Part 6: Knowledge Representation and Reasoning II: First-Order Logic

(First-order Predicate Calculus; Predicate Logic)

Univ.-Prof. Dr. Gerhard Widmer
Department of Computational Perception
Johannes Kepler University Linz

gerhard.widmer@jku.at
http://www.cp.jku.at/people/widmer

Overview

Motivation: Limitations of propositional logic

First-order logic: Syntax and Semantics

Representing knowledge with first-order logic

Reasoning in first-order logic

- A specific first-order inference rule: modus ponens
- Substitutions and unification
- A general (and sound and complete) inference procedure: resolution in first-order logic (only briefly)
- Efficient inference in restricted logics
    - first-order Horn clauses
    - forward chaining
    - backward chaining

First-order Horn clause logic as a general programming language: PROLOG

Propositional Logic: Pros and Cons

- + Propositional logic is **declarative** (i.e., permits to represent knowledge directly)
- + Propositional logic allows expressing **partial / disjunctive / negated information** (unlike most data structures and databases)

- – Propositional logic has **very limited expressive power**

For example:

It cannot express *“for all rooms in the Wumpus world, it is true that pits cause breezes in adjacent rooms“* in a general way, in one sentence. Instead, this fact must be stated for each room separately:

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$$
$$B_{2,1} \Leftrightarrow (P_{1,1} \vee P_{2,2} \vee P_{3,1})$$
$$\ldots$$

From Propositional to First-order Logic

Propositional logic:

- can represent only **atomic facts (propositions)** about the world (and logical connections between these)
- cannot name things, refer to them by name, make statements about things and whole sets of things

First-order logic:

can represent facts involving

- **objects** (named by constants)
- **sets of objects** (via variables and quantifiers)
- **properties** of objects (via unary predicates)
- **relations** between objects (via n-ary predicates)
- **things that are determined by other objects** (via functions)

Syntax of First-order Logic (1)

Basic constituents of first-order logic sentences (NOTE: Lower-case initials indicate *variables*):

Constants (names for specific things): *BarackObama, 2, U2, Pi, …*

Variables (arbitrary things): $x, y, a, b, \ldots$

Functions (things that are determined by other things): *Sqrt*(*2*), *FatherOf*(x), …

Predicates (statements about things): *Democrat(Obama)*, *Brother*(x,y), *Greater-than*(x,y), …

Special Predicate: Equality: $x = y$ (shorthand for a predicate *Equal*(x,y))

Logical Connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Quantifiers: $\forall, \exists$

Syntax of First-order Logic (2)

1. Atomic sentences:

Atomic sentence = $predicate(term_1, \ldots, term_n)$
or $term_1 = term_2$

Term = $constant$
or $variable$
or $function(term_1, \ldots, term_n)$

note recursive definition
➔ functions can be nested
(e.g., $Father(Mother(x))$)

Examples:

$Blue(Ball1)$, $Older(Peter, Mary)$, $Older(\underbrace{FatherOf(x)}, x)$,
$3=4$, $Age(Peter)=Age(Paul)$, …

Note: A **function** applied to **arguments** simply stands for a specific **object** (one that can be derived from, or is determined by, its arguments, but for which we do not care to define a name/constant).
A function in logic does not *compute* anything, it *stands for* something ...

Syntax of First-order Logic (3)

2. Complex Sentences:

Sentence =		*atomicSentence*
	or	$\neg$ *sentence*
	or	(*sentence connective sentence*)
	or	*quantifier variable, ... sentence*
Connective =		$\wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$
Quantifier =		$\forall \mid \exists$

Examples:

$$Blue(Ball1)$$

$$Hungry(Wumpus) \wedge \neg Friendly(Wumpus)$$

$$Correct(Theory) \vee Wrong(Theory)$$

$$\forall x\ (Human(x) \Rightarrow Mortal(x))$$

$$\forall x, y\ (Grandfather(x, y) \Rightarrow \exists z\ (Father(x, z) \wedge Parent(z, y)))$$

Semantics of First-order Logic

Complex issue – will not be treated here.

Essentially:

- a **model** contains ≥ 1 objects (domain elements, “the things in the world”)
- an **interpretation** specifies which syntactic elements (constants, predicates, function symbols) correspond to which domain elements in the model

➔ An atomic sentence $predicate(term_1, ..., term_n)$ is **true** iff the objects referred to by $term_1, ..., term_n$ are in the relation referred to by $predicate$ …

Semantics of Quantifiers

Universal Quantification **($\forall$, *"for all x, …"*):**

$\forall x\ S$ is true in a model m iff S is true for all possible values x could take (i.e., with x being **any possible object (constant)** in the given world)

Existential Quantification **($\exists$, *"there exists some x such that …"*):**

$\exists x\ S$ is true in a model m iff S is true with x being **at least one object** in the model
(i.e., there is at least one constant C such that, if we replace x by C, the sentence S is true)

Connections between Quantifiers

1. $\forall x (\forall y\ S)$ is the same as $\forall y (\forall x\ S)$ ➔ can write $\forall x, y\ S$

2. $\exists x (\exists y\ S)$ is the same as $\exists y (\exists x\ S)$ ➔ can write $\exists x, y\ S$

3. $\exists x \forall y\ S$ is **NOT** the same as $\forall y \exists x\ S$!

   $\exists x \forall y\ \ Loves(x, y)$ … "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"

4. **Duality between quantifiers:**
   each can be expressed using the other (➔ strictly, we need only one)

$$\forall x\ Likes(x, Beer) \equiv \neg\exists x\ \neg Likes(x, Beer)$$

$$\exists x\ Likes(x, IceCream) \equiv \neg\forall x\ \neg Likes(x, IceCream)$$

Defining Simple Concepts and Relations in First-order Logic: Some Examples

Note:
Variables must **always** be quantified (either by a $\forall$ or by a $\exists$)

Brothers are siblings: $\forall x \, \forall y \;\; Brother(x,y) \Rightarrow Sibling(x,y)$

“Sibling” is symmetric: $\forall x \, \forall y \;\; Sibling(x,y) \Leftrightarrow Sibling(y,x)$

One’s mother is one’s female parent:

$$\forall x \, \forall y \;\; Mother(x,y) \Leftrightarrow (Female(x) \wedge Parent(x,y))$$

A cousin is a child of a parent’s sibling:

$$\forall x \, \forall y \;\; Cousin(x,y) \;\Leftrightarrow\; \exists p,q \;\; Parent(p,x) \wedge Sibling(q,p) \wedge Parent(q,y)$$

