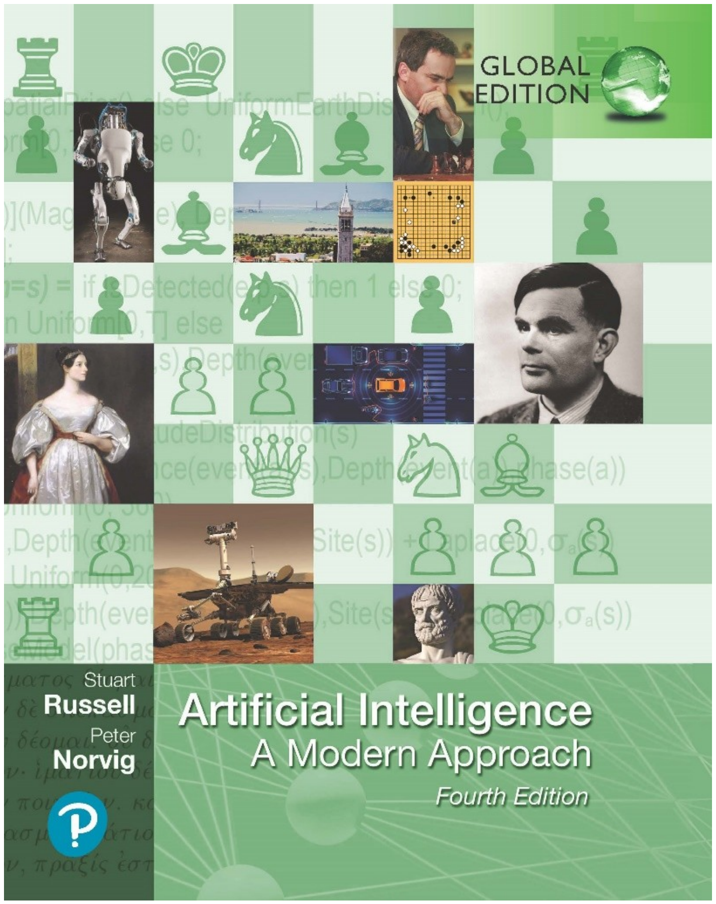
Artificial Intelligence: A Modern Approach

Fourth Edition, Global Edition



Chapter 8

First-order logic

Lecture Presentations: Artificial Intelligence

Adapted from:
"Artificial Intelligence: A Modern Approach, Global Edition", 4th Edition by Stuart Russell and Peter Norvig © 2021 Pearson Education.

Adapted for educational use at ACE Engineering College. Some slides customized by Mr. Shafakhatullah Khan Mohammed, Assistant Professor @ ACE Engineering College. For instructional use only. Not for commercial distribution.

Outline

- Why FOL?
- Syntax and semantics of FOL
- Fun with sentences
- Wumpus world in FOL
- Knowledge Engineering in FOL

Pros and cons of propositional logic

- Propositional logic is declarative: pieces of syntax correspond to facts
- 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 world contains facts,
first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- Relations: red, round, bogus, prime, multistoried . . .,
  brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, . . .
- Functions: father of, best friend, third inning of, one more than, end of . . .

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

Pearson

Syntax of FOL: Basic elements

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

Atomic sentences

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

Term = $function(term_1, \ldots, term_n)$
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

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

E.g. $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$
$>(1,2) \vee \leq(1,2)$
$>(1,2) \wedge \neg >(1,2)$

Truth in first-order logic

Sentences are true with respect to a model and an interpretation

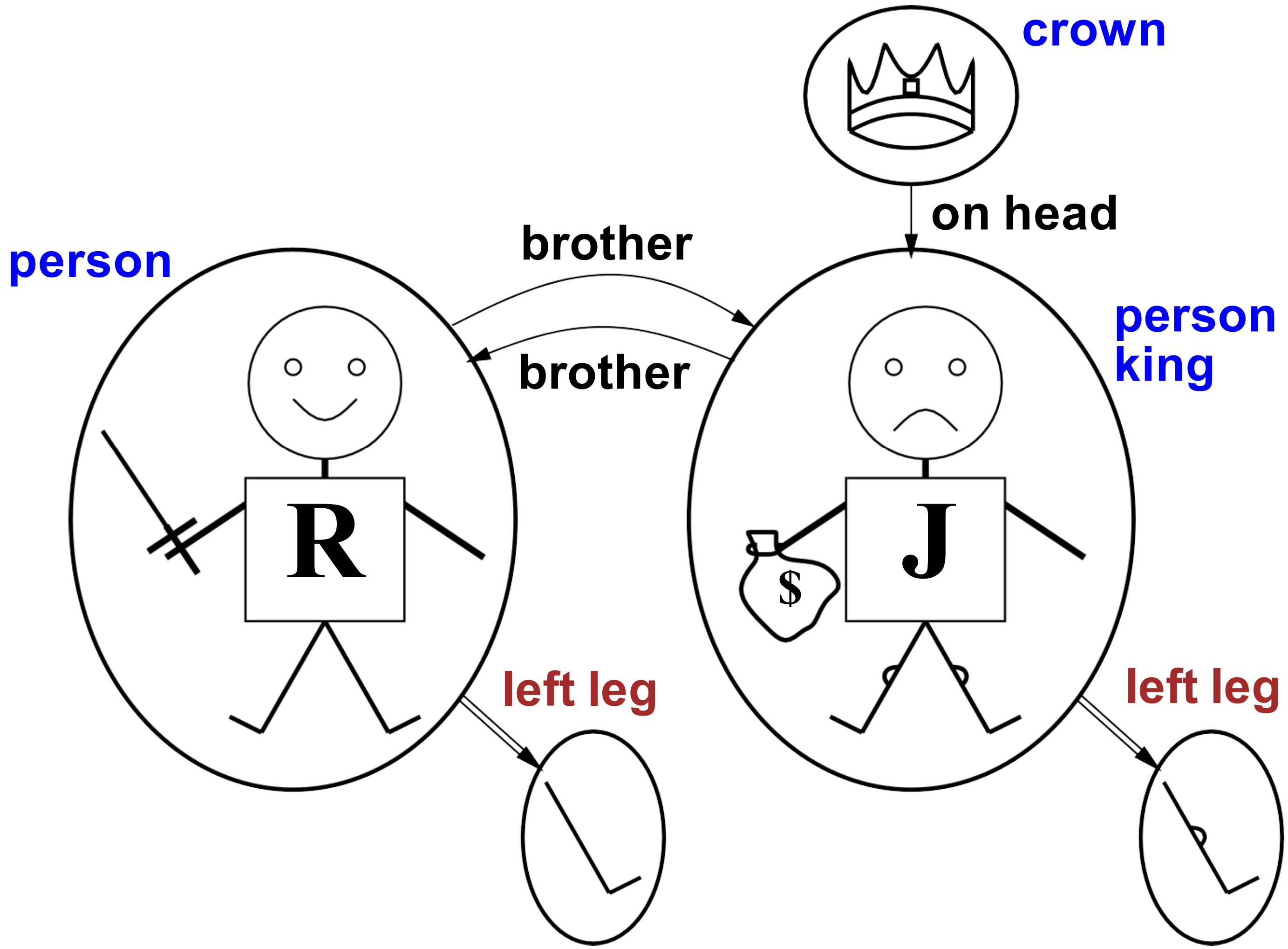
Model contains ≥ 1 objects (domain elements) and relations among them

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



Pearson

Truth example

Consider the interpretation in which
$Richard$ → Richard the Lionheart
$John$ → the evil King John
$Brother$ → 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

Models for FOL: Lots!

Entailment in propositional logic can be computed by enumerating models

We can enumerate the FOL models for a given KB vocabulary:

For each number of domain elements n from 1 to ∞

    For each k-ary predicate P_k in the vocabulary

        For each possible k-ary relation on n objects

            For each constant symbol C in the vocabulary

                For each choice of referent for C from n objects . . .

Computing entailment by enumerating FOL models is not easy!

Universal quantification

$\forall \langle variables \rangle \langle sentences \rangle$

Everyone at Berkeley is smart:
$\forall x \; At(x, Berkeley) