equiv \exists x.\,(\phi \wedge \psi)$$
$$(\exists x.\,\phi) \vee \psi \equiv \exists x.\,(\phi \vee \psi)$$
$$\forall x.\,\phi \wedge \forall x.\,\psi \equiv \forall x.\,(\phi \wedge \psi)$$
$$\exists x.\,\phi \vee \exists x.\,\psi \equiv \exists x.\,(\phi \vee \psi)$$

$$\neg\forall x.\,\phi \equiv \exists x.\,\neg\phi$$
$$\neg\exists x.\,\phi \equiv \forall x.\,\neg\phi$$

& propositional equivalences

Knowledge Engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

A simple genealogy KB (Another Example)

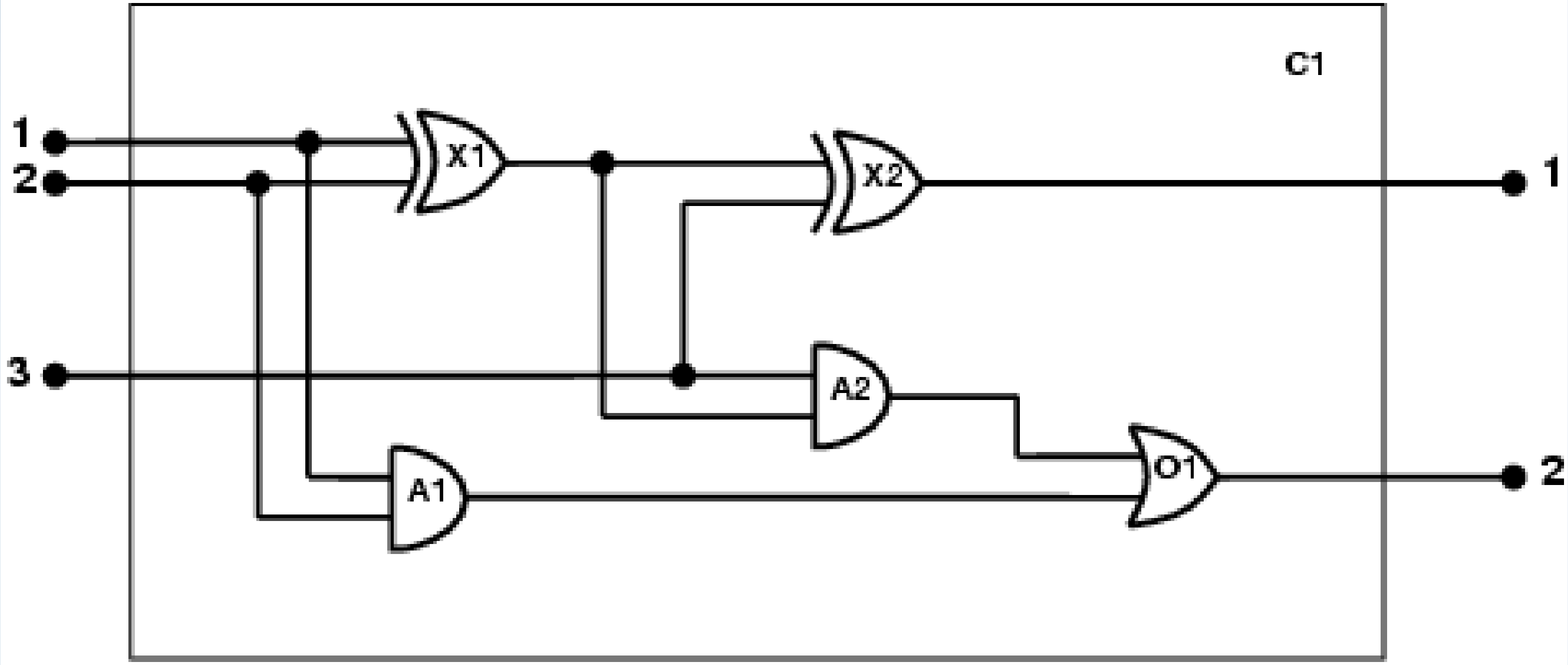
- **Build a small genealogy knowledge base by FOL that**
  - contains facts of immediate family relations (spouses, parents, etc.)
  - contains definitions of more complex relations (ancestors, relatives)
  - is able to answer queries about relationships between people
- **Predicates:**
  - parent(x, y), child (x, y), father(x, y), daughter(x, y), etc.
  - spouse(x, y), husband(x, y), wife(x,y)
  - ancestor(x, y), descendent(x, y)
  - relative(x, y)
- **Facts:**
  - husband(Joe, Mary), son(Fred, Joe)
  - spouse(John, Nancy), male(John), son(Mark, Nancy)
  - father(Jack, Nancy), daughter(Linda, Jack)
  - daughter(Liz, Linda)
  - etc.

A simple genealogy KB (Another Example)

- **Rules for genealogical relations**
  - (∀x,y) parent(x, y) <=> child (y, x)
    (∀x,y) father(x, y) <=> parent(x, y) ^ male(x) (similarly for mother(x, y))
    (∀x,y) daughter(x, y) <=> child(x, y) ^ female(x) (similarly for son(x, y))
  - (∀x,y) husband(x, y) <=> spouse(x, y) ^ male(x) (similarly for wife(x, y))
    (∀x,y) spouse(x, y) <=> spouse(y, x) (**spouse relation is symmetric**)
  - (∀x,y) parent(x, y) => ancestor(x, y)
    (∀x,y)(∃z) parent(x, z) ^ ancestor(z, y) => ancestor(x, y)
  - (∀x,y) descendent(x, y) <=> ancestor(y, x)
  - (∀x,y)(∃z) ancestor(z, x) ^ ancestor(z, y) => relative(x, y)
    (related by common ancestry)
    (∀x,y) spouse(x, y) => relative(x, y) (related by marriage)
    (∀x,y)(∃z) relative(z, x) ^ relative(z, y) => relative(x, y) (**transitive**)
    (∀x,y) relative(x, y) => relative(y, x) (**symmetric**)
- **Queries**
  - ancestor(Jack, Fred) /* the answer is yes */
  - relative(Liz, Joe) /* the answer is yes */
  - relative(Nancy, Mathews)
    /* no answer in general, no if under closed world assumption */

The electronic circuits domain

One-bit full adder



The electronic circuits domain

1. Identify the task
   – Does the circuit actually add properly? (circuit verification)
2. Assemble the relevant knowledge
   – Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)
   – Irrelevant: size, shape, color, cost of gates
3. Decide on a vocabulary
   – Alternatives:

     $Type(X_1) = XOR$

     $Type(X_1, XOR)$

     $XOR(X_1)$

The electronic circuits domain

4. Encode general knowledge of the domain
5. $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Signal(t_1) = Signal(t_2)

- $\forall t$ Signal(t) = 1 $\vee$ Signal(t) = 0
- $1 \neq 0$
- $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Connected(t_2, t_1)
- $\forall g$ Type(g) = OR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 1
- $\forall g$ Type(g) = AND $\Rightarrow$ Signal(Out(1,g)) = 0 $\Leftrightarrow \exists n$ Signal(In(n,g)) = 0
- $\forall g$ Type(g) = XOR $\Rightarrow$ Signal(Out(1,g)) = 1 $\Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
- $\forall g$ Type(g) = NOT $\Rightarrow$ Signal(Out(1,g)) $\neq$ Signal(In(1,g))

The electronic circuits domain

5. Encode the specific problem instance

$Type(X_1) = XOR$

$Type(X_2) = XOR$

$Type(A_1) = AND$

$Type(A_2) = AND$

$Type(O_1) = OR$

$Connected(Out(1,X_1),In(1,X_2))$

$Connected(Out(1,X_1),In(2,A_2))$

$Connected(Out(1,A_2),In(1,O_1))$

$Connected(Out(1,A_1),In(2,O_1))$

$Connected(Out(1,X_2),Out(1,C_1))$

$Connected(Out(1,O_1),Out(2,C_1))$

$Connected(In(1,C_1),In(1,X_1))$

$Connected(In(1,C_1),In(1,A_1))$

$Connected(In(2,C_1),In(2,X_1))$

$Connected(In(2,C_1),In(2,A_1))$

$Connected(In(3,C_1),In(2,X_2))$

$Connected(In(3,C_1),In(1,A_2))$

The electronic circuits domain

6. Pose queries to the inference procedure
7. What are the possible sets of values of all the terminals for the adder circuit?
8. $\exists i_1,i_2,i_3,o_1,o_2$ Signal(In(1,C_1)) = $i_1 \wedge$ Signal(In(2,C_1)) = $i_2 \wedge$ Signal(In(3,C_1)) = $i_3 \wedge$ Signal(Out(1,C_1)) = $o_1 \wedge$ Signal(Out(2,C_1)) = o_2

7. Debug the knowledge base

   May have omitted assertions like $1 \neq 0$

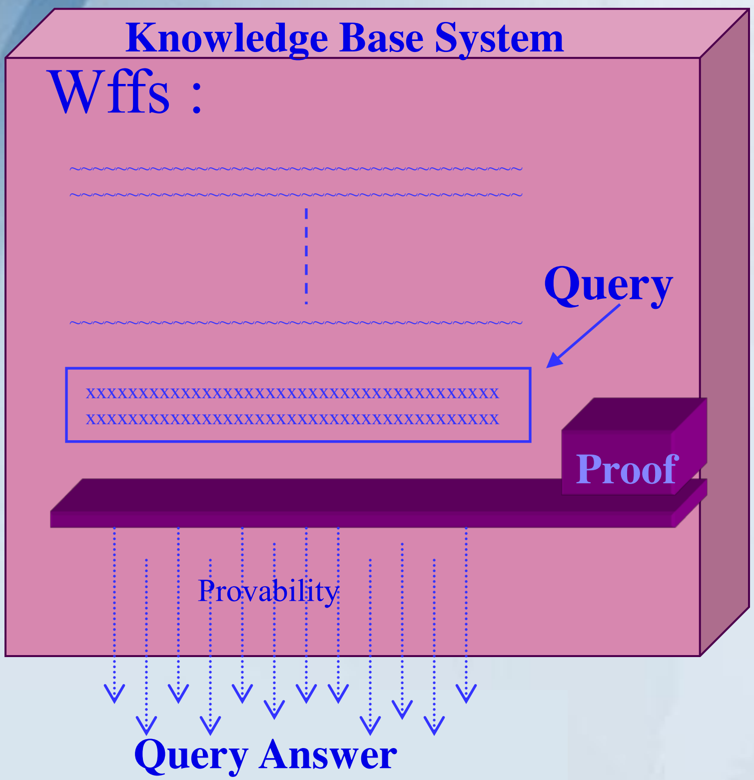
Summary

- First-order logic:
  - objects and relations are semantic primitives
  - syntax: constants, functions, predicates, equality, quantifiers
- Increased expressive power: sufficient to define Wumpus world

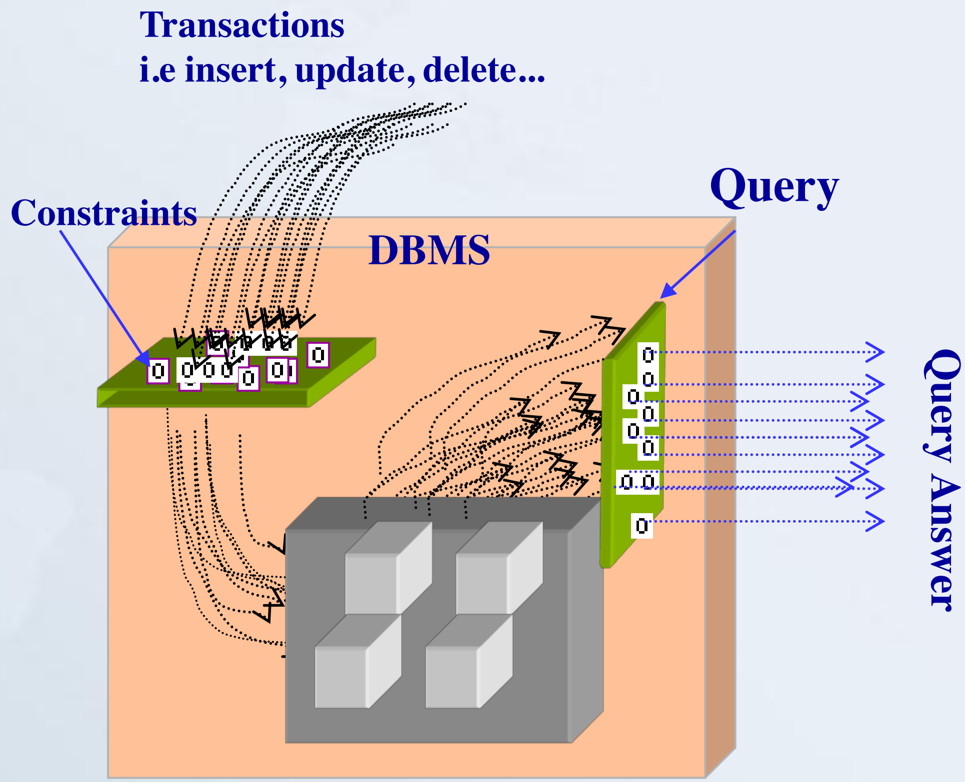
Outline

- FOL, First Order Logic
  - Motivation (why FOL)
  - Syntax
  - Semantics
- FOL Inference Methods
  - Enumeration Method
  - Inference rules
  - Resolution
  - Forward and backward Chaining

Knowledge Bases vs. Databases



Proof Theoretic View



Model Theoretic View

The KB is a set of formulae and the query evaluation is to prove that the result is provable.

Evaluating the truth formula for each tuple in the table "Publish"

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward chaining
- Backward chaining
- Resolution

Inference in First-Order Logic

- We may inference in FOL by mapping FOL sentences into propositions, and apply the inference methods of propositional logic.
- This mapping is called **propositionalization.**
- Thus, Inference in first-order logic can be achieved using:
  - Inference rules
  - Forward chaining
  - Backward chaining
  - Resolution
    - Unification
    - Proofs
    - Clausal form
    - Resolution as search

Universal Instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

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

  for any variable *v* and ground term *g*

- **Example:**

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

⇓

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

Existential Instantiation (EI)

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

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

Example:

$\exists x\ Crown(x) \wedge OnHead(x, John)$

⇓

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

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

- The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only.
- In other words, we don't want to accidentally draw other inferences about it by introducing the constant.
- Convenient to use this to reason about the unknown object, rather than constantly manipulating the existential quantifier.

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)

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

- 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
- (A ground sentence is entailed by new KB iff entailed by original KB)
- Idea: 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

create a propositional KB by instantiating with depth-n terms
see if α is entailed by this KB

Problem: works if α is entailed, loops if α is not entailed.

Godel's Completeness Theorem says that FOL entailment is only semidecidable:

- If a sentence is **true** given a set of axioms, there is a procedure that will determine this.
- If the sentence is **false**, then there is no guarantee that a procedure will ever determine this–i.e., it **may never halt.**

Completeness of some inference techniques

- **Truth Tabling** is not complete for FOL because truth table size may be infinite.
- **Natural Deduction** is complete for FOL but is not practical because the "branching factor" in the search is too large (so we would have to potentially try every inference rule in every possible way using the set of known sentences).
- **Generalized Modus Ponens** is not complete for FOL.
- **Generalized Modus Ponens** is complete for KBs containing only Horn clauses.

Problems with Propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
  E.g., from:

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

  King(John)

  $\forall y$ Greedy(y)

  Brother(Richard, John)

- It seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant

- With *p* *k*-ary predicates and *n* constants, there are $p \cdot n^k$ instantiations.

Problems with Propositionalization

Given this KB:

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

How do we really know that Evil(John)?

- We find x that is a King and Greedy, if so then x is Evil.
- That is, we need to a substitution {x/John}

But Given this KB:

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

How do we really know that Evil(John)?

- That is, we need to the substitutions {x/John, y,John}, but how?

Unification

- We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

  θ = {x/John,y/John}

- This called **Unification**, a “pattern-matching” procedure:
  - Takes two atomic sentences, called literals, as input
  - Returns “Failure” if they do not match and a substitution list, θ, if they do

  Unify(P,Q) = θ if Pθ = Qθ

- That is, *unify(p,q) = θ* means *subst(θ, p) = subst(θ, q)* for two atomic sentences, *p* and *q*
- **θ** is called the **Most General Unifier** (MGU)
- All variables in the given two literals are implicitly universally quantified.
- To make literals match, replace (universally quantified) variables by terms

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,Bill)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,Elizabeth)	

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	**Knows(John,Jane)**	**{x/Jane}**
Knows(John,x)	**Knows(y,Bill)**	
Knows(John,x)	**Knows(y,Mother(y))**	
Knows(John,x)	**Knows(x,Elizabeth)**	

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	**Knows(John,Jane)**	**{x/Jane}**
Knows(John,x)	**Knows(y,Bill)**	**{x/Bill,y/John}**
Knows(John,x)	**Knows(y,Mother(y))**	
Knows(John,x)	**Knows(x,Elizabeth)**	

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	**Knows(John,Jane)**	**{x/Jane}**
Knows(John,x)	**Knows(y,Bill)**	**{x/Bill,y/John}**
Knows(John,x)	**Knows(y,Mother(y))**	**{y/John,x/Mother(John)}**
Knows(John,x)	**Knows(x,Elizabeth)**	

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	**Knows(John,Jane)**	**{x/Jane}**
Knows(John,x)	**Knows(y,Bill)**	**{x/Bill,y/John}**
Knows(John,x)	**Knows(y,Mother(y))**	**{y/John,x/Mother(John)}**
Knows(John,x)	**Knows(x,Elizabeth)**	**fail**

- The last unification failed because x cannot take on the values John and Elizabeth at the same time.
- Because it happens that both sentences use the same variable name.
- Solution: rename x in Knows(x,Elizabeth) into Knows(z_{17},Elizabeth) , without changing its meaning. (this is called **standardizing apart**)

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	Knows(John,Jane)	**{x/Jane}**
Knows(John,x)	Knows(y,Bill)	**{x/Bill,y/John}**
Knows(John,x)	Knows(y,Mother(y))	**{y/John,x/Mother(John)}**
Knows(John,x)	Knows(z_{17},Elizabeth)	**{x/Elizabeth, z_{17}/John}**

- The last unification failed because x cannot take on the values John and Elizabeth at the same time.
- Because it happens that both sentences use the same variable name.
- Solution: rename x in Knows(x,Elizabeth) into Knows(z17,Elizabeth) , without changing its meaning. (this is called **standardizing apart**)

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	**Knows(John,Jane)**	**{x/Jane}**
Knows(John,x)	**Knows(y,Bill)**	**{x/Bill,y/John}**
Knows(John,x)	**Knows(y,Mother(y))**	**{y/John,x/Mother(John)}**
Knows(John,x)	**Knows(z_{17},Elizabeth)**	**{x/Elizabeth, z_{17}/John}**

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	Knows(John,Jane)	**{x/Jane}**
Knows(John,x)	Knows(y,Bill)	**{x/Bill,y/John}**
Knows(John,x)	Knows(y,Mother(y))	**{y/John,x/Mother(John)}**
Knows(John,x)	Knows(z_{17},Elizabeth)	**{x/Elizabeth, z_{17}/John}**
Knows(John,x)	Knows(y,z)	**??**

In the last case, we have two answers:

θ= **{y/John,x/z}, or**

θ= **{y/John,x/John, z/John}**

This first unification is more genera as it places fewer restrictions on th values of the variables.