The importance of equality: e.g., definition of (full) *Sibling* in terms of *Parent*:
(“full siblings” have same mother *and* same father)

$$\forall x \, \forall y \;\; Sibling(x,y) \;\Leftrightarrow\; \neg(x=y) \wedge \exists m,f \;\; (\neg(m=f) \wedge \\ Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y))$$

Diagnostic vs. Causal Rules vs. Definitions: An Example from the Wumpus World (1)

"Squares that are adjacent to a pit are breezy"

Causal rules:

- reflect assumed direction of causality in the world
- predict effect from cause:

$$\forall x, y \ \ Pit(x) \wedge Adjacent(x, y) \Rightarrow \ Breezy(y)$$

Diagnostic rules:

- lead from observed effects to hidden causes:

$$\forall y \ \ Breezy(y) \Rightarrow \exists x \ \ Adjacent(x, y) \wedge Pit(x)$$

Note: Neither of the above is complete!
(e.g., causal rule does not say whether squares far away can be breezy)

➔ **Complete definition** of Breezy predicate requires biconditional ("iff"):

$$\forall y \ \ Breezy(y) \Leftrightarrow \exists x \ \ Adjacent(x, y) \wedge Pit(x)$$

Diagnostic vs. Causal Rules vs. Definitions: An Example from the Wumpus World (2)

To do: define predicate (relation) *Adjacent*

Simplest possible solution: enumerate all pairs of adjacent squares

$$Adjacent(Square_{1,1}, Square_{1,2})$$
$$Adjacent(Square_{1,2}, Square_{1,3})$$
$$Adjacent(Square_{1,1}, Square_{2,1})$$
......

… and don't forget to specify that adjacency is symmetric:

$$\forall x, y \quad Adjacent(x, y) \Leftrightarrow Adjacent(y, x)$$

Exercise:
Formalise all the rules of the Wumpus World in first-order logic

Reasoning in First-order Logic

Overview of the following slides:

- First look at a specific first-order inference rule: *modus ponens*
- Formalising modus ponens (and other inference rules) in first-order logic: when are two sentences “the same” / “compatible”?
  ➔ Substitutions and Unification
- A general (and sound and “refutation-complete”) inference procedure: *Resolution* in first-order logic (only briefly)
- Efficient inference in restricted logics
  - first-order Horn clauses
  - forward chaining
  - backward chaining

First-order Horn clause logic + Backward Chaining as a general programming language: PROLOG
(only some simple examples, for time reasons …)

Reasoning in First-order Logic

Goal:

Want to develop specific logical inference patterns like ***modus ponens*** or general inference rules like ***resolution*** for first-order logic (as we did for propositional logic)

Example 1*: Modus Ponens: want to be able to make the following inference:

$$\frac{At(x, JKU) \wedge Human(x) \Rightarrow Smart(x); \quad At(ProfA, JKU),\ Human(ProfA)}{Smart(ProfA)}$$

Problem:

The conditions in the left-hand side of the implication and the two known facts are not identical

➔ we need some way of determining if, e.g., $At(x,JKU)$ and $At(ProfA,JKU)$ are ‘compatible’

➔ and we need to define what exactly we should be permitted to infer

* **From now on:** assume all variables to be universally quantified, if nothing else is indicated; do not write universal quantification explicitly

Reasoning in First-order Logic

Example 2*:

Inference rule must work with any (possibly complex) terms that can be constructed (via functions) in a given model:

$$\begin{array}{l} StudiesAt(x,y) \wedge ProfessorAt(Father(x),y) \Rightarrow Privileged(x) \\ StudiesAt(Son(ProfA),JKU) \\ ProfessorAt(Father(Son(ProfA)),JKU) \\ \hline Privileged(Son(ProfA)) \end{array}$$

Question: How can this kind of inference be formalised in a general way?
➔ need a way to match the arguments of literals to each other
➔ concept of **Unification**

* **From now on:** assume all variables to be universally quantified, if nothing else is indicated; do not write universal quantification explicitly

Reasoning in First-order Logic: Basic Concepts

Definition:

A **substitution** ("binding list") σ is a set of pairs of the form *variable*/*term*, such that each distinct variable is associated with no more than 1 term.

Example: $\sigma = \{x / JKU, y / ProfA\}$

Note: A variable cannot be bound to different terms at the same time in a given sentence (i.e., any variable can appear at most once in a substitution)!

Definition:

The result of **applying a substitution** σ to a first-order sentence *S* is a new sentence *S*σ where all variables in *S* are replaced by the appropriate terms as specified in σ.

Example:

$$S = (At(x, y) \wedge ProfessorAt(Father(x), y) \Rightarrow Smart(x))$$
$$\sigma = \{x / JKU, y / ProfA\}$$
$$S\sigma = (At(JKU, ProfA) \wedge ProfessorAt(Father(JKU), ProfA) \Rightarrow Smart(JKU))$$

Reasoning in First-order Logic: Basic Concepts

Definition:

Unification is the process of **finding a substitution** θ that, when applied to two atomic sentences α and β, makes them **identical**:

$$\text{UNIFY}(\alpha, \beta) = \theta \ \text{ such that } \ \alpha\theta = \beta\theta$$

Such a substitution θ is called a **Unifier**.

Examples:

$$\text{UNIFY}(\,Knows(\,John, x),\; Knows(\,John, Jane)) = \{x/Jane\}$$

$$\text{UNIFY}(\,Knows(\,John, x),\; Knows(\,y, Bill)) = \{x/Bill,\, y/John\}$$

$$\text{UNIFY}(\,Knows(\,John, x),\; Knows(\,y, Mother(y))) = \{y/John,\, x/Mother(John)\}$$

Unification

Some more examples:

UNIFY($Knows(John, x)$, $Likes(John, Jane)$) = **FAIL! (predicate mismatch)**

UNIFY($Knows(John, Mother(Jane))$, $Knows(y, Mother(y))$) = **FAIL!**

– a variable cannot be bound to two different terms within the same sentence!

UNIFY($Knows(John, x)$, $Knows(x, Bill)$) = ***FAIL??***

But: $Knows(John,x)$ means “John knows everybody”; $Knows(x,Bill)$ means “everybody knows Bill”; these two statements ***should*** be compatible.

Solution: The x are really two ***different variables*** in two ***independent sentences*** (each with a universal quantifier): $\forall x\ Knows(John, x)$; $\forall x\ Knows(x, Bill)$

➔ Rename the variables in the two sentences to avoid name clashes *(“standardising apart”).*

➔ UNIFY($Knows(John, x)$, $Knows(z, Bill)$) = $\{x/Bill, z/John\}$

Unifiers are not Unique

Example:

$$\text{UNIFY}(\underbrace{Knows(John, x)}_{\alpha}, \underbrace{Knows(y, z)}_{\beta}) = \{y/John, x/z\} = \theta_1$$

$$\alpha\,\theta_1 = \beta\,\theta_1 = Knows(John, z) =: S_1$$

But: other solutions are also possible!

$$\text{UNIFY}(Knows(John, x), Knows(y, z)) = \{y/John, x/John, z/John\} = \theta_2$$

$$\alpha\,\theta_2 = \beta\,\theta_2 = Knows(John, John) =: S_2$$

Note: S_2 could be obtained from S_1 by an additional substitution $\theta' = \{z/John\}$:

$$S_2 = Knows(John, John) = Knows(John, z)\{z/John\} = S_1\{z/John\}$$

$$\Rightarrow \beta\,\theta_2 = \alpha\,\theta_1\,\theta'$$

➔ We say that S_1 (resp. θ_1) is **more general than** S_2 (resp. θ_2)
(because it places fewer restrictions on the values of the variables)

The Most General Unifier (MGU)

Definition:

A unifier θ_1 is said to be **more general than** a unifier θ_2 iff there is a (non-trivial) unifier θ' such that

$$\beta\,\theta_2 = \alpha\,\theta_1\,\theta'$$

Definition:

The **Most General Unifier (MGU)** of two atomic sentences α and β is a unifier θ such that there is no unifier θ' that is more general than θ

Theorem:

The MGU of two unifiable sentences α and β is **unique** (except for a possible renaming of the variables).
In other words: for each pair of sentences α and β, there is **at most one** MGU.

Exercise: Write an algorithm for computing the MGU of two first-order sentences ...

➔ Modus Ponens in First-order Logic

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

if $(p_1 \wedge p_2 \wedge \ldots \wedge p_n)$ **and** $(p_1' \wedge p_2' \wedge \ldots \wedge p_n')$ **are unifiable, i.e., if there is some substitution** θ **such that** $(\bigwedge_i p_i)\theta = (\bigwedge_i p_i')\theta$

we will usually choose $\theta = MGU(\bigwedge_i p_i, \bigwedge_i p_i')$, to derive the most general conclusion possible

Example:

$$\frac{At(x, JKU) \wedge Human(x) \Rightarrow Smart(x);\quad At(ProfA, JKU),\ Human(ProfA)}{}$$

Let $\theta = \{x/ProfA\}$ $(= \mathrm{MGU}(At(x, JKU) \wedge Human(x),$
$At(ProfA, JKU) \wedge Human(ProfA))$

➔ can conclude $q\theta = Smart(x)\theta = \underline{Smart(ProfA)}$

➔ Modus Ponens in First-order Logic

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

if there is some substitution θ such that $(\bigwedge_i p_i)\theta = (\bigwedge_i p_i')\theta$

Example:

$$\frac{\begin{array}{l} At(x,y) \wedge ProfessorAt(Father(x),y) \Rightarrow Privileged(x); \\ At(Son(ProfA),JKU),\ ProfessorAt(Father(Son(ProfA)),JKU) \end{array}}{}$$

Let $\theta = \{x/Son(ProfA), y/JKU\}$ (exercise: check that this is the MGU here!)

➔ can conclude $q\theta = Privileged(x)\theta = \underline{Privileged(Son(ProfA))}$

A Single Sound and Complete Inference Rule for First-order Logic: Resolution

In principle:

Analogous to resolution in propositional logic:
to prove that $KB \models \alpha$, show that $(KB \wedge \neg\alpha)$ is unsatisfiable
(via conversion to CNF, derivation of empty clause from $(KB \wedge \neg\alpha)$
by applying the resolution rule ***with MGU unification***)

Note:

Conversion to CNF is slightly more complex, due to quantifiers
- need to perform *skolemisation* (to deal with existential quantifiers) etc.

➔ Will not treat general first-order resolution here
(but perhaps in the exam … ☺)

Decidability of First-order Logic: Main Message

- There is a sound and complete inference procedure for first-order logic (resolution)
- Computational complexity of inference in first-order logic is prohibitive
- Every sentence that follows logically from a first-order knowledge base can be proved
- **But:** the fact that a sentence is ***not*** entailed by a KB may be ***unprovable*** in principle (may lead to an infinite proof that never stops)

➔ First-order logic is **SEMI-DECIDABLE** (A. Turing, A. Church, 1936/1937)

Efficient Reasoning with First-order Definite Clauses

Definition:

A **First-order Horn Clause** is a ***universally quantified*** disjunction of literals of which **at most one** is positive (i.e., without negation)

Example:

$(\neg Female(x) \vee \neg Parent(x,y) \vee Mother(x,y))$ and

$(\neg True(x) \vee \neg False(x))$ are Horn clauses

Definition *(relevant for Prolog – see later)* **:**

A **Definite Clause** is a disjunction of literals of which **exactly one** is positive

Example:

$(\neg Female(x) \vee \neg Parent(x,y) \vee Mother(x,y))$ and

$Female(Mary)$ are definite clauses

Efficient Reasoning in a Restricted Logic

Relevance of definite clauses:

1. Every *definite* clause can be written as an implication with non-negated conditions to the left and a single positive literal to the right:

$$(\neg Female(x) \vee \neg Parent(x,y) \vee Mother(x,y))$$
$$\equiv \forall x,y\ (Female(x) \wedge Parent(x,y)\ \Rightarrow Mother(x,y))$$

   or as a fact (atomic sentence):

$$Female(Mary)$$

2. Inference with definite clauses can be done via **Forward Chaining** and **Backward Chaining** (very natural and intuitive form of inference), in an efficient manner

BUT: **Definite and Horn clauses cannot represent everything that full logic can represent !** (simple example: $P \vee Q$)
(➔ Note: Definite Clauses cannot even express simple negation ($\neg P$)!)

Forward Chaining

Efficient method for proving that a literal Q follows from a KB consisting of definite clauses

Basic idea (as in propositional logic):

- Start with atomic sentences ('facts') in KB
- Apply Modus Ponens in forward direction, adding new atomic sentences
- until goal proved or no more inferences possible

In first-order logic:

- Must combine modus ponens with ***unification***
  ➔ A rule (implication) becomes applicable when there exists a unifier (MGU) that makes all its preconditions matchable to corresponding facts in the knowledge base
- Answer is not just "true" or "false", but a specific binding of variables that makes the query provable **("Answer Substitution")**

Forward Chaining

```
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
            $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART($r$)
            for each $\theta$ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
                            for some $p'_1, \ldots, p'_n$ in $KB$
                $q' \leftarrow$ SUBST($\theta, q$)
                if $q'$ is not a renaming of a sentence already in $KB$ or $new$ then do
                    add $q'$ to $new$
                    $\phi \leftarrow$ UNIFY($q', \alpha$)
                    if $\phi$ is not $fail$ then return $\phi$
        add $new$ to $KB$
    return $false$
```

Rename all variables such that no two sentences share any vars

If all conditions satisfied in KB …

… derive conclusion and add to KB

check whether newly added sentence matches the query (literal to be proved)

A Simple Example Problem

“The U.S. law says that it is a crime for Americans 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.”

(Russell & Norvig, p. 280)

The Simple Example Problem in Definite Horn Clauses

"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:"

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

"All of its missiles were sold to it by Col. West:"

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

"Missiles are weapons:"

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

"The country Nono, an enemy of America …"

$$Enemy(Nono, America)$$

"Col. West, who is American …"

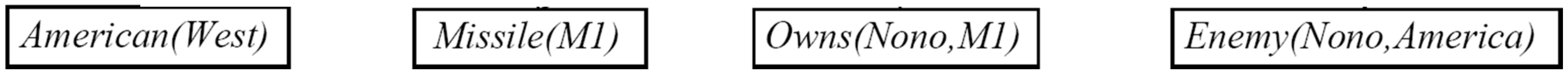
$$American(West)$$

"An enemy of America counts as a hostile country:"

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

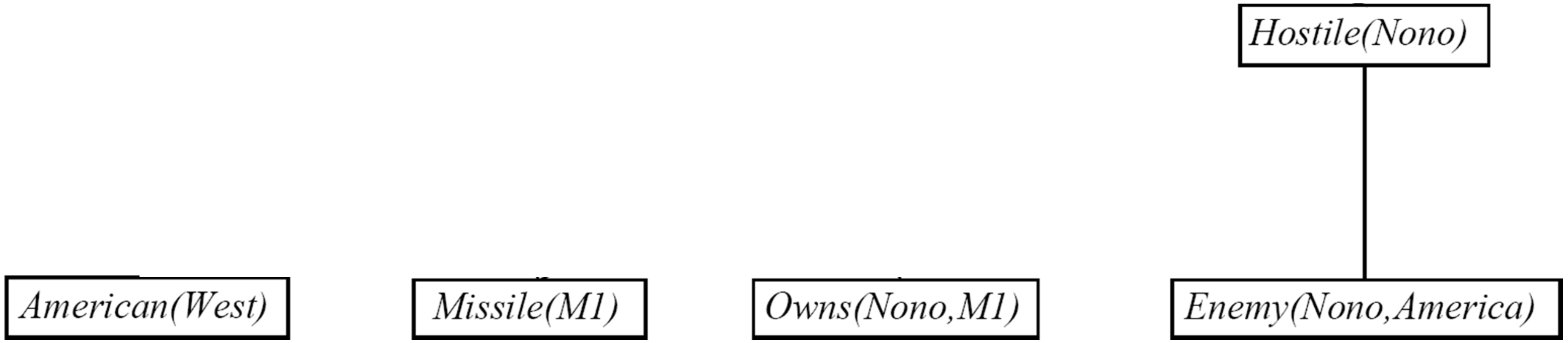
Forward Chaining Proof

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



Forward Chaining Proof

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$
$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$
$Missile(x) \Rightarrow Weapon(x)$
$Enemy(x, America) \Rightarrow Hostile(x)$ $\{x_1/Nono\}$
$Owns(Nono, M_1)$
$Missile(M_1)$
$Enemy(Nono, America)$
$American(West)$



Forward Chaining Proof

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

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

$Missile(x) \Rightarrow Weapon(x)$ $\{x_2/M1\}$

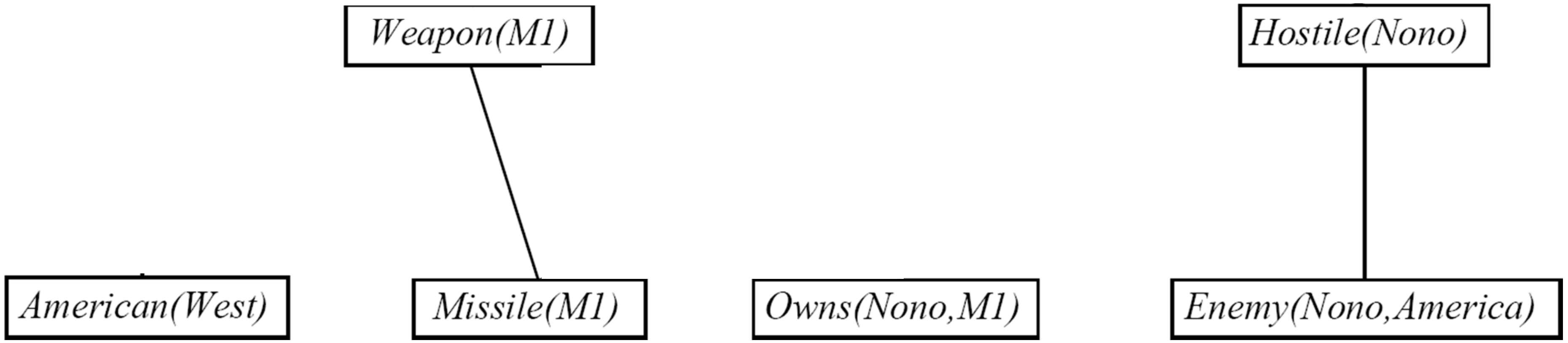
$Enemy(x, America) \Rightarrow Hostile(x)$ $\{x_1/Nono\}$

