AI

Artificial Intelligence

First Order Logic (Ch. 8)

Outline

- Why FOL?
- Syntax and semantics of FOL
- Using FOL
- Wumpus world in FOL
- Knowledge engineering in FOL

Pros and cons of propositional logic

- ☺ Propositional logic is declarative
- ☺ Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)
- ☺ Propositional logic is compositional:
  - meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$
- ☺ Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)
- ☹ Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares“
    - except by writing one sentence for each square

First-order logic

- Whereas propositional logic assumes the world contains facts,
- first-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

Logics in general

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief
Fuzzy logic	facts + degree of truth	known interval value

Syntax of FOL: Basic elements

- Constants KingJohn, 2, CU,...
- Predicates Brother, >,...
- Functions Sqrt, LeftLegOf,...
- Variables x, y, a, b,...
- Connectives $\neg, \Rightarrow, \wedge, \vee, \Leftrightarrow$
- Equality $=$
- Quantifiers $\forall, \exists$

Atomic sentences

Atomic sentence = *predicate* $(term_1,...,term_n)$
or $term_1 = term_2$

Term = *function* $(term_1,...,term_n)$
or *constant* or *variable*

- E.g., *Brother(KingJohn,RichardTheLionheart) > (Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))*

Complex sentences

- Complex sentences are made from atomic sentences using connectives

$$\neg S,\ S_1 \wedge S_2,\ S_1 \vee S_2,\ S_1 \Rightarrow S_2,\ S_1 \Leftrightarrow S_2,$$

E.g. $Sibling(KingJohn,Richard) \Rightarrow Sibling(Richard,KingJohn)$

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

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

Models in FOL

- Constants
  - exact mapping between constants and objects
  - $m(\text{Mary}) = x_1$
- Predicates
  - $m(\text{CapitalOf}) = ((x_1,y_1), (x_2,y_2), \ldots)$

FOL vs Propositional Logic

- FOL is just a fancier way to write propositional logic statements (propositionalization)

Knowledge base in first-order logic

$$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$$
$$\forall x\, \text{Student}(x) \rightarrow \text{Person}(x)$$
$$\exists x\, \text{Student}(x) \wedge \text{Creative}(x)$$

Knowledge base in propositional logic

$$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$$
$$(\text{Student}(\text{alice}) \rightarrow \text{Person}(\text{alice})) \wedge (\text{Student}(\text{bob}) \rightarrow \text{Person}(\text{bob}))$$
$$(\text{Student}(\text{alice}) \wedge \text{Creative}(\text{alice})) \vee (\text{Student}(\text{bob}) \wedge \text{Creative}(\text{bob}))$$

[Example from Percy Liang]

Truth in first-order logic

- Sentences are true with respect to a model and an interpretation
- Model contains objects (domain elements) and relations among them
- Interpretation specifies referents for
  constant symbols → objects
  predicate symbols → relations
  function symbols → functional relations
- An atomic sentence *predicate*$(term_1,\ldots,term_n)$ is true
  iff the objects referred to by $term_1,\ldots,term_n$
  are in the relation referred to by *predicate*

Models for FOL: Example

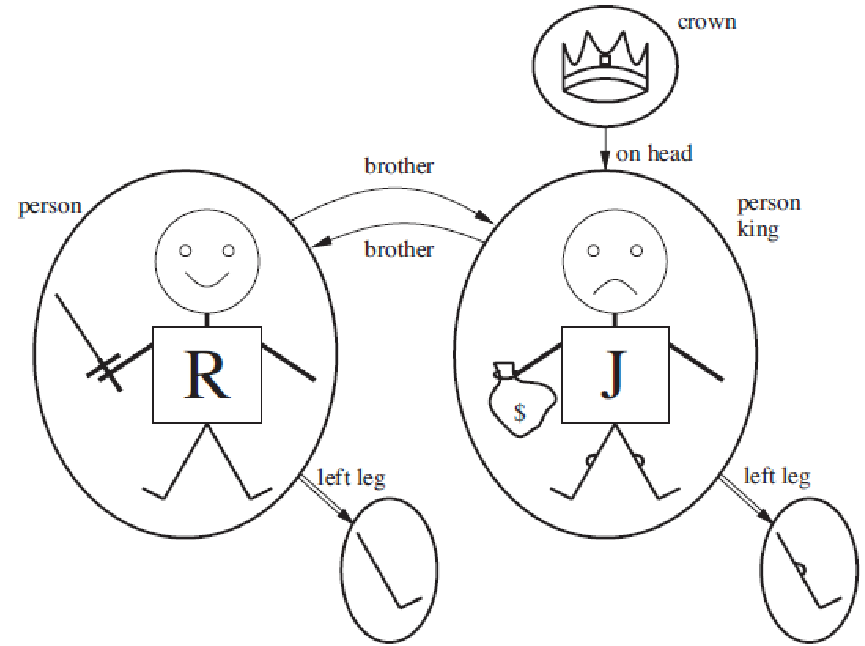


Figure 8.2 A model containing five objects, two binary relations, three unary relations (indicated by labels on the objects), and one unary function, left-leg.

Truth example

Consider the interpretation in which
$Richard \rightarrow$ Richard the Lionheart
$John \rightarrow$ the evil King John
$Brother \rightarrow$ the brotherhood relation

Under this interpretation, $Brother(Richard, John)$ is true just in case Richard the Lionheart and the evil King John are in the brotherhood relation in the model

Universal quantification

- $\forall$*<variables>* *<sentence>*

Everyone at CU is smart:

$\forall x\ At(x,CU) \Rightarrow Smart(x)$

- $\forall x$ *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model

- Roughly speaking, equivalent to the conjunction of instantiations of *P*

$$\begin{aligned}&At(KingJohn,CU) \Rightarrow Smart(KingJohn)\\ \wedge\ &At(Richard,CU) \Rightarrow Smart(Richard)\\ \wedge\ &At(CU,CU) \Rightarrow Smart(CU)\\ \wedge\ &\ldots\end{aligned}$$

A common mistake to avoid

- Typically, $\Rightarrow$ is the main connective with $\forall$
- Common mistake: using $\wedge$ as the main connective with $\forall$:

  $\forall x\ At(x,CU) \wedge Smart(x)$

  means "Everyone is at CU and everyone is smart"

Existential quantification

- $\exists$<*variables*> <*sentence*>

- Someone at CU is smart:
- $\exists x$ At(x,CU) $\wedge$ Smart(x)$

- $\exists x$ *P* is true in a model *m* iff *P* is true with *x* being some possible object in the model

- Roughly speaking, equivalent to the disjunction of instantiations of *P*

  At(KingJohn,CU) $\wedge$ Smart(KingJohn)

  $\vee$ At(Richard,CU) $\wedge$ Smart(Richard)

  $\vee$ At(CU,CU) $\wedge$ Smart(CU)

  $\vee$ ...

Another common mistake to avoid

- Typically, $\land$ is the main connective with $\exists$
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \ At(x,CU) \Rightarrow Smart(x)$$

is true if there is anyone who is not at CU!

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$

- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
- $\exists x\ \forall y$ 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,IceCream) $\neg\exists x\ \neg$Likes(x,IceCream)
- $\exists x$ Likes(x,Broccoli) $\neg\forall x\ \neg$Likes(x,Broccoli)

Fun with sentences

Brothers are siblings

$\forall x, y \;\; Brother(x, y) \; \Rightarrow \; Sibling(x, y).$

"Sibling" is symmetric

$\forall x, y \;\; Sibling(x, y) \; \Leftrightarrow \; Sibling(y, x).$

One's mother is one's female parent

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

A first cousin is a child of a parent's sibling

$\forall x, y \;\; FirstCousin(x, y) \; \Leftrightarrow \; \exists p, ps \;\; Parent(p, x) \land Sibling(ps, p) \land Parent(ps, y)$

Equality

$term_1 = term_2$ is true under a given interpretation
if and only if $term_1$ and $term_2$ refer to the same object

E.g., $1 = 2$ and $\forall\, x \;\; \times(Sqrt(x), Sqrt(x)) = x$ are satisfiable
$2 = 2$ is valid

E.g., definition of (full) $Sibling$ in terms of $Parent$:

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

Using FOL

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Leftrightarrow Sibling(x,y)$

- One's mother is one's female parent

  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- "Sibling" is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

1. The only sets are the empty set and those made by adjoining something to a set:

$$\forall s \;\; Set(s) \;\Leftrightarrow\; (s = \{\,\}) \lor (\exists x, s_2 \;\; Set(s_2) \land s = \{x|s_2\}) \,.$$

2. The empty set has no elements adjoined into it. In other words, there is no way to decompose $\{\,\}$ into a smaller set and an element:

$$\neg\exists x, s \;\; \{x|s\} = \{\,\} \,.$$

3. Adjoining an element already in the set has no effect:

$$\forall x, s \;\; x \in s \;\Leftrightarrow\; s = \{x|s\} \,.$$

4. The only members of a set are the elements that were adjoined into it. We express this recursively, saying that x is a member of s if and only if s is equal to some set s_2 adjoined with some element y, where either y is the same as x or x is a member of s_2:

$$\forall x, s \;\; x \in s \;\Leftrightarrow\; \exists y, s_2 \;\; (s = \{y|s_2\} \land (x = y \lor x \in s_2)) \,.$$

5. A set is a subset of another set if and only if all of the first set's members are members of the second set:

$$\forall s_1, s_2 \;\; s_1 \subseteq s_2 \;\Leftrightarrow\; (\forall x \;\; x \in s_1 \;\Rightarrow\; x \in s_2) \,.$$

6. Two sets are equal if and only if each is a subset of the other:

$$\forall s_1, s_2 \;\; (s_1 = s_2) \;\Leftrightarrow\; (s_1 \subseteq s_2 \land s_2 \subseteq s_1) \,.$$

7. An object is in the intersection of two sets if and only if it is a member of both sets:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cap s_2) \;\Leftrightarrow\; (x \in s_1 \land x \in s_2) \,.$$

8. An object is in the union of two sets if and only if it is a member of either set:

$$\forall x, s_1, s_2 \;\; x \in (s_1 \cup s_2) \;\Leftrightarrow\; (x \in s_1 \lor x \in s_2) \,.$$

Interacting with FOL KBs

Suppose a wumpus-world agent is using an FOL KB
and perceives a smell and a breeze (but no glitter) at $t = 5$:

$Tell(KB, Percept([Smell, Breeze, None], 5))$
$Ask(KB, \exists a \;\; Action(a, 5))$

I.e., does KB entail any particular actions at $t = 5$?

Answer: $Yes,\ \{a/Shoot\}$ $\leftarrow$ substitution (binding list)

Given a sentence S and a substitution σ,
$S\sigma$ denotes the result of plugging σ into S; e.g.,
$S = Smarter(x, y)$
$\sigma = \{x/Hillary, y/Bill\}$
$S\sigma = Smarter(Hillary, Bill)$

$Ask(KB, S)$ returns some/all σ such that $KB \models S\sigma$

Knowledge base for the wumpus world

"Perception"

$\forall\, b, g, t \;\; Percept([Smell, b, g], t) \;\Rightarrow\; Smelt(t)$

$\forall\, s, b, t \;\; Percept([s, b, Glitter], t) \;\Rightarrow\; AtGold(t)$

Reflex: $\forall\, t \;\; AtGold(t) \;\Rightarrow\; Action(Grab, t)$

Reflex with internal state: do we have the gold already?

$\forall\, t \;\; AtGold(t) \wedge \neg Holding(Gold, t) \;\Rightarrow\; Action(Grab, t)$

$Holding(Gold, t)$ cannot be observed

$\Rightarrow$ keeping track of change is essential

Deducing hidden properties

Properties of locations:

$\forall x,t \;\; At(Agent,x,t) \land Smelt(t) \;\Rightarrow\; Smelly(x)$

$\forall x,t \;\; At(Agent,x,t) \land Breeze(t) \;\Rightarrow\; Breezy(x)$

Squares are breezy near a pit:

Diagnostic rule—infer cause from effect

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

Causal rule—infer effect from cause

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

Neither of these is complete—e.g., the causal rule doesn't say whether squares far away from pits can be breezy

Definition for the $Breezy$ predicate:

$$\forall y \;\; Breezy(y) \;\Leftrightarrow\; [\exists x \;\; Pit(x) \land Adjacent(x,y)]$$

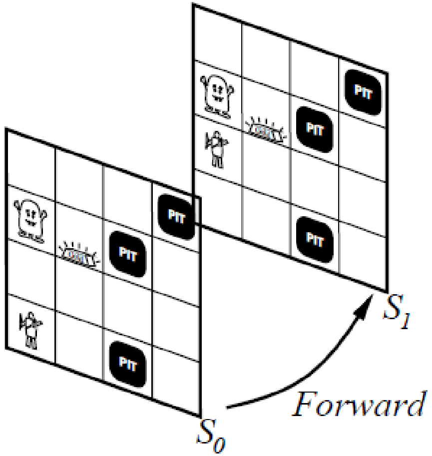
Keeping track of change

Facts hold in situations, rather than eternally
E.g., $Holding(Gold, Now)$ rather than just $Holding(Gold)$

Situation calculus is one way to represent change in FOL:

Adds a situation argument to each non-eternal predicate
E.g., Now in $Holding(Gold, Now)$ denotes a situation

Situations are connected by the $Result$ function
$Result(a, s)$ is the situation that results from doing a in s



Describing actions I

"Effect" axiom—describe changes due to action

$\forall s \;\; AtGold(s) \;\Rightarrow\; Holding(Gold, Result(Grab, s))$

"Frame" axiom—describe **non-changes** due to action

$\forall s \;\; HaveArrow(s) \;\Rightarrow\; HaveArrow(Result(Grab, s))$

Frame problem: find an elegant way to handle non-change

(a) representation—avoid frame axioms

(b) inference—avoid repeated "copy-overs" to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or ...

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, ...

Describing actions II

Successor-state axioms solve the representational frame problem

Each axiom is "about" a **predicate** (not an action per se):

P true afterwards $\Leftrightarrow$ [an action made P true
$\vee$ P true already and no action made P false]

For holding the gold:

$$\forall a, s \;\; Holding(Gold, Result(a, s)) \;\Leftrightarrow$$
$$[(a = Grab \wedge AtGold(s))$$
$$\vee\; (Holding(Gold, s) \wedge a \neq Release)]$$

Making plans

Initial condition in KB:

$$At(Agent, [1,1], S_0)$$
$$At(Gold, [1,2], S_0)$$

Query: $Ask(KB, \exists\, s\ \ Holding(Gold, s))$

i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$

i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $Ask(KB, \exists p \;\; Holding(Gold, PlanResult(p, S_0)))$
has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$$\forall s \;\; PlanResult([\,], s) = s$$
$$\forall a, p, s \;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

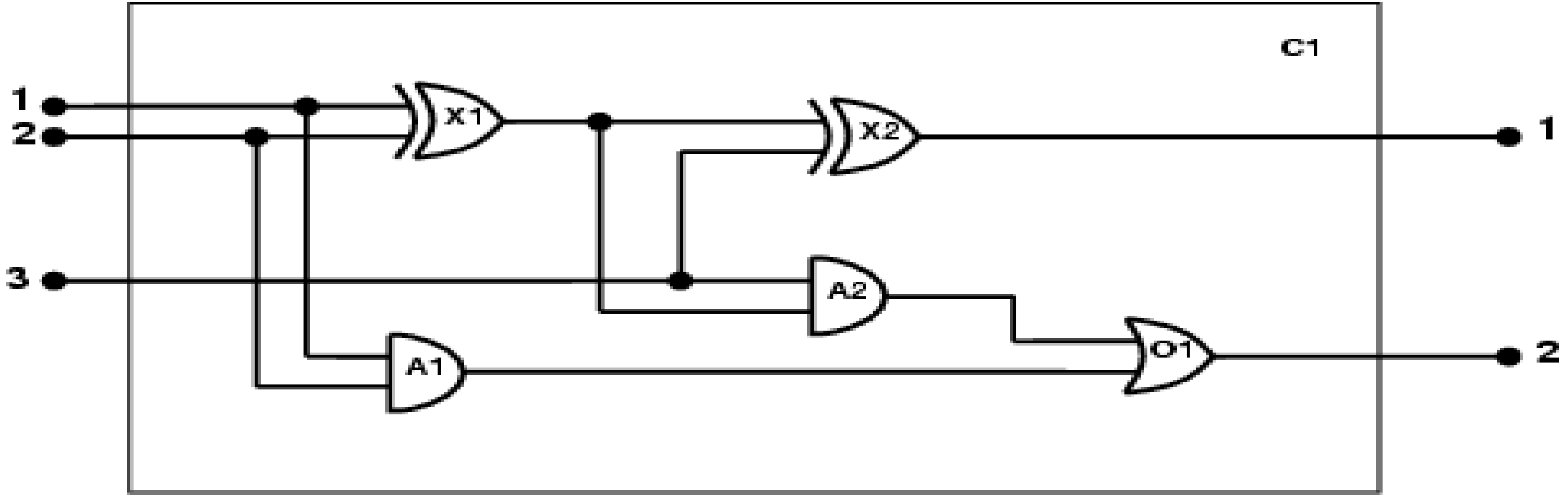
Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

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

The electronic circuits domain

One-bit full adder



1. If two terminals are connected, then they have the same signal:
$$\forall t_1, t_2 \;\; Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \;\Rightarrow\; Signal(t_1) = Signal(t_2)\ .$$
2. The signal at every terminal is either 1 or 0:
$$\forall t \;\; Terminal(t) \;\Rightarrow\; Signal(t) = 1 \vee Signal(t) = 0\ .$$
3. Connected is commutative:
$$\forall t_1, t_2 \;\; Connected(t_1, t_2) \;\Leftrightarrow\; Connected(t_2, t_1)\ .$$
4. There are four types of gates:
$$\forall g \;\; Gate(g) \wedge k = Type(g) \;\Rightarrow\; k = AND \vee k = OR \vee k = XOR \vee k = NOT\ .$$
5. An AND gate's output is 0 if and only if any of its inputs is 0:
$$\forall g \;\; Gate(g) \wedge Type(g) = AND \;\Rightarrow\; Signal(Out(1, g)) = 0 \;\Leftrightarrow\; \exists n \;\; Signal(In(n, g)) = 0\ .$$
6. An OR gate's output is 1 if and only if any of its inputs is 1:
$$\forall g \;\; Gate(g) \wedge Type(g) = OR \;\Rightarrow\; Signal(Out(1, g)) = 1 \;\Leftrightarrow\; \exists n \;\; Signal(In(n, g)) = 1\ .$$
7. An XOR gate's output is 1 if and only if its inputs are different:
$$\forall g \;\; Gate(g) \wedge Type(g) = XOR \;\Rightarrow\; Signal(Out(1, g)) = 1 \;\Leftrightarrow\; Signal(In(1, g)) \neq Signal(In(2, g))\ .$$
8. A NOT gate's output is different from its input:
$$\forall g \;\; Gate(g) \wedge (Type(g) = NOT) \;\Rightarrow\; Signal(Out(1, g)) \neq Signal(In(1, g))\ .$$
9. The gates (except for NOT) have two inputs and one output.
$$\forall g \;\; Gate(g) \wedge Type(g) = NOT \;\Rightarrow\; Arity(g, 1, 1)\ .$$
$$\forall g \;\; Gate(g) \wedge k = Type(g) \wedge (k = AND \vee k = OR \vee k = XOR) \;\Rightarrow\; Arity(g, 2, 1)$$
10. A circuit has terminals, up to its input and output arity, and nothing beyond its arity:
$$\forall c, i, j \;\; Circuit(c) \wedge Arity(c, i, j) \;\Rightarrow$$
$$\forall n \;\; (n \leq i \;\Rightarrow\; Terminal(In(c, n))) \wedge (n > i \;\Rightarrow\; In(c, n) = Nothing) \wedge$$
$$\forall n \;\; (n \leq j \;\Rightarrow\; Terminal(Out(c, n))) \wedge (n > j \;\Rightarrow\; Out(c, n) = Nothing)$$
11. Gates, terminals, signals, gate types, and *Nothing* are all distinct.
$$\forall g, t \;\; Gate(g) \wedge Terminal(t) \;\Rightarrow\; g \neq t \neq 1 \neq 0 \neq OR \neq AND \neq XOR \neq NOT \neq Nothing\ .$$
12. Gates are circuits.
$$\forall g \;\; Gate(g) \;\Rightarrow\; Circuit(g)$$

$$
\begin{aligned}
&\mathit{Circuit}(C_1) \wedge \mathit{Arity}(C_1, 3, 2)\\
&\mathit{Gate}(X_1) \wedge \mathit{Type}(X_1) = \mathit{XOR}\\
&\mathit{Gate}(X_2) \wedge \mathit{Type}(X_2) = \mathit{XOR}\\
&\mathit{Gate}(A_1) \wedge \mathit{Type}(A_1) = \mathit{AND}\\
&\mathit{Gate}(A_2) \wedge \mathit{Type}(A_2) = \mathit{AND}\\
&\mathit{Gate}(O_1) \wedge \mathit{Type}(O_1) = \mathit{OR}\ .
\end{aligned}
$$

$$
\begin{array}{ll}
\mathit{Connected}(\mathit{Out}(1, X_1), \mathit{In}(1, X_2)) & \mathit{Connected}(\mathit{In}(1, C_1), \mathit{In}(1, X_1))\\
\mathit{Connected}(\mathit{Out}(1, X_1), \mathit{In}(2, A_2)) & \mathit{Connected}(\mathit{In}(1, C_1), \mathit{In}(1, A_1))\\
\mathit{Connected}(\mathit{Out}(1, A_2), \mathit{In}(1, O_1)) & \mathit{Connected}(\mathit{In}(2, C_1), \mathit{In}(2, X_1))\\
\mathit{Connected}(\mathit{Out}(1, A_1), \mathit{In}(2, O_1)) & \mathit{Connected}(\mathit{In}(2, C_1), \mathit{In}(2, A_1))\\
\mathit{Connected}(\mathit{Out}(1, X_2), \mathit{Out}(1, C_1)) & \mathit{Connected}(\mathit{In}(3, C_1), \mathit{In}(2, X_2))\\
\mathit{Connected}(\mathit{Out}(1, O_1), \mathit{Out}(2, C_1)) & \mathit{Connected}(\mathit{In}(3, C_1), \mathit{In}(1, A_2))\ .
\end{array}
$$

Pose queries to the inference procedure

What combinations of inputs would cause the first output of C_1 (the sum bit) to be 0 and the second output of C_1 (the carry bit) to be 1?

$$
\begin{aligned}
\exists\, i_1, i_2, i_3\ \ &\mathit{Signal}(\mathit{In}(1, C_1)) = i_1 \wedge \mathit{Signal}(\mathit{In}(2, C_1)) = i_2 \wedge \mathit{Signal}(\mathit{In}(3, C_1)) = i_3\\
&\wedge \mathit{Signal}(\mathit{Out}(1, C_1)) = 0 \wedge \mathit{Signal}(\mathit{Out}(2, C_1)) = 1\ .
\end{aligned}
$$

The answers are substitutions for the variables i_1, i_2, and i_3 such that the resulting sentence is entailed by the knowledge base. ASKVARS will give us three such substitutions:

$$
\{i_1/1,\ i_2/1,\ i_3/0\} \quad \{i_1/1,\ i_2/0,\ i_3/1\} \quad \{i_1/0,\ i_2/1,\ i_3/1\}\ .
$$

What are the possible sets of values of all the terminals for the adder circuit?

$$
\begin{aligned}
\exists\, i_1, i_2, i_3, o_1, o_2\ \ &\mathit{Signal}(\mathit{In}(1, C_1)) = i_1 \wedge \mathit{Signal}(\mathit{In}(2, C_1)) = i_2\\
&\wedge \mathit{Signal}(\mathit{In}(3, C_1)) = i_3 \wedge \mathit{Signal}(\mathit{Out}(1, C_1)) = o_1 \wedge \mathit{Signal}(\mathit{Out}(2, C_1)) = o_2\ .
\end{aligned}
$$

Summary

First-order logic:

– objects and relations are semantic primitives

– syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define wumpus world

Situation calculus:

– conventions for describing actions and change in FOL

– can formulate planning as inference on a situation calculus KB

Artificial Intelligence

First Order Logic Problem (from 3.6.2)

First Order Logic

- NACLO problem from 2014
- Author: Ben King
- http://www.nacloweb.org/resources/problems/2014/N2014-H.pdf
- http://www.nacloweb.org/resources/problems/2014/N2014-HS.pdf

First Order Logic

(H) Bertrand and Russell (1/3) [10 points]

Teachers can be hard to understand sometimes. Case in point, the math teacher, Mr. Whitehead. Just this morning, he told the class, "It's not the case that if at least one student studied for the test, then every student failed the test." What does that even mean?

Well, the two new kids in the class, Bertrand and Russell, have come up with a plan to make sense of Mr. Whitehead's statements. They call it first-order logic (FOL), a way to map these confusing statements into an unambiguous representation. Bertrand says the whole system is built the idea of propositions, a statement that is either true or false. Propositions can be statements about people or things like *studied_for(John, test)* or *is_hard(test)*. Propositions can also be combined to make more complex statements with the following symbols:

Symbol	Example statement	Interpretation	Explanation
¬	¬ *studied_for*(*John*, *test*)	John did <u>not</u> study for the test.	The statement is true if and only if John did not study for the test.
∧	*is_hard*(test) ∧ *is_long*(test)	The test is long <u>and</u> hard.	This statement is true whenever the test is long and the test is hard.
∨	*is_hard*(test) ∨ *is_long*(test)	The test is long <u>or</u> hard.	This statement is true if the test is long, or if the test is hard, or both.
⟹	*studied_for*(*John*, *test*) ⟹ *aced*(*John*, *test*)	<u>If</u> John studied for the test, <u>then</u> he aced it.	This is true if the statement on the right side of the arrow is always true whenever the statement on the left side of the arrow is true. If the statement on the left is false, then the whole statement is true by default (if John didn't study, we don't know how he did on the test).

“But,” says Russell, “the most important part of first-order logic is the quantifiers.” Quantifiers allow you to make general statements like Mr. Whitehead loves to do.

Symbol	Example statement	Interpretation	Explanation
$\forall$	$[\forall_x : \mathit{student}(x) \Longrightarrow \mathit{studied\_for}(x, \mathit{test})]$	Every student studied for the test.	The $\forall$ symbol makes a statement about every possible object (whether a student or not). It temporarily gives it the name x to make such a statement. We use the $\Longrightarrow$ symbol because we don't want to make any claims about whether non-students studied.
$\exists$	$[\exists_x : \mathit{student}(x) \land \mathit{aced}(x, \mathit{test})]$	There exists at least one student who aced the test.	The $\exists$ symbol makes the claim that there is at least one (possibly more) object in the universe, temporarily called x, that satisfies the statement listed.

Bertrand and Russell also note that there are also a couple other things we can say about individuals (but not propositions or quantifiers). For example, if the names Jonathan and Jon both refer to the same person, we can say $Jon = Jonathan$. If we want to emphasize that John and Jon are different people, we can say $John \neq Jon$.

H1. Translate Mr. Whitehead's statements into first-order logic by finding the proposition below that is equivalent to each statement and writing the letter of the proposition in the blank. Each statement has exactly one correct answer; not every proposition will be used.

	Everyone either passed or failed the test.
	Every student did not pass the test.
	Exactly one student passed the test.
	A student did not pass the test.
	It is not the case that if at least one student studied for the test, then every student failed the test.

A.	$[\exists_x : student(x) \land \neg\, passed(x, test)]$
B.	$[\exists_x : student(x) \land passed(x, test) \land [\forall_y : passed(y, test) \implies x = y]]$
C.	$[\exists_x : student(x) \land passed(x, test) \land [\exists_y : passed(y, test) \land x = y]]$
D.	$[\forall_x : passed(x, test) \lor failed(x, test)]$
E.	$\neg\, ([\exists_x : student(x) \land studied\_for(x, test)] \implies [\forall_x : student(x) \implies failed(x, test)])$
F.	$[\exists_x : passed(x, test) \land failed(x, test)]$
G.	$[\forall_x : \neg\, student(x) \implies passed(x, test)]$
H.	$[\exists_x : student(x) \land studied\_for(x, test)] \implies \neg\, [\forall_x : student(x) \implies failed(x, test)]$
I.	$\neg\, [\exists_x : student(x) \land \neg\, passed(x, test)]$
J.	$[\forall_x : student(x) \implies \neg\, passed(x, test)]$

H2. Translate first-order logic propositions into their equivalent English sentences by finding the statement below that is equivalent to each proposition and writing the letter of the statement in the blank. Each proposition has exactly one correct answer; not every statement will be used.

	$[\forall_x : student(x) \implies studied\_for(x, test)] \vee [\forall_y : student(y) \implies passed(y, test)]$
	$[\forall_x : student(x) \implies [studied\_for(x, test) \vee passed(x, test)]]$
	$[\forall_x : (test(x) \wedge long(x)) \implies hard(x)]$
	$[\exists_x : test(x) \wedge (long(x) \vee hard(x))]$
	$[\forall_x : test(x) \wedge \neg\, (long(x) \wedge hard(x)) \implies \neg\, [\forall_y : student(y) \implies failed(y, x)]]$

A.	There is a test that is long or hard.
B.	If a test is not long and not hard, then every student did not fail it.
C.	Every student studied for or passed the test.
D.	Every test that is long is also hard.
E.	Every student studied for the test or every student passed the test.
F.	If there is a test that is hard or not long, then at least one student failed it.
G.	Every test is long and hard.
H.	If a test is not both long and hard, then not every student failed it.

Solutions

1. D
   J
   B
   A
   E

2. E
   C
   D
   A
   H

AI

Artificial Intelligence

Inference in FOL (Ch. 9)

Outline

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

A brief history of reasoning

450 B.C.	Stoics	propositional logic, inference (maybe)
322 B.C.	Aristotle	“syllogisms” (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	“practical” algorithm for propositional logic
1965	Robinson	“practical” algorithm for FOL—resolution

Modus Ponens

- Modus ponens:

$$\frac{\alpha \qquad \alpha \Rightarrow \beta}{\beta}$$

- Example:

$$\frac{\mathit{Cat(Martin)} \qquad \forall x: \mathit{Cat}(x) \Rightarrow \mathit{EatsFish}(x)}{\mathit{EatsFish(Martin)}}$$

FOL Examples

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Rightarrow Sibling(x,y)$

- One's mother is one's female parent

  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- “Sibling” is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

Inference

- Forward chaining
  - as individual facts are added to the database, all derived inferences are generated
- Backward chaining
  - starts from queries
  - Example: the Prolog programming language
- Prolog example
  - ```
    father(X, Y) :- parent(X, Y), male(X).
    parent(john, bill).
    parent(jane, bill).
    female(jane).
    male (john).
    ?- father(M, bill).
    ```
    ```

Universal Instantiation

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

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

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

- E.g., $\forall x\ Cat(x) \wedge Fish(y) \Rightarrow Eats(x,y)$ yields:
  $Cat(Martin) \wedge Fish(Blub) \Rightarrow Eats(Martin,Blub)$

Existential Instantiation

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

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

- E.g., $\exists x\ Cat(x) \wedge EatsFish(x)$ yields:

$$Cat(C_1) \wedge EatsFish(C_1)$$

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

Universal instantiation (UI)

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

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

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

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

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

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

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

Existential instantiation (EI)

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

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

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

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

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)

- 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

Theorem: Turing (1936), Church (1936) Entailment for FOL is semidecidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.)

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.

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)*

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

- $\text{Unify}(\alpha,\beta) = \theta$ 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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} works

Whom does John know?

- Unify(α,β) = θ if αθ = βθ

p	q	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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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} works

- 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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} works

- 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)	

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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} works

- 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}

- Standardizing apart eliminates overlap of variables, e.g., Knows(z_{17},OJ)

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 }

The unification algorithm

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
    inputs: x, a variable, constant, list, or compound
            y, a variable, constant, list, or compound
            θ, the substitution built up so far

    if θ = failure **then return** failure
    else if $x = y$ **then return** θ
    else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
    else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
    else if COMPOUND?(x) **and** COMPOUND?(y) **then**
        return UNIFY(ARGS[x], ARGS[y], UNIFY(OP[x], OP[y], θ))
    else if LIST?(x) **and** LIST?(y) **then**
        return UNIFY(REST[x], REST[y], UNIFY(FIRST[x], FIRST[y], θ))
    else return failure

The unification algorithm

```
function UNIFY-VAR(var, x, θ) returns a substitution
    inputs: var, a variable
            x, any expression
            θ, the substitution built up so far

    if {var/val} ∈ θ then return UNIFY(val, x, θ)
    else if {x/val} ∈ θ then return UNIFY(var, val, θ)
    else if OCCUR-CHECK?(var, x) then return failure
    else return add {var/x} to θ
```

Another Example

- Example
  - p=Eats(x,y), q=Eats(x,Blub), possible if θ = {y/Blub}
  - p=Eats(Martin,y), q=Eats(x,Blub), possible if θ = {x/Martin,y/Blub}
  - p=Eats(Martin,y), q=Eats(y,Blub), fails because Martin$\neq$Blub
- Subsumption
  - Unification works not only when two things are the same but also when one of them subsumes the other one
  - Example: All cats eat fish, Martin is a cat, Blub is a fish

Subsumption Lattice

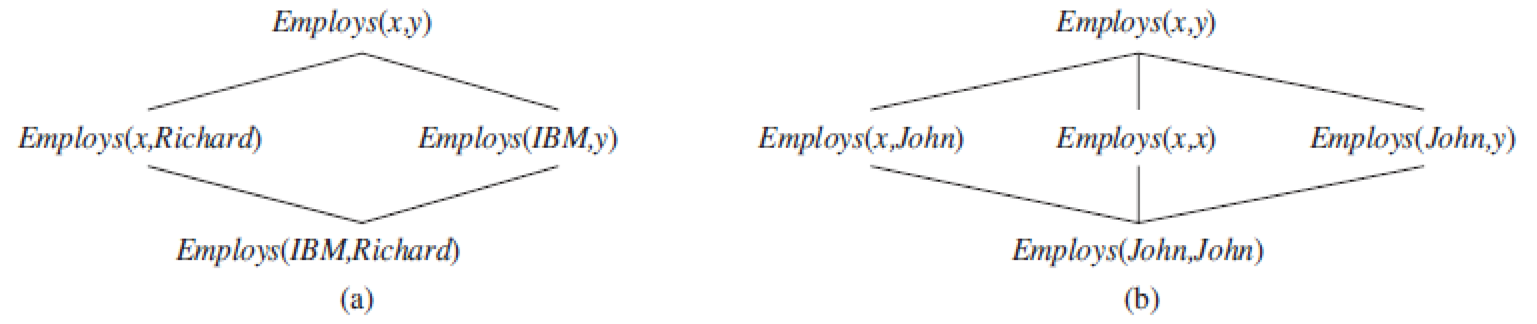


Figure 9.2 (a) The subsumption lattice whose lowest node is $Employs(IBM, Richard)$. (b) The subsumption lattice for the sentence $Employs(John, John)$.

Generalized Modus Ponens (GMP)

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

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

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

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

θ is {x/John,y/John} q is *Evil*(*x*)

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

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

Soundness of GMP

- Need to show that

$$p_1', \ldots, p_n', (p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models q\theta$$

provided that $p_i'\theta = p_i\theta$ for all *I*

- Lemma: For any sentence *p*, we have $p \models p\theta$ by UI

1. $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \models (p_1 \wedge \ldots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \ldots \wedge p_n\theta \Rightarrow q\theta)$
2. p_1', \; …, \;p_n' $\models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

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) and Missile(M_1)

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

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

Missiles are weapons:

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

An enemy of America counts as "hostile“:

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

West, who is American …

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

"The law says that it is a crime for an American to sell weapons to a hostile nation. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is an American." Prove Colonel West is a criminal.

"The law says that it is a crime for an American to sell weapons to a hostile nation."

1. $\forall$x,y,z american(x) $\wedge$ weapon(y) $\wedge$ nation(z) $\wedge$ hostile(z) $\wedge$ sells(x,y,z) $\rightarrow$ criminal(x)

"The country Nono is an enemy of America."

2. <u>nation(Nono)</u>
3. enemy(Nono,America)
4. $\forall$x nation(x) $\wedge$ hostile(x,America) $\rightarrow$ hostile(x)

"The country Nono has some missiles."

5. $\exists$x missile(x) $\wedge$ owns(Nono,x)
6. $\forall$x missile(x) $\rightarrow$ weapon(x)

"All of Non's missiles were sold to it by Colonel West."

7. $\forall$x missile(x) $\wedge$ owns(Nono,x) $\rightarrow$ sells(West,x,Nono)

“Colonel West is an American.”

8. <u>American(West)</u>

9. missile(m_1) $\wedge$ owns(Nono,m1) [existential instantiation/skolemization of 5, q={x/m_1}]

10. missile(m_1) [and elimination on 9]

11. <u>weapon(m_1)</u> [modus ponens on 6 an 10, q={x/m_1}]

12. <u>hostile(Nono)</u> [modus ponens on 2, 3, and 4, q={x/m_1}]

13. <u>sells(West, m_1,Nono)</u> [modus ponens on 7 an 9, q={x/m_1}]

14. criminal(West) [modus ponens on 1, 8, 2, 12, 11, and 13, q={x/West,y/m_1,z/Nono }]

Forward chaining algorithm

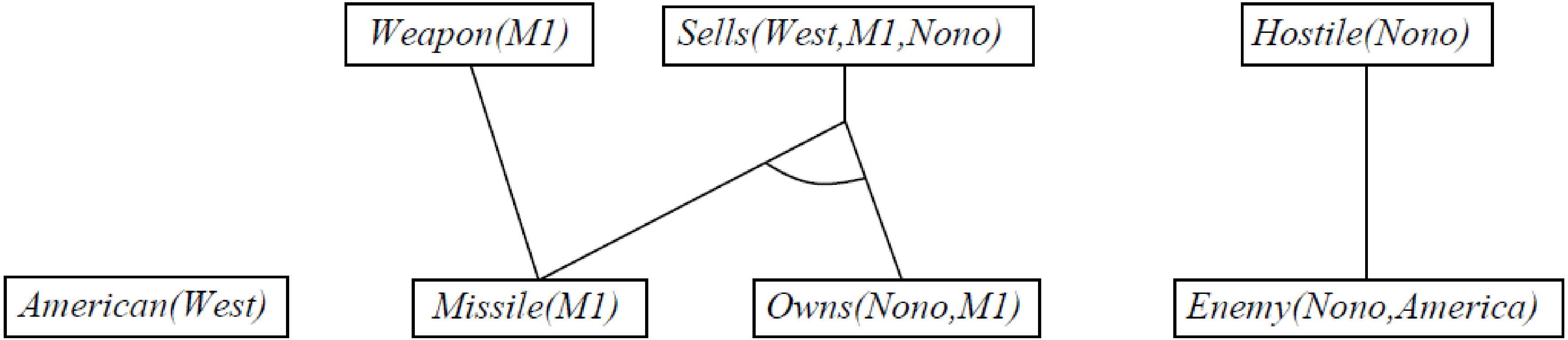
function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

    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 θ 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(θ, 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', α)
                    if ϕ is not *fail* **then return** ϕ
        add *new* to KB
    return *false*

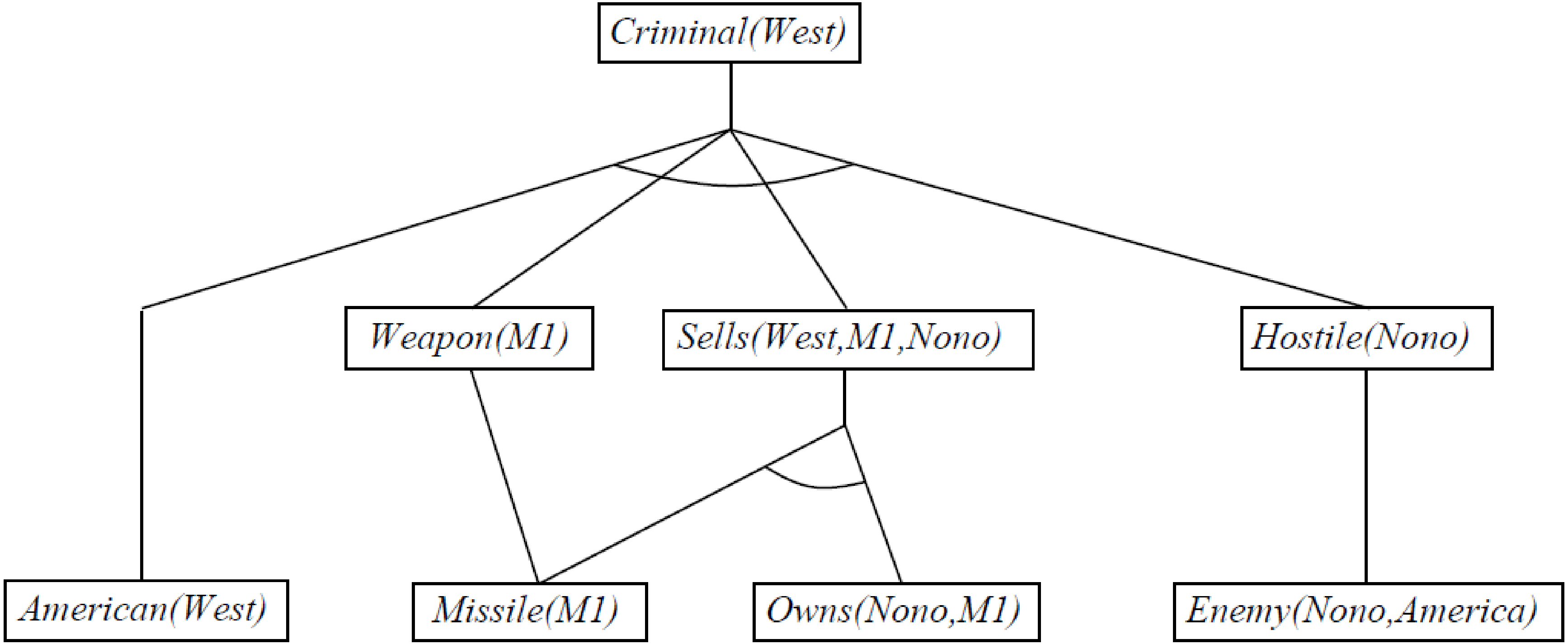
Forward chaining proof

$\boxed{American(West)}$ $\boxed{Missile(M1)}$ $\boxed{Owns(Nono,M1)}$ $\boxed{Enemy(Nono,America)}$

Forward chaining proof



Forward chaining proof



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

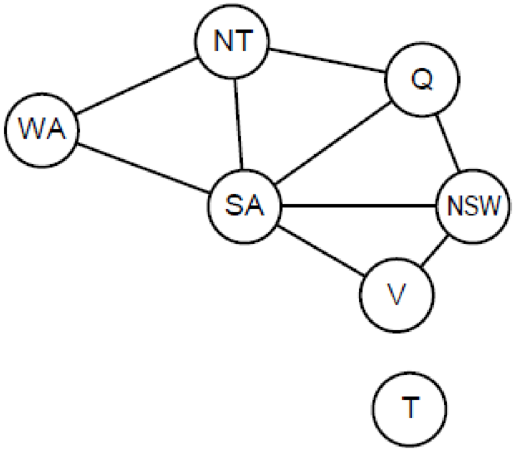
Matching itself can be expensive:

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

Hard matching example



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

Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)* *Diff(Blue,Red)*
Diff(Blue,Green)

- *Colorable*() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

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

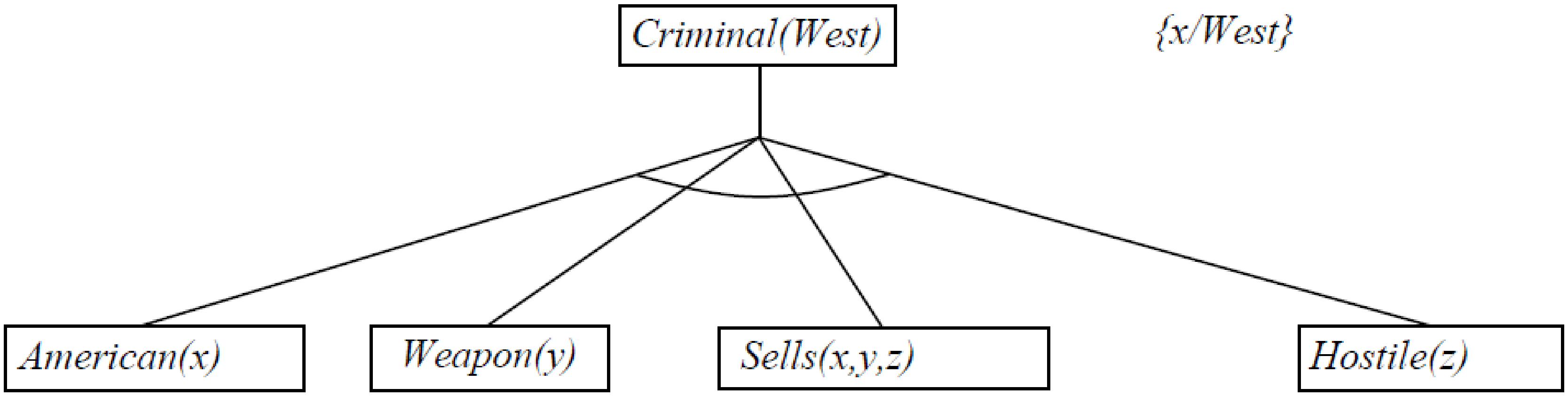
    if goals is empty then return {θ}
    q′ ← SUBST(θ, FIRST(goals))
    for each sentence r in KB
            where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
            and θ′ ← UNIFY(q, q′) succeeds
        new_goals ← [ p1, ..., pn|REST(goals)]
        answers ← FOL-BC-ASK(KB, new_goals, COMPOSE(θ′, θ)) ∪ answers
    return answers
```