Unification Example

Unify (p,q) = θ where Subst(θ,p) = Subset(θ,q)

Suppose we have a query Knows(John,x), we need to unify Knows(John,x) with all sentences in KD.

P	Q	θ
Knows(John,x)	Knows(John,Jane)	**{x/Jane}**
Knows(John,x)	Knows(y,Bill)	**{x/Bill,y/John}**
Knows(John,x)	Knows(y,Mother(y))	**{y/John,x/Mother(John)}**
Knows(John,x)	Knows(z_{17},Elizabeth)	**{x/Elizabeth, z_{17}/John}**
Knows(John,x)	Knows(y,z)	**{y/John,x/z}**

In the last case, we have two answers:

θ= **{y/John,x/z}, or**

θ= **{y/John,x/John, z/John}**

For every unifiable pair of expressions, there is a **Most General Unifier MGU**

Another Example

- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(Bill), y)
  - {x/Bill, y/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(y), z)
  - {x/Bill, y/Bill, z/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Jane))
  - parents(Bill, father(y), mother(y))
  - Failure

Generalized Modus Ponens (GMP)

- **A first-order inference rule, to find substitutions easily.**
- Apply modus ponens reasoning to generalized rules.
- Combines And-Introduction, Universal-Elimination, and Modus Ponens . Example: $\{P(c), Q(c), \forall x(P(x) \wedge Q(x)) \Rightarrow R(x)\}$ derive $R(c)$
- General case: Given
  - **Atomic sentences** $P_1, P_2, ..., P_n$
  - **Implication sentence** $(Q_1 \wedge Q_2 \wedge ... \wedge Q_n) \Rightarrow R$
    - $Q_1, ..., Q_n$ and R are atomic sentences
  - **Substitution** subst(θ, P_i) = subst(θ, Q_i) (for i=1,...,n)
  - **Derive new sentence:** subst(θ, R)
- Substitutions
  - subst(θ, α) denotes the result of applying a set of substitutions defined by θ to the sentence α
  - A substitution list θ = $\{v_1/t_1, v_2/t_2, ..., v_n/t_n\}$ means to replace all occurrences of variable symbol v_i by term t_i
  - Substitutions are made in left-to-right order in the list

Generalized Modus Ponens (GMP)

A first-order inference rule, to find substitutions easily.

$$\frac{P_1, P_2, \ldots, P_n, \ (Q_1 \wedge Q_2 \wedge \ldots \wedge Q_n \Rightarrow R)}{\text{Subst}\ (R, \theta)}$$

where $P_i\,\theta = Q_i\,\theta$ **for all** i

$$\frac{King(John),\ Greedy(y),\quad (King(x),\ Greedy(x) \Rightarrow Evil(x))}{Subst(Evil(x),\ \{x/John,\ y/John\})}$$

- GMP used with KB of definite clauses (exactly one positive literal).
- All variables assumed universally quantified.

Soundness of GMP

Need to show that

$$P_1, \ldots, P_n, (Q_1 \wedge \ldots \wedge Q_n \Rightarrow Q) \models R\theta$$

provided that $Pi\,\theta = Qi\,\theta$ for all i

Lemma: For any sentence Q, we have $Q \models Q\theta$ by UI

$(P_1 \wedge \ldots \wedge P_n \Rightarrow \mathrm{R}) \models (P_1 \wedge \ldots \wedge p_\mathrm{n} \Rightarrow R)\,\theta = (P_1\theta \wedge \ldots \wedge P_n\theta \Rightarrow R\,\theta)$

$\mathrm{Q}_1 \backslash \ldots, \backslash P_n \models Q_1 \wedge \ldots \wedge Q_n \models P_1\theta \wedge \ldots \wedge Q_n\theta$

From 1 and 2, R θ follows by ordinary Modus Ponens

Forward Chaining

- Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived

- This defines a **forward-chaining** inference procedure because it moves “forward” from the KB to the goal

- Natural deduction using GMP is **complete** for KBs containing **only Horn clauses**

Example Knowledge Base

- 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.
- 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) $\wedge$ *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):

$Owns(Nono,M_1) \wedge Missile(M_1)$

… 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)*

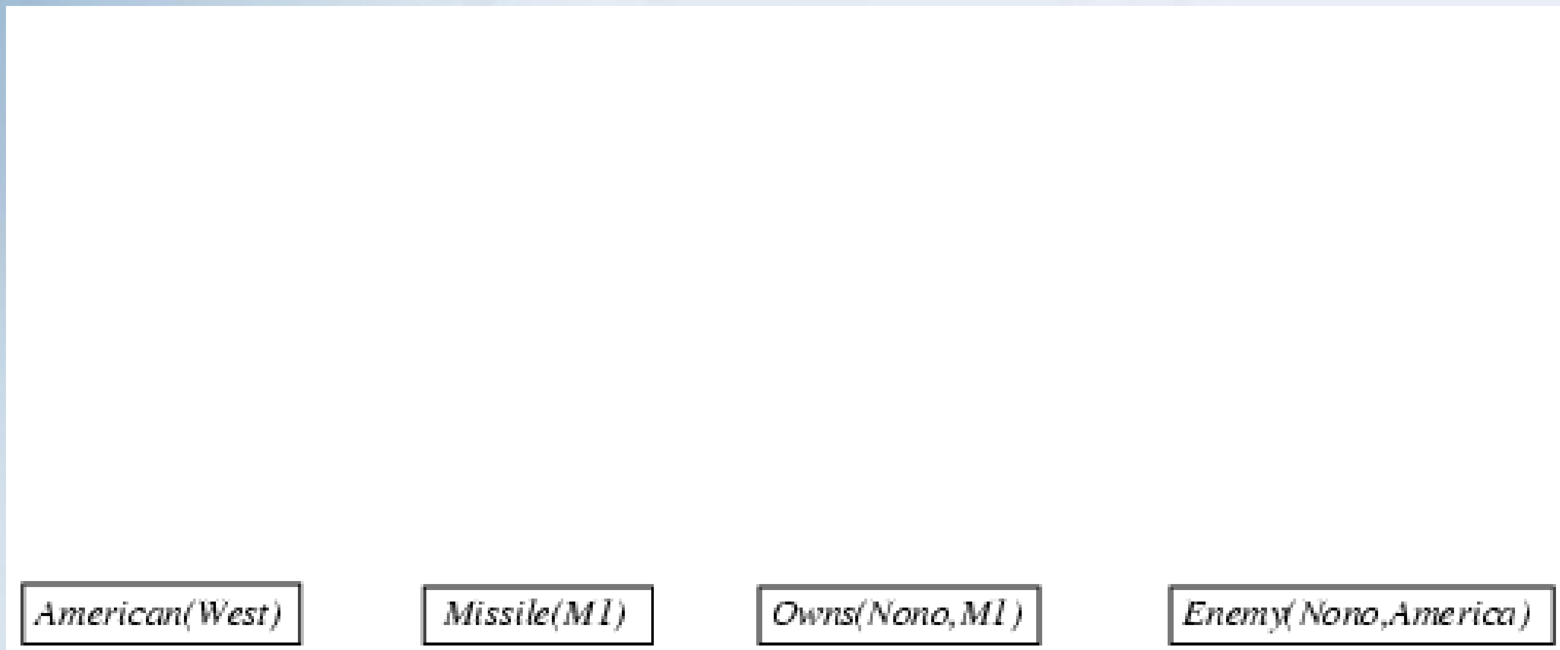
West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

Forward Chaining Proof



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

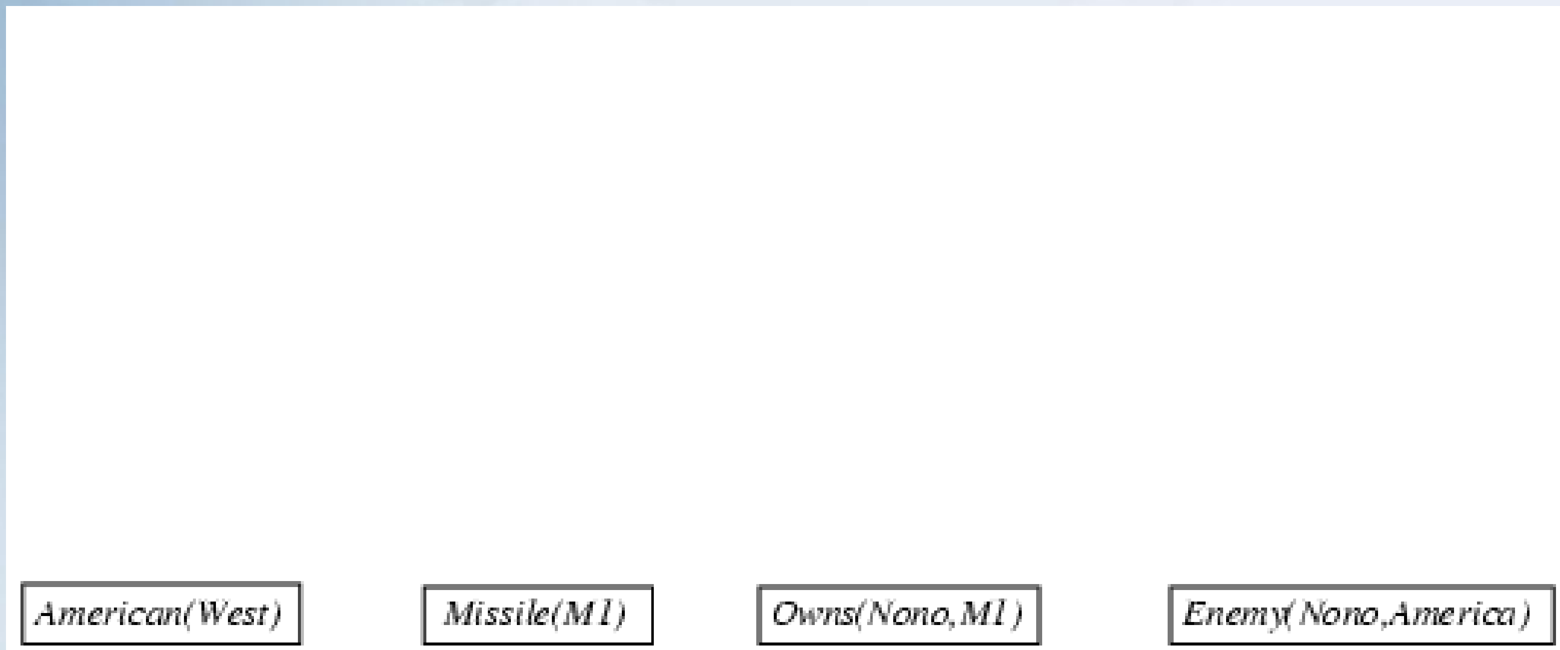
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Forward Chaining Proof



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

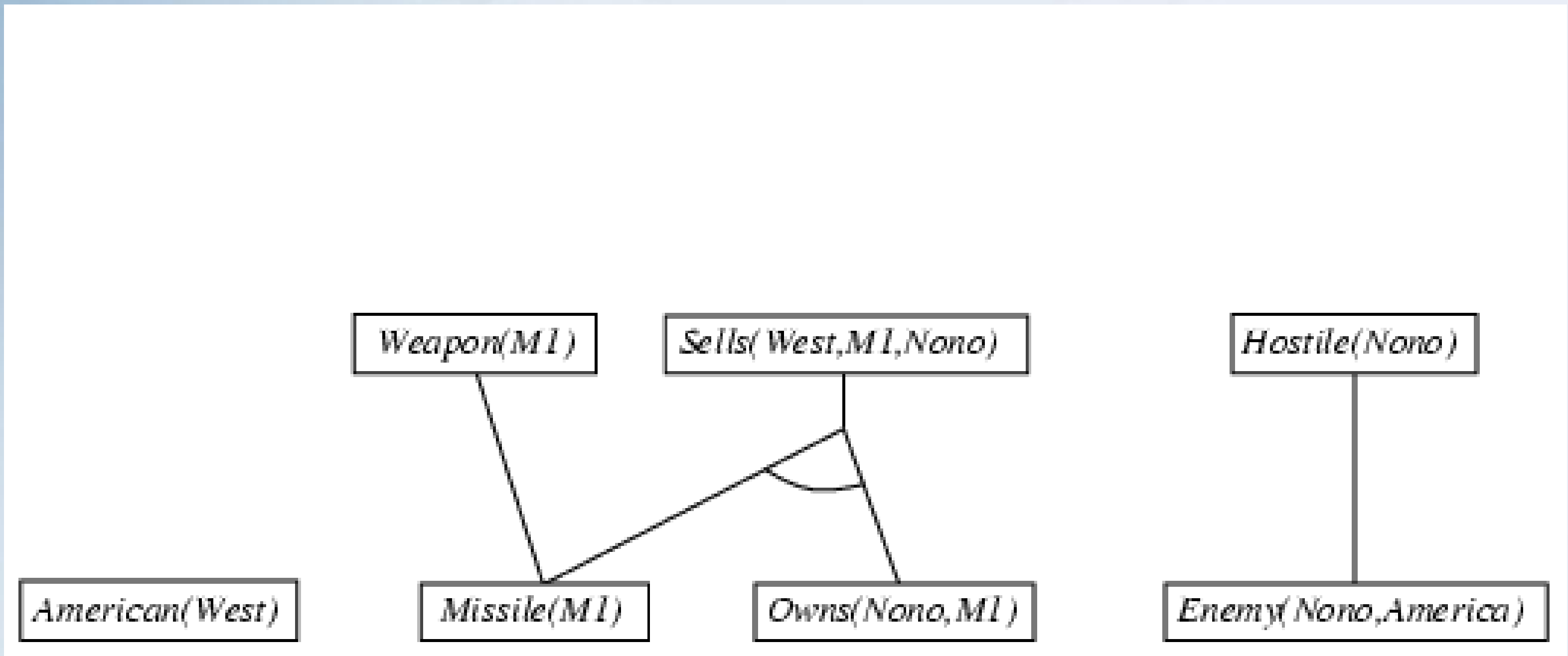
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Forward Chaining Proof



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

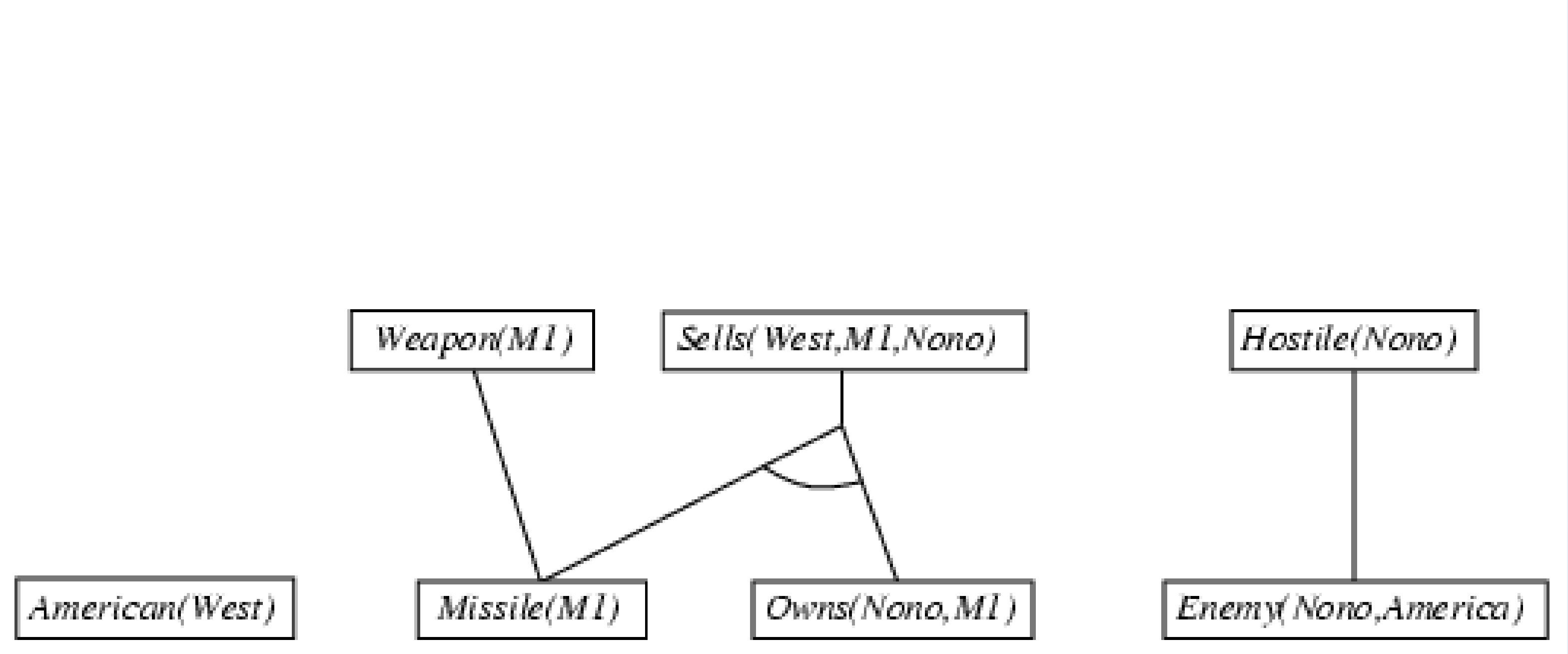
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Forward Chaining Proof



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

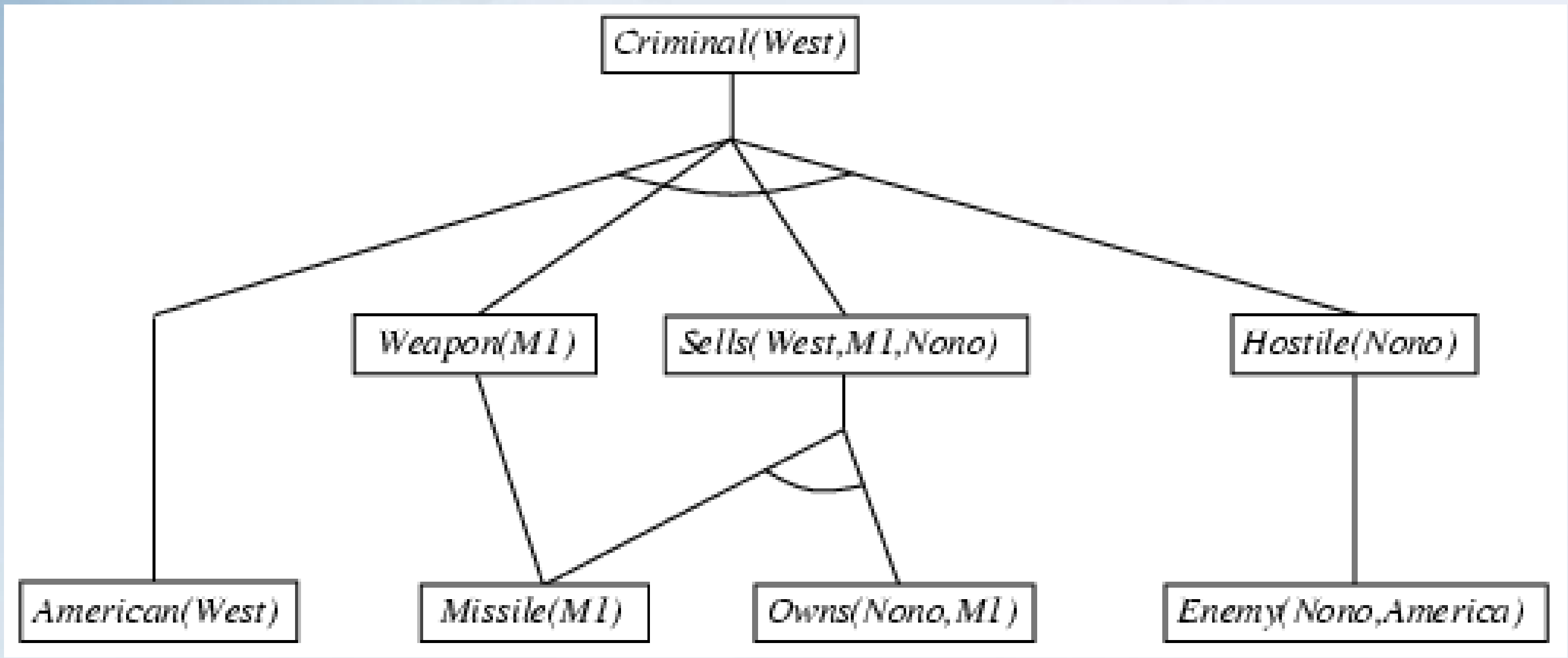
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Forward Chaining Proof



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

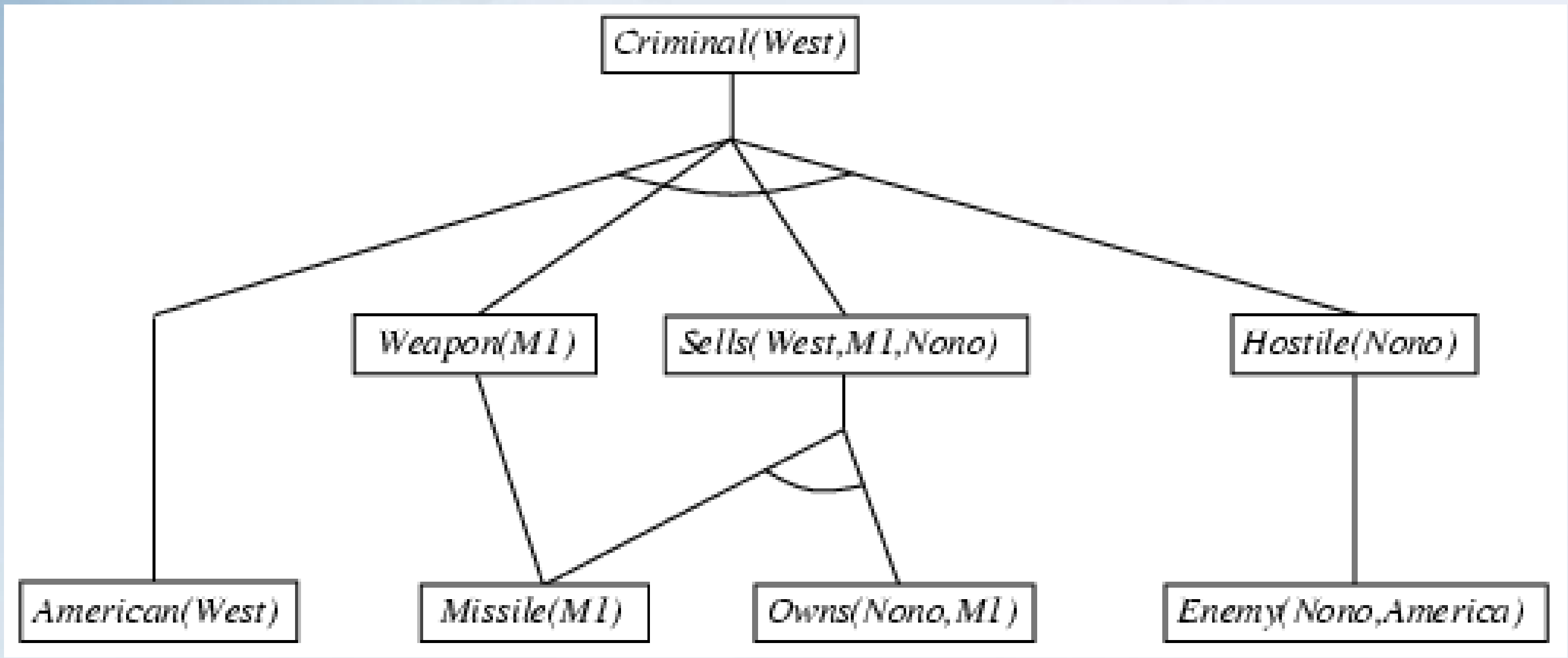
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Forward Chaining Proof



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

Owns(*Nono*,M_1) ∧ *Missile*(M_1)

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

Missile(*x*) ⇒ *Weapon*(*x*)

Enemy(*x*,*America*) ⇒ *Hostile*(*x*)

American(*West*)

Enemy(*Nono*,*America*)

Properties of Forward Chaining

- **Sound** and **complete** for first-order definite clauses.

- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations.

- May not terminate in general if α is not entailed.

- This is unavoidable: entailment with definite clauses is semidecidable.

Efficiency of Forward Chaining

Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*

$\Rightarrow$ Match each rule whose premise contains a newly added positive literal.

Matching itself can be expensive:

Database indexing allows O(1) retrieval of known facts

e.g., query *Missile(x)* retrieves $Missile(M_1)$

Forward chaining is widely used in deductive databases.

Backward Chaining

- Proofs start with the goal query, find implications that would allow you to prove it, and then prove each of the antecedents in the implication, continuing to work "backwards" until you arrive at the axioms, which we know are true.
- Backward-chaining deduction using GMP is **complete** for KBs containing **only Horn clauses.**

Backward chaining example

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \Rightarrow$ $\underline{Criminal(x)}$

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

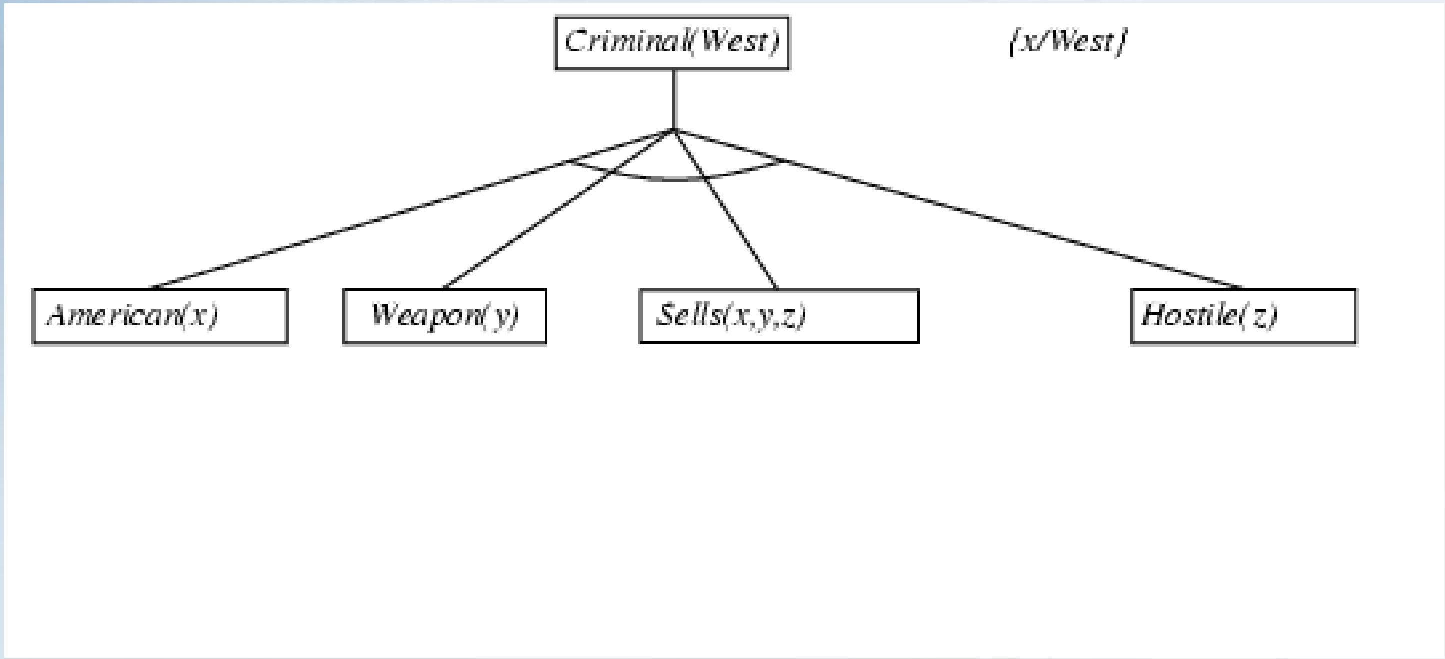
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Backward chaining example



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

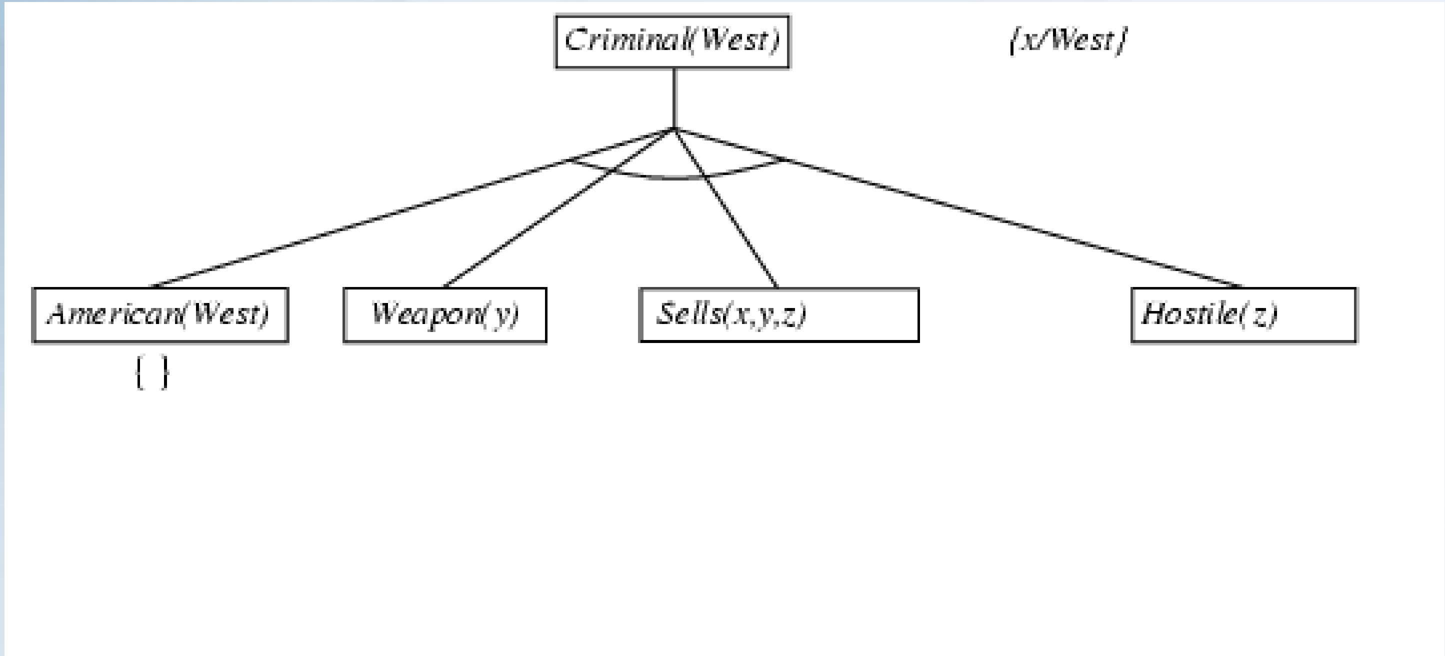
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Backward chaining example



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

Owns($Nono,M_1$) ∧ *Missile*(M_1)

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

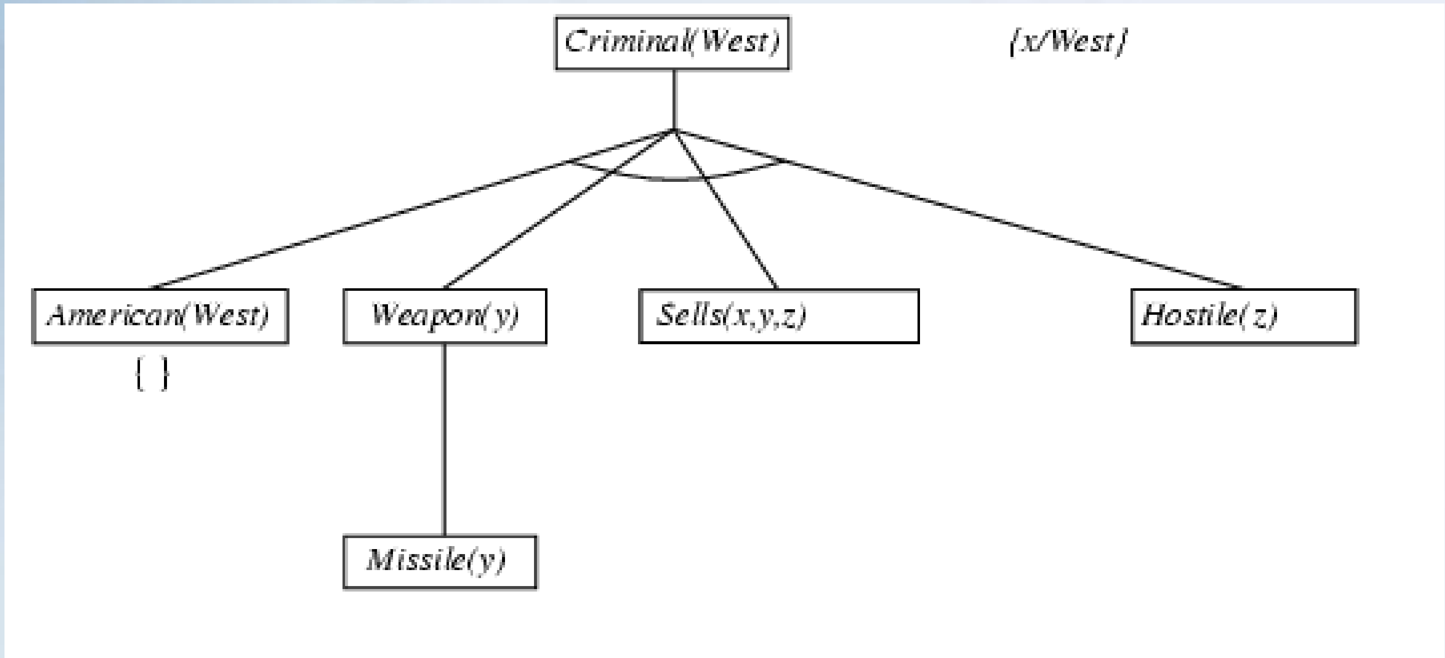
Missile(x) ⇒ *Weapon*(x)

Enemy($x,America$) ⇒ *Hostile*(x)

American($West$)

Enemy($Nono,America$)

Backward chaining example



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

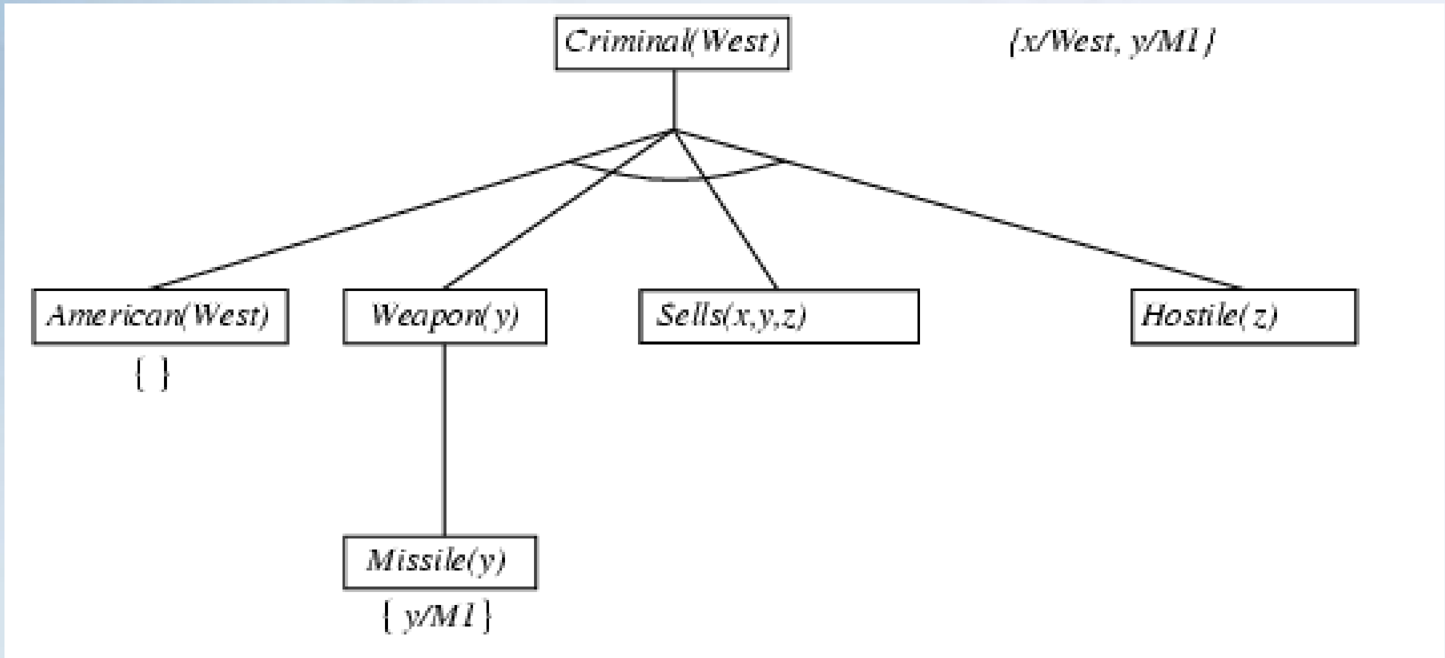
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Backward chaining example



$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \Rightarrow$ **$\underline{Criminal(x)}$**

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

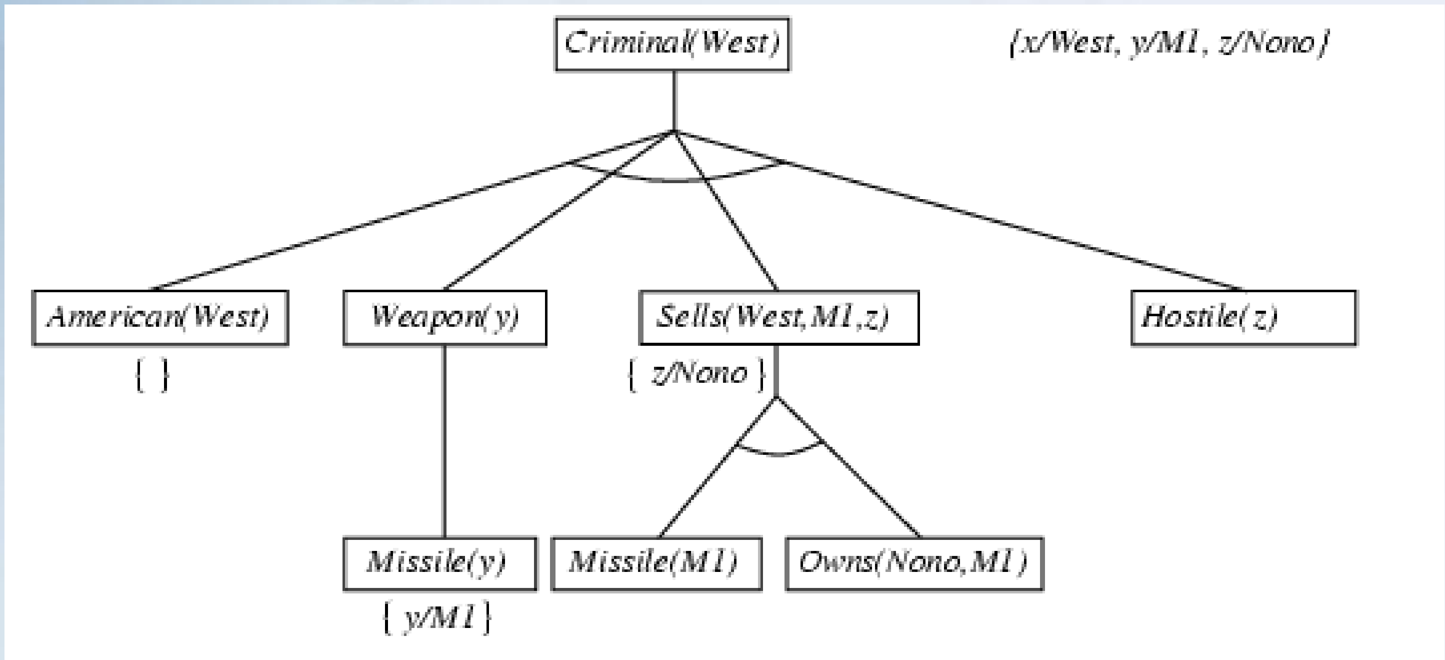
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Backward chaining example



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

$Owns(Nono,M_1) \wedge Missile(M_1)$

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

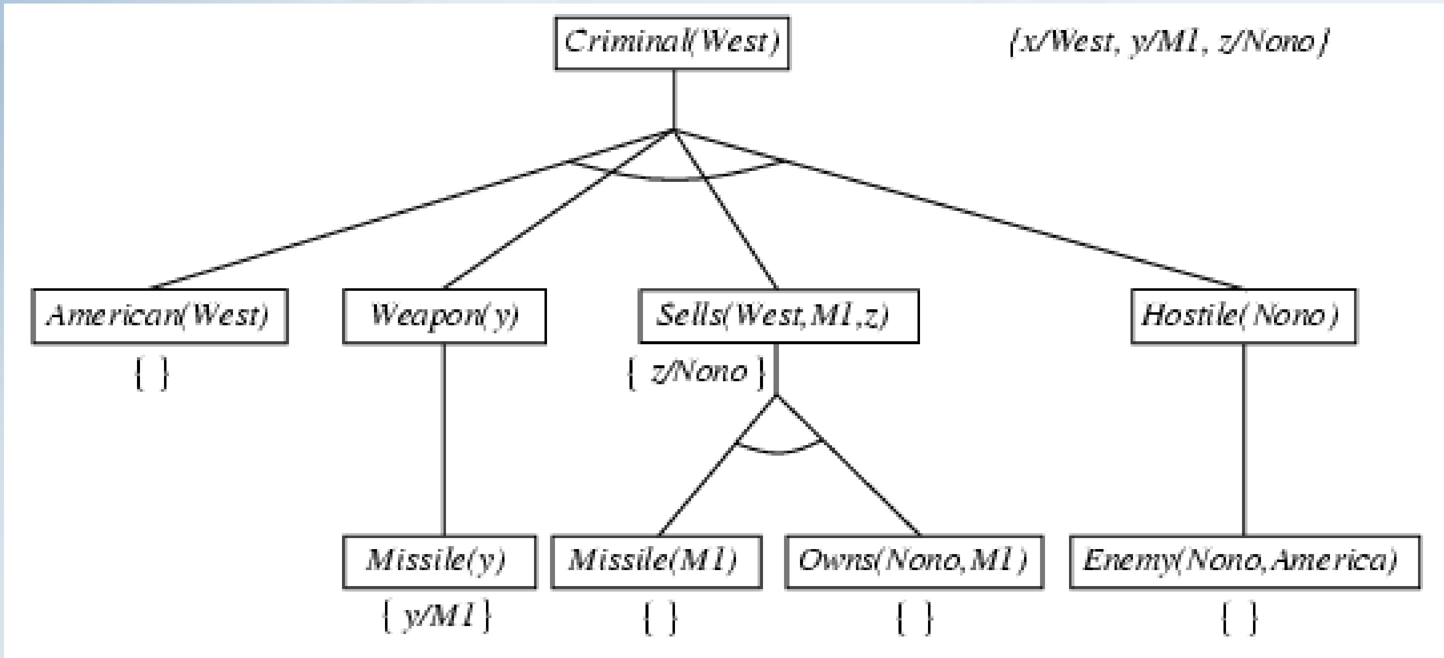
$Missile(x) \Rightarrow Weapon(x)$

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

$American(West)$

$Enemy(Nono,America)$

Backward chaining example



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

Owns(*Nono*,M_1) ∧ *Missile*(M_1)

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

Missile(*x*) ⇒ *Weapon*(*x*)

Enemy(*x*,*America*) ⇒ *Hostile*(*x*)

American(*West*)

Enemy(*Nono*,*America*)

Properties of Backward Chaining

- Depth-first recursive proof search: space is linear in size of proof.
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack.
- Inefficient due to repeated subgoals (both success and failure).
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming.

Forward vs. Backward Chaining

- FC is data-driven
  - Automatic, unconscious processing
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
  - More efficient when you want to compute all conclusions.
- BC is goal-driven, better for problem-solving
  - Where are my keys? How do I get to my next class?
  - Complexity of BC can be much less than linear in the size of the KB
  - More efficient when you want one or a few decisions.

Logic Programming

- Algorithm = Logic + Control
- A backward chain reasoning theorem-prover applied to declarative sentences in the form of implications:

  If B_1 and … and B_n then H

- Implications are treated as goal-reduction procedures:

  to show/solve H, show/solve B_1 and … and B_n.

  where implication would be interpreted as a solution of problem H given solutions of B_1 … B_n.

- Find a solution is a proof search, which done Depth-first backward chaining.
- Because automated proof search is generally infeasible, logic programming relies on the programmer to ensure that inferences are generated efficiently. Also by restricting the underlying logic to a "well-behaved" fragment such as Horn clauses or Hereditary Harrop formulas.

Logic Programming: Prolog

Developed by Alain Colmerauer(Marseille) and Robert Kowalski(Edinburgh) in 1972.

Program = set of clauses of the form

$$P(x)_1 \wedge \ldots \wedge p(x_n) \Rightarrow \text{head}$$

written as

`head :- P(x_1), … , P(x_n).`

For example:

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

Closed-world assumption ("negation as failure").

- `alive(X) :- not dead(X).`
- `alive(joe)` succeeds if `dead(joe)` fails.

Logic Programming: Prolog

```
mother(Nuha, Sara).
father(Ali, Sara).
father(Ali, Dina).
father(Said, Ali).
sibling(X, Y) :- parent(Z, X), parent(Z, Y).
parent (X, Y) :- father(X, Y).
parent(X, Y) :- mother (X, Y).
```

```
?- sibling(Sara, Dina).
Yes
```

```
?- father(Father, Child).
```

// enumerates all valid answers

Resolution in FOL

Resolution in FOL

- Recall: We saw that the propositional resolution is a refutationly **complete** inference procedure **for Propositional Logic**.
- Here, we extend resolution to FOL.
- First we need to covert sentences in to CNF, for example:

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

- becomes

$$\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z) \vee Criminal(x)$$

- Every sentence of first-order logic can be converted into **inferentially equivalent** CNF sentence.
- The procedure for conversion to CNF is similar to the propositional case.

Conversion to CNF

- **The procedure for conversion to CNF is similar to the positional case.**
- For example: "Everyone who loves all animals is loved by someone", or

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

Step 1 Eliminate Implications

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

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

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

Conversion to CNF contd.

Step 2. Move ¬ inwards:

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

Step 3. Standardize variables: each quantifier should use a different one

$$\forall x\ [\exists y\ Animal(y) \land \neg Loves(x,y)] \lor [\exists z\ Loves(z,x)]$$

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

$$\forall x\ [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

Step 5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

Step 6. Distribute ∨ over ∧ :

$$[Animal(F(x)) \lor Loves(G(x),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(x),x)]$$

Resolution in FOL

- The inference rule (FOL 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`$(\ell_i, \neg m_j) = \theta$.

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

- Apply resolution steps to CNF(KB $\wedge \neg\alpha$).

- Let's extend the previous example, and apply the resolution:
  Everyone who loves all animals is loved by someone.
  Anyone who kills an animal is loved by no one.
  Ali loves all animals.
  Either Ali or Kais killed the cat, who is named Foxi.
  Did Kais killed the cat?

Resolution in FOL (Example)

Let's extend the previous example, and apply the resolution:

Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Ali loves all animals.
Either Ali or Kais killed the cat, who is an animal and its is named Foxi.
Did Kais killed the cat?