$Owns(Nono, M_1)$

$Missile(M_1)$

$Enemy(Nono, America)$

$American(West)$



Forward Chaining Proof

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

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$ $\{x_3/M1\}$

$Missile(x) \Rightarrow Weapon(x)$ $\{x_2/M1\}$

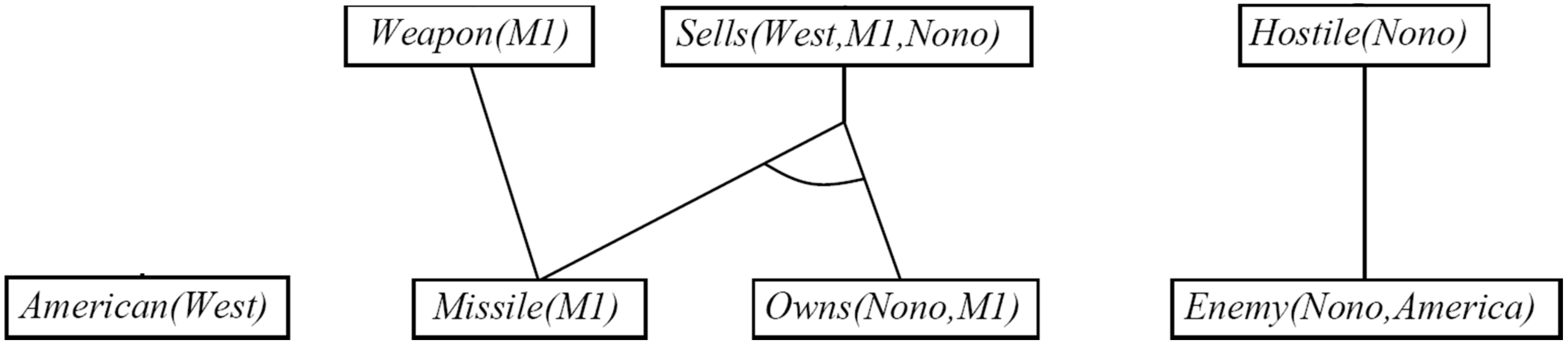
$Enemy(x, America) \Rightarrow Hostile(x)$ $\{x_1/Nono\}$

$Owns(Nono, M_1)$

$Missile(M_1)$

$Enemy(Nono, America)$

$American(West)$



Forward Chaining Proof

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$ $\{x_4/West, y/M1, z/Nono\}$

$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$ $\{x_3/M1\}$

$Missile(x) \Rightarrow Weapon(x)$ $\{x_2/M1\}$

$Enemy(x, America) \Rightarrow Hostile(x)$ $\{x_1/Nono\}$

$Owns(Nono, M_1)$

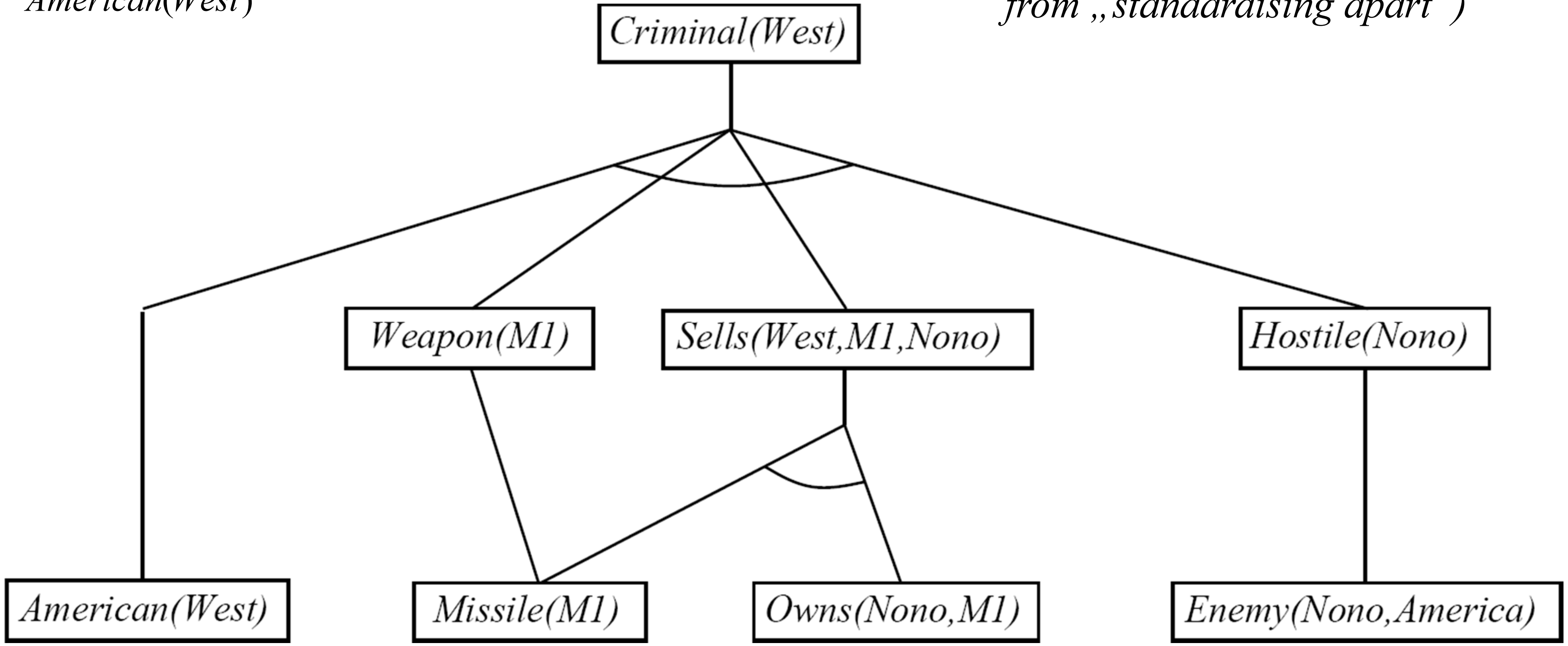
$Missile(M_1)$

$Enemy(Nono, America)$

$American(West)$

Answer Substitution σ

(note different x variables – result from „standardising apart")



Backward Chaining

Efficient method for proving that a literal Q follows from a KB consisting of definite clauses

Basic idea (as in propositional logic):

- Start with query (literal to be proved)
- Apply Modus Ponens in backward direction, adding new subgoals to the "agenda" (= list of goals still to be proven)
- Prove subgoals recursively, one by one
- If proof ***fails*** at any point, ***backtrack:*** undo last step(s) (including any substitutions) and check for alternative (there might be several rules that can conclude a given query literal)
- until all subgoals are proved

- Finding an applicable rule = finding a rule whose conclusion (right-hand side) can be **unified (MGU)** with the goal to be proven
- Must keep track of substitutions throughout entire proof