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

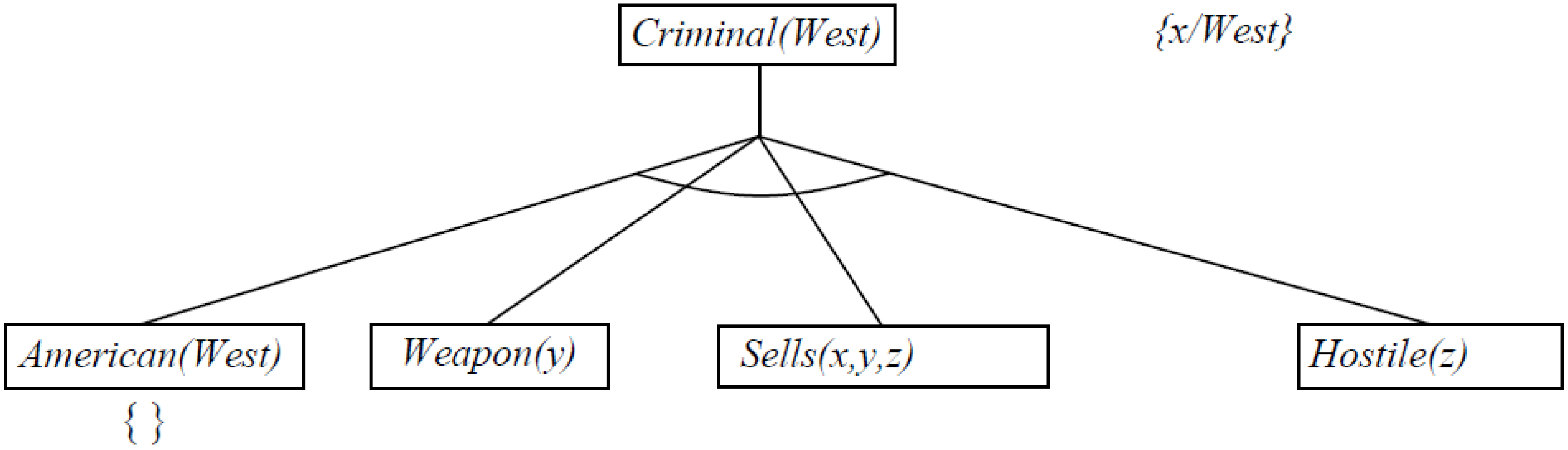
Backward chaining example

$$\boxed{Criminal(West)}$$

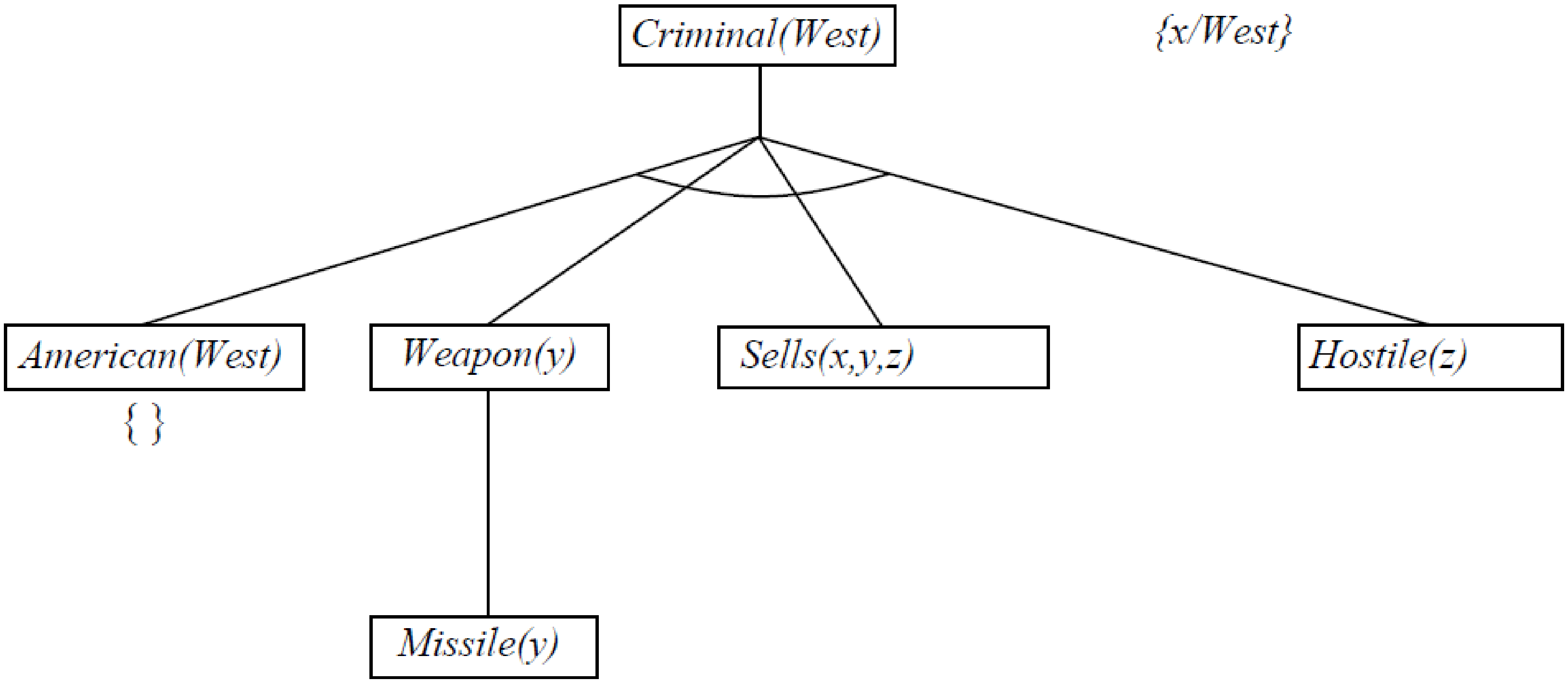
Backward chaining example



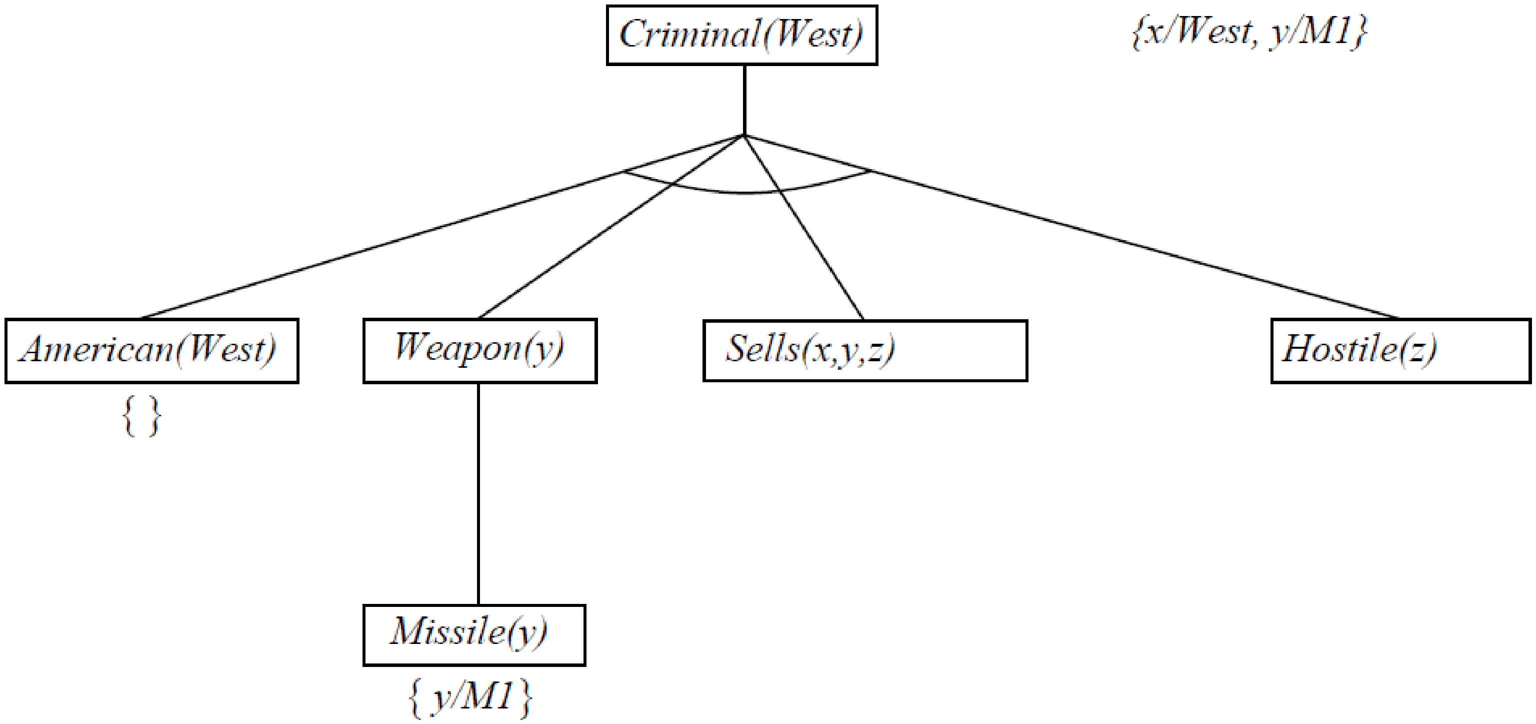
Backward chaining example



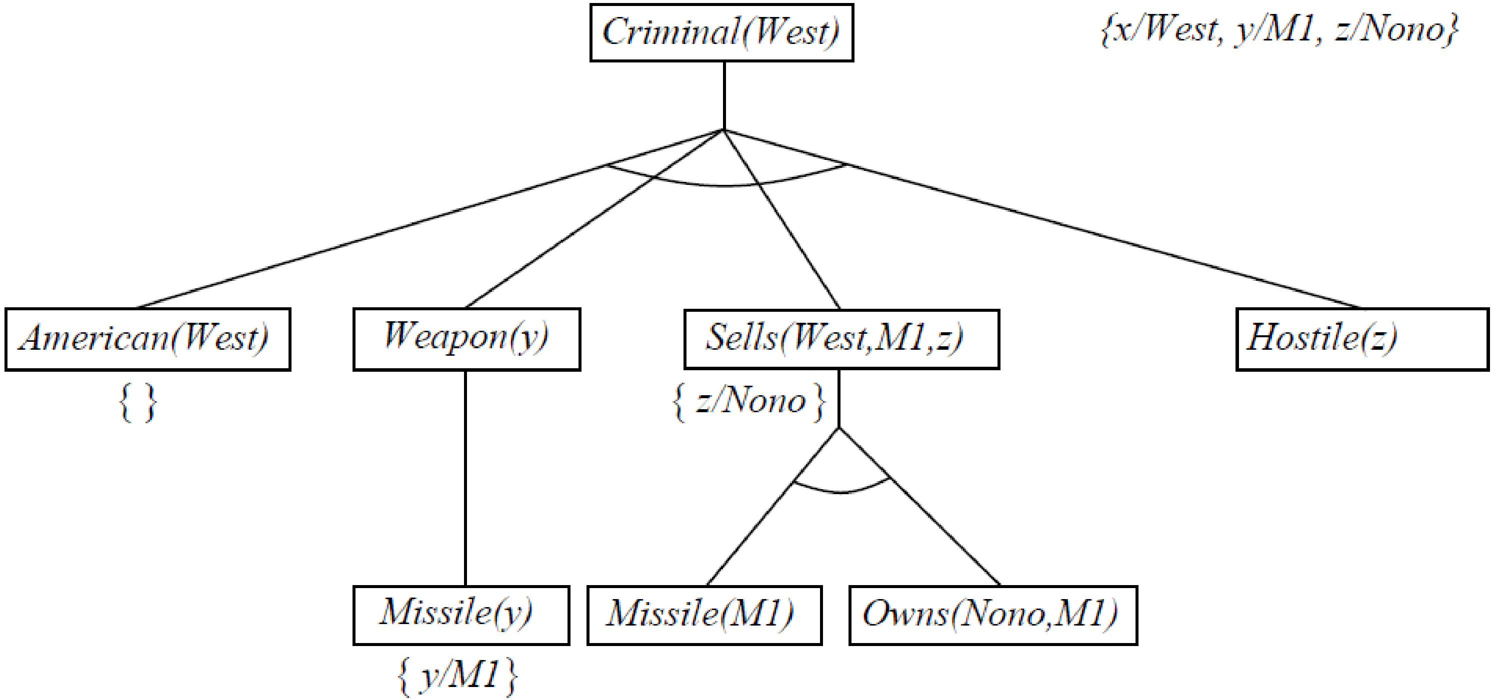
Backward chaining example



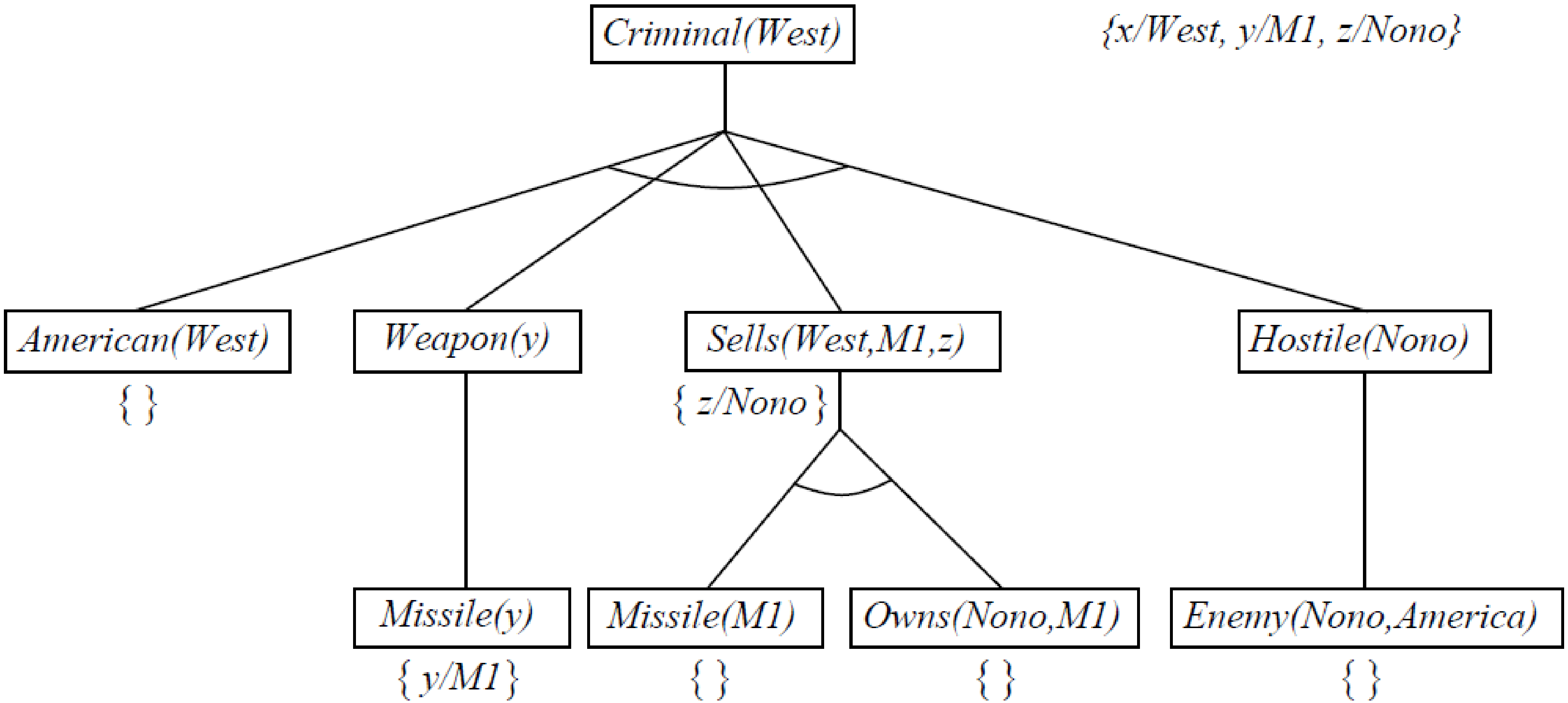
Backward chaining example



Backward chaining example



Backward chaining example



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

Logic programming: Prolog

- Algorithm = Logic + Control

- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS

- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$`.`

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

- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output
- predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails

Prolog

- Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- query: `append(A,B,[1,2]) ?`

- answers:

```
A=[]     B=[1,2]
A=[1]    B=[2]
A=[1,2]  B=[]
```

Resolution in FOL

- FOL includes non-Horn clauses, e.g.,
  - $\forall x$: Country(x) $\rightarrow \exists y$: CapitalOf(y,x)
- Strategy (just like in propositional logic)
  - Convert all formulas to CNF
  - Apply resolution rule

Resolution: brief summary

- 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 $\texttt{Unify}(\ell_i, \neg m_j) = \theta$.