In FOL:

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
B. $\forall x\ [\exists y\ Animal(y) \Rightarrow Kills(x,y)] \Rightarrow [\exists z\ \neg Loves(z,x)]$
C. $\forall x\ Animal(x) \Rightarrow Loves(Ali,x)$
D. $Kills\ (Ali,Foxi) \vee Kills(Kais,x)$
E. $Cat(Foxi)$
F. $\forall x\ Cat(x) \Rightarrow Animal\ (x)$
$\neg$G. $\neg Kills(Kais,Foxi)$

Resolution in FOL (Example)

A. $\forall x\ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ Loves(y,x)]$
B. $\forall x\ [\exists y\ Animal(y) \Rightarrow Kills(x,y)] \Rightarrow [\exists z\ \neg Loves(z,x)]$
C. $\forall x\ Animal(x) \Rightarrow Loves(Ali,x)$
D. $Kills(Ali,Foxi) \lor Kills(Kais,x)$
E. $Cat(Foxi)$
F. $\forall x\ Cat(x) \Rightarrow Animal(x)$
¬G. $\neg Kills(Kais,Foxi)$

After applying the CNF, we obtain:

A1. $Animal(F(x)) \lor Loves(G(x),x)$
A2. $\neg Loves(x,F(x)) \lor Loves(G(x),x)$
B. $\neg Animal(y) \lor Kills(x,y) \lor \neg Loves(z,x)$
C. $\neg Animal(x)\ Cat(Foxi)\ Loves(Ali,x)$
D. $Kills(Ali,Foxi) \lor Kills(Kais, Foxi)$
E. $Cat(Foxi)$
F. $\neg Cat(x) \lor Animal(x)$
¬G. $\neg Kills(Kais,Foxi)$

Resolution in FOL (Example)

A1	Animal(F(x)) ∨ Loves(G(x),x)		
A2	¬Loves(x,F(x)) ∨ Loves(G(x),x)		
B	¬Loves(y,x) ∨ ¬Animal(z) ∨ ¬Kills(x,z)		
C	¬Animal(x) Cat(Foxi) Loves(Ali,x)		
D	Kills(Ali,Foxi) ∨ Kills(Kais, Foxi)		
E	Cat(Foxi)		
F	¬Cat(x) ∨ Animal (x)		
G	¬Kills(Kais,Foxi)		
H	Animal (Foxi)	E,F	{x/Foxi}
I	Kills(Ali,Foxi)	D,G	{}
J	¬Animal(F(Ali)) ∨ Loves(G(Ali), Ali)	A2,C	{x/Ali, F(x)/x}
K	Loves(G(Ali), Ali)	J,A1	{F(x)/F(Ali), X/Ali}
L	¬Loves(y,x) ∨ ¬Kills(x,Foxi)	H,B	{z/Foxi}
M	¬Loves(y,Ali)	I,L	{x/Ali}
N	.	M,K	{y/G(Ali)}

Resolution in FOL (Another Example)

- 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.

- Prove that Col. West is a criminal
- Assume this is represented in FOL (and in CNF):

$\neg$ American(x) $\vee$ $\neg$Weapon(y) $\vee$ $\neg$Sells(x,y,z) $\vee$ $\neg$Hostile(z) $\vee$ Criminal(x)
$\neg$Missile(x) $\vee$ $\neg$Owns(Nono,x) $\vee$ Sells(West,x,Nano)
$\neg$Enemy(x,America) $\vee$ Hostile(x)
$\neg$Missile(x) $\vee$ Weapon(x)
Owns(Nono,M_1)
Missile(M_1)
American(West)
Enemy(Nano,America)
$\neg$Criminal (West)

Resolution in FOL (Another Example)

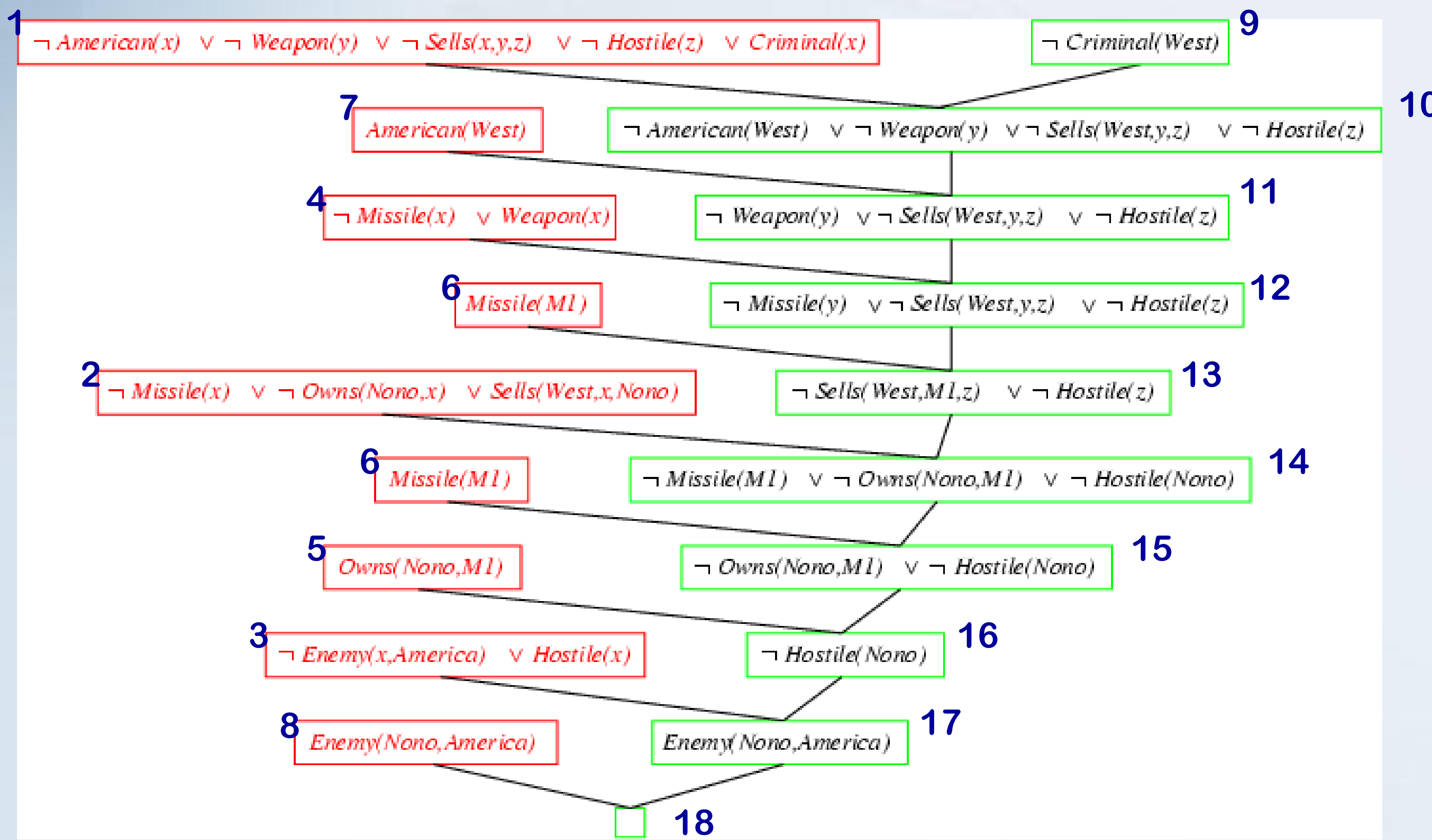
1	¬ American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x,y,z) ∨ ¬Hostile(z) ∨ Criminal(x)		
2	¬Missile(x) ∨ ¬Owns(Nono,x) ∨ Sells(West,x,Nano)		
3	¬Enemy(x,America) ∨ Hostile(x)		
4	¬Missile(x) ∨ Weapon(x)		
5	Owns(Nono,M_1)		
6	Missile(M_1)		
7	American(West)		
8	Enemy(Nano,America)		
9	¬Criminal (West)		

Resolution in FOL (Another Example)

1	¬ American(x) ∨ ¬Weapon(y) ∨ ¬Sells(x,y,z) ∨ ¬Hostile(z) ∨ Criminal(x)		
2	¬Missile(x) ∨ ¬Owns(Nono,x) ∨ Sells(West,x,Nano)		
3	¬Enemy(x,America) ∨ Hostile(x)		
4	¬Missile(x) ∨ Weapon(x)		
5	Owns(Nono,M_1)		
6	Missile(M_1)		
7	American(West)		
8	Enemy(Nano,America)		
9	¬Criminal (West)		
10	¬ American(West) ∨ ¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)	1,9	{x/West}
11	¬Weapon(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)	7,10	{x/West}
12	¬Missile(y) ∨ ¬Sells(West,y,z) ∨ ¬Hostile(z)	4,11	{x/y}
13	¬Sells(West,M_1,z) ∨ ¬Hostile(z)	6,12	{y/M_1}
14	¬Missile(M_1) ∨ ¬Owns(Nono, M_1) ∨ ¬Hostile(Nano)	2,13	{x/M_1, z/Nano}
15	¬Owns(Nono, M_1) ∨ ¬Hostile(Nano)	6,14	{}
16	¬Hostile(Nano)	5,15	{}
17	¬Enemy(Nano,America)	3,16	{x/Nano}
18	.	8,17	{}

Resolution in FOL (Another Example)

Another representation (as Tree)



Summary

- **Instantiating quantifiers** is typically very low.
- **Unification** is much more efficient than Instantiating quantifiers.
- **Generalized Modus Ponens** = Modus Ponens + unification, which is then used in forward/backward chaining.
- **Generalized Modus Ponens** is complete but semidecidable**.**
- **Forward chaining** is complete, and used in deductive databases, and Datalogs with polynomial time.
- **Backward chaining** is complete, used in logic programming, suffers from redundant inference and infinite loops.
- Generalized **Resolution** is refutation complete for sentences with CNF.
- There are **no decidable** inference methods for FOL.
- The exam will evaluate: What\How\Why (for all above)
- Next Lecture: Description logics are decidable logics.