➔ Result of proof (if literal can be proved) is a ***substitution***

Backward Chaining

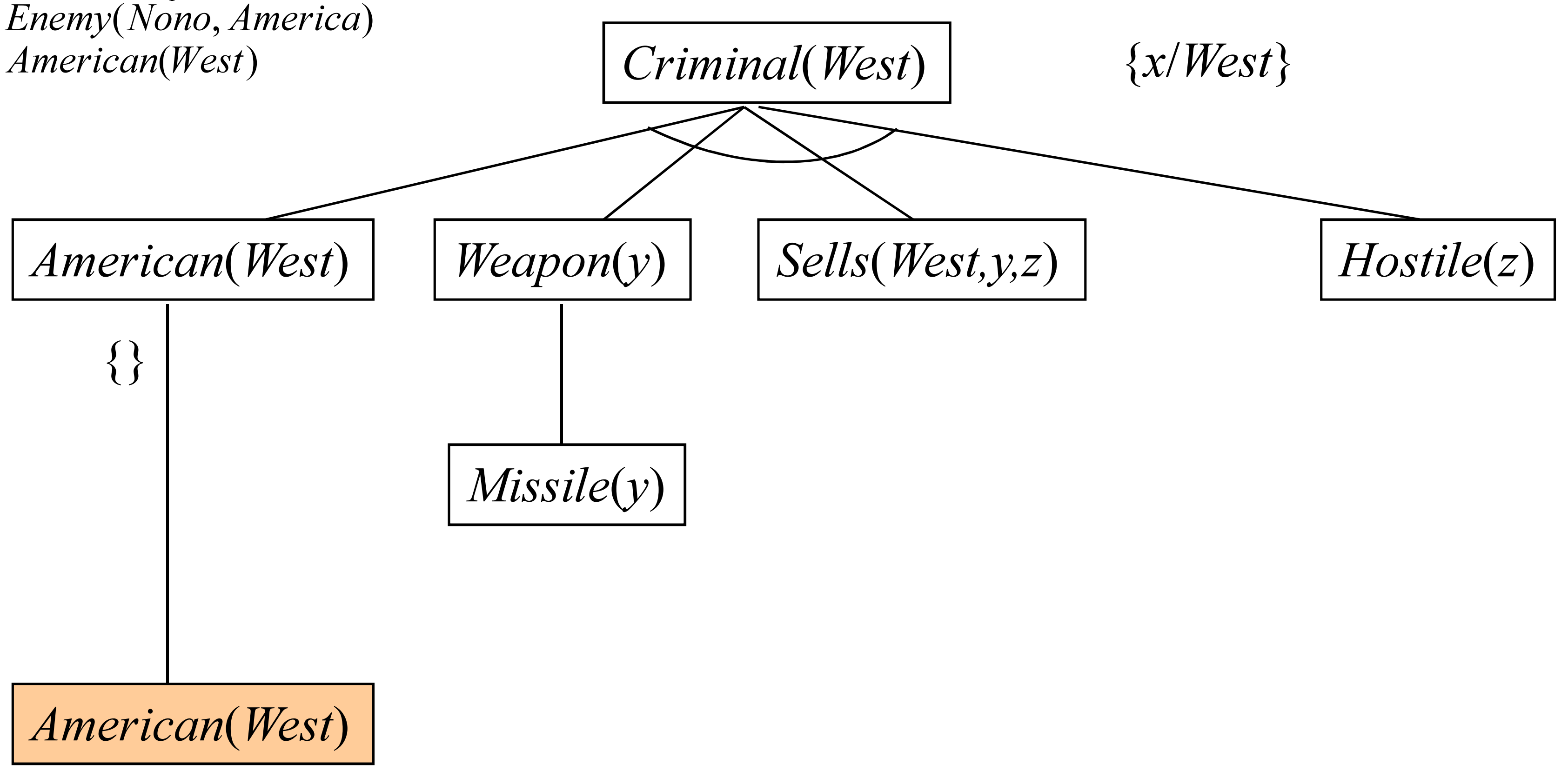
```
function FOL-BC-ASK($KB, goals, \theta$) returns a set of substitutions
    inputs: $KB$, a knowledge base
            $goals$, a list of conjuncts forming a query ($\theta$ already applied)
            $\theta$, the current substitution, initially the empty substitution $\{\ \}$
    local variables: $answers$, a set of substitutions, initially empty

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

left-to-right, depth-first search

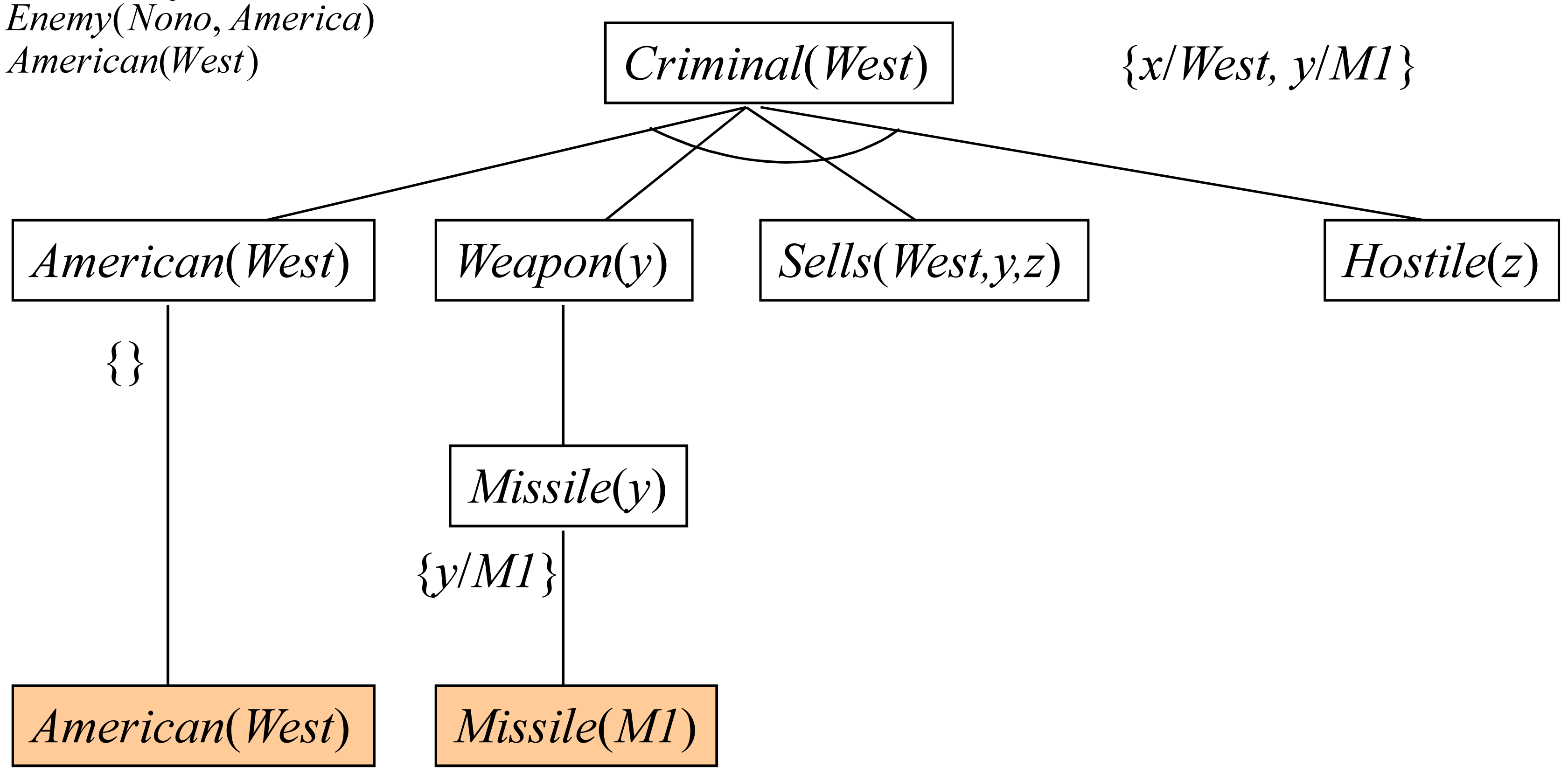
Backward Chaining Proof

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



Backward Chaining Proof

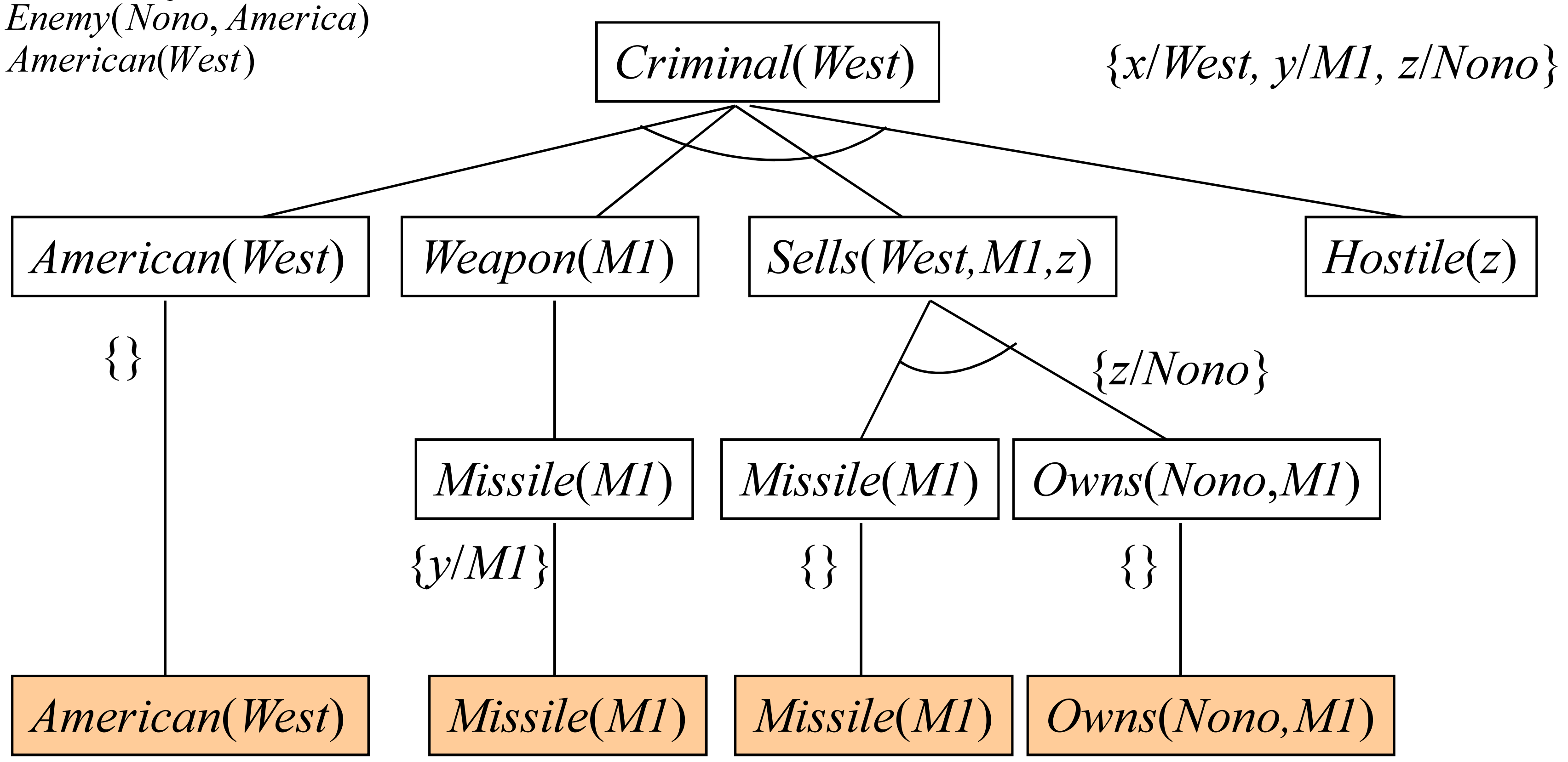
$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x)$
$Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
$Missile(x) \Rightarrow Weapon(x)$
$Enemy(x, America) \Rightarrow Hostile(x)$
$Owns(Nono, M_1)$
$Missile(M_1)$
$Enemy(Nono, America)$
$American(West)$



… what if there were another fact $Missile(M2)$ in the KB …?

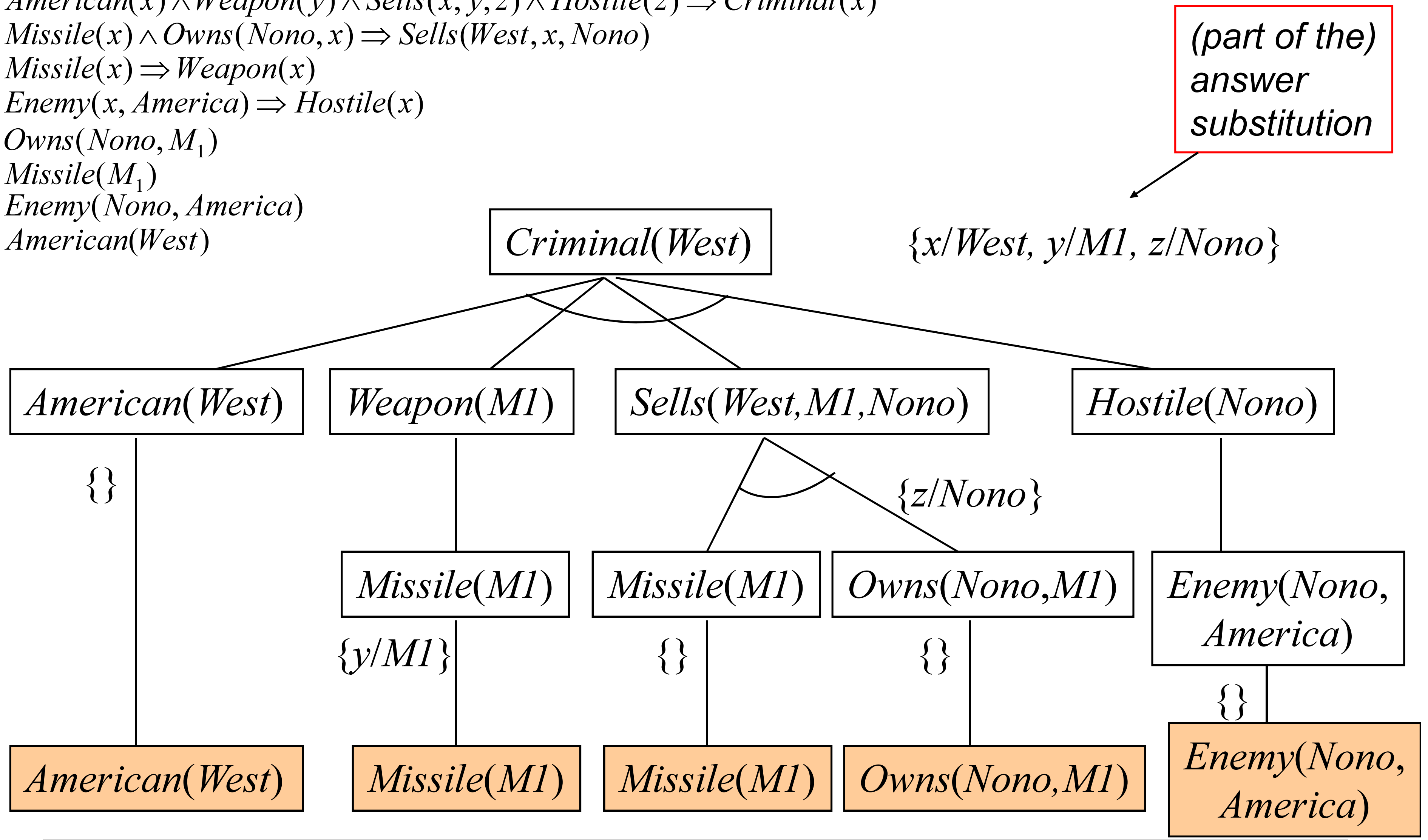
Backward Chaining Proof

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



Backward Chaining Proof

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



Logic as a Universal Programming Language

Prolog = “Programming in Logic” =

First-order Horn clause logic (*Definite Clauses* only!)
+ backward chaining proof algorithm (depth-first search + backtracking)
+ unification (MGU)
+ a few extra-logical control features (“cut”, assert/retract)
+ very efficient clause indexing and compilation

Prolog = a full universal programming language for practical use

Invented in the 1970’s (Alain Colmerauer (France); Robert Kowalski (UK))

Fundamental difference to other programming languages:

- totally declarative specification of a solution (ideally):
  rather than specifying an algorithm for finding a solution,
  simply define problem and solution conditions in first-order logic
- Prolog’s built-in inference process (proof procedure) constructs solution

The Nono World in Prolog

Prolog Program = Knowledge Base:

Notation: variables: upper case; predicates, functions, constants: lower case

```
% "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." (Russell & Norvig, p.280).

criminal( X) :-
    american( X), weapon( Y),          % It is a crime for an American to sell
    sells( X, Y, Z), hostile( Z).      % weapons to hostile nations
```

Note: implications are written in reverse order in Prolog: "conclusion :- (IF) conditions"

```
sells( west, X, nono) :-
    missile( X),                       % All of the missiles owned by Nono
    owns( nono, X).                    % were sold to it by West

weapon( X) :- missile( X).             % Missiles are weapons
% weapon (X) :- spear( X).             % Spears are weapons

hostile( X) :- enemy( X, america).     % ... and enemies of America are hostile nations

owns( nono, m1).                       % Nono owns something called m1
missile( m1).                          % … and this m1 is a missile.
enemy( nono, america).                 % Nono is an enemy of America.
american( west).                       % Colonel West is American.
```

Asking Prolog about the Nono World

Prolog Program Calls = Queries:

```
| ?- criminal( west).          % Query: is Colonel West a criminal?
yes

| ?- criminal(X).              % is there a criminal, and if so, what is his/her name?
X = west

| ?- weapon( X).               % is there a weapon, and if so, what is its name?
X = m1

| ?- enemy(X,Y).               % who is the enemy of whom?
X = nono
Y = america

| ?- enemy(X,X).               % is there anything that is its own enemy?
no
```

answer substitution

Prolog's Control Strategy: Prove Query via Backward Chaining + Unification (left-to-right, depth-first)

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

sells( west, X, nono) :-
    missile( X),
    owns( nono, X).

weapon( X) :- missile( X).

hostile( X) :- enemy( X, america).

owns( nono, m1).
missile( m1).
enemy( nono, america).
american( west).
```

```
| ?- trace.
% The debugger will first creep -- showing everything (trace)
yes
| ?- criminal(X).
   1    1 Call: criminal(X) ?
   2      2 Call: american(X) ?
   2      2 Exit: american(west) ?                { X/west}
   3      2 Call: weapon(_1026) ?
   4        3 Call: missile(_1026) ?
   4        3 Exit: missile(m1) ?                 { _1026/m1}
   3      2 Exit: weapon(m1) ?
   5      2 Call: sells(west,m1,_1021) ?          { _1021/nono}
   6        3 Call: missile(m1) ?
   6        3 Exit: missile(m1) ?
   7        3 Call: owns(nono,m1) ?
   7        3 Exit: owns(nono,m1) ?
   5      2 Exit: sells(west,m1,nono) ?
   8      2 Call: hostile(nono) ?
   9        3 Call: enemy(nono,america) ?
   9        3 Exit: enemy(nono,america) ?
   8      2 Exit: hostile(nono) ?
   1    1 Exit: criminal(west) ?
 X = west
 yes
```

Summary

- **First-order logic** talks about objects and relations
- **Syntax:** constants, functions, predicates, equality, quantifiers + variables
- **Increased expressive power:** sufficient to define wumpus world in a compact way (not shown here)
- Reasoning in first-order logic: **substitutions, unification, inference**
- **Resolution:** sound and complete inference procedure
- First-order Horn / definite clauses: efficient inference possible via **Forward** and **Backward Chaining**
- Backward Chaining on Horn clauses: basis for **logic programming (Prolog)**