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

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

with $\theta = \{x/Ken\}$

- Apply resolution steps to CNF(KB $\wedge \neg\alpha$); complete for FOL

Conversion to CNF

- Everyone who loves all animals is loved by someone:

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

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: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\exists x\ p \equiv \forall x\ \neg p$

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

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

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

Conversion to CNF contd.

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

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

4. Skolemize: a more general form of existential instantiation.

   Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

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

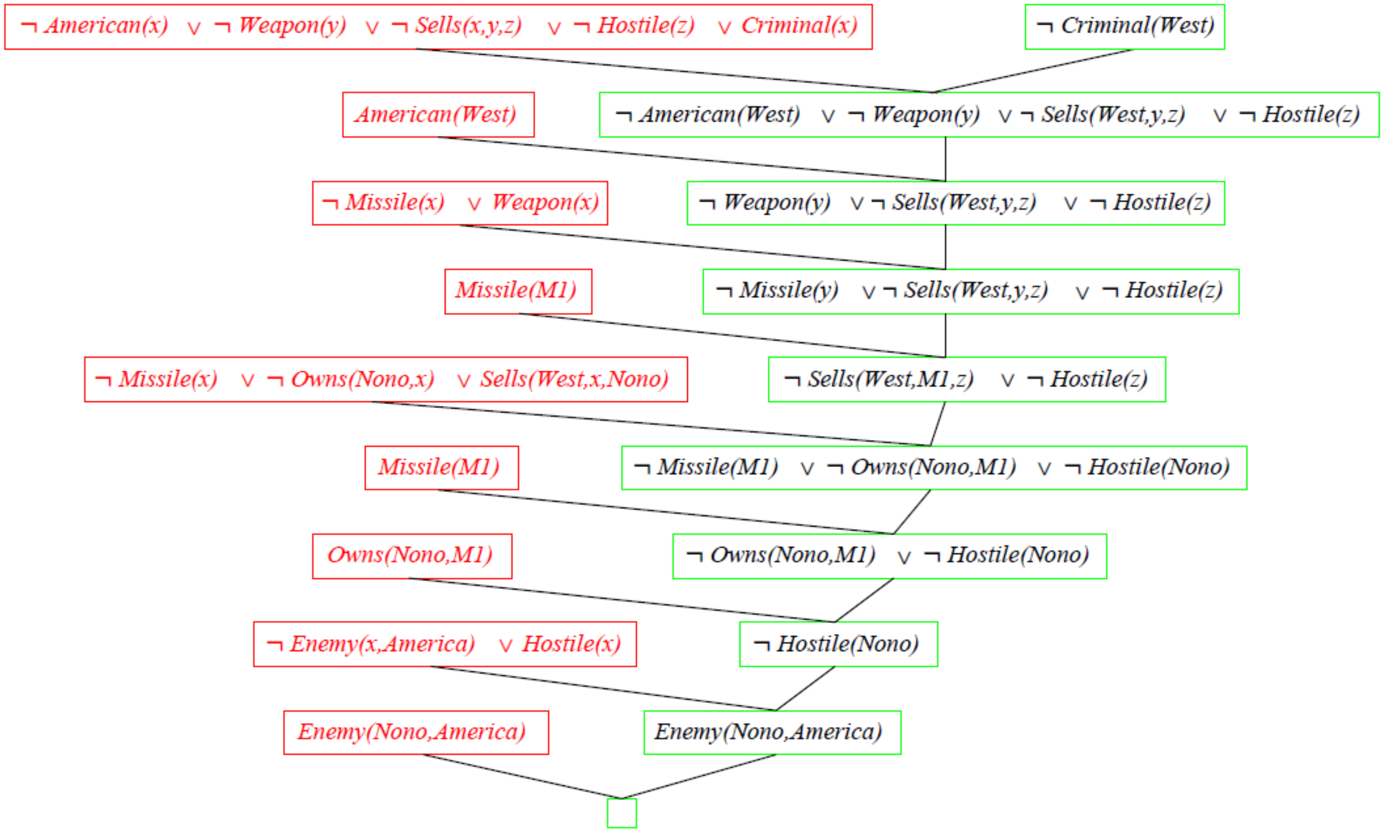
5. Drop universal quantifiers:

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

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

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

Resolution proof: definite clauses



Everyone who loves all animals is loved by someone.
Anyone who kills an animal is loved by no one.
Jack loves all animals.
Either Jack or Curiosity killed the cat, who is named Tuna.
Did Curiosity kill the cat?

First, we express the original sentences, some background knowledge, and the negated goal G in first-order logic:

A.	$\forall x\ [\forall y\ \ Animal(y) \Rightarrow Loves(x, y)] \Rightarrow [\exists y\ \ Loves(y, x)]$
B.	$\forall x\ [\exists z\ \ Animal(z) \wedge Kills(x, z)] \Rightarrow [\forall y\ \ \neg Loves(y, x)]$
C.	$\forall x\ \ Animal(x) \Rightarrow Loves(Jack, x)$
D.	$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E.	$Cat(Tuna)$
F.	$\forall x\ \ Cat(x) \Rightarrow Animal(x)$
¬G.	$\neg Kills(Curiosity, Tuna)$

Now we apply the conversion procedure to convert each sentence to CNF:

A1.	$Animal(F(x)) \vee Loves(G(x), x)$
A2.	$\neg Loves(x, F(x)) \vee Loves(G(x), x)$
B.	$\neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kills(x, z)$
C.	$\neg Animal(x) \vee Loves(Jack, x)$
D.	$Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E.	$Cat(Tuna)$
F.	$\neg Cat(x) \vee Animal(x)$
¬G.	$\neg Kills(Curiosity, Tuna)$

The resolution proof that Curiosity killed the cat is given in Figure 9.12. In English, the proof could be paraphrased as follows:

> Suppose Curiosity did not kill Tuna. We know that either Jack or Curiosity did; thus Jack must have. Now, Tuna is a cat and cats are animals, so Tuna is an animal. Because anyone who kills an animal is loved by no one, we know that no one loves Jack. On the other hand, Jack loves all animals, so someone loves him; so we have a contradiction. Therefore, Curiosity killed the cat.

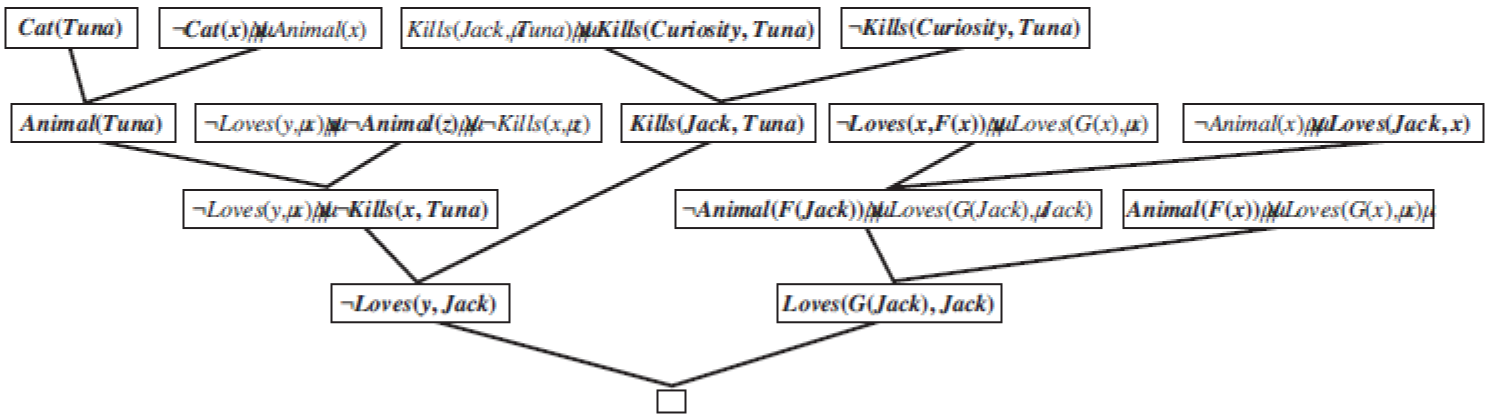


Figure 9.12 A resolution proof that Curiosity killed the cat. Notice the use of factoring in the derivation of the clause $Loves(G(Jack), Jack)$. Notice also in the upper right, the unification of $Loves(x, F(x))$ and $Loves(Jack, x)$ can only succeed after the variables have been standardized apart.

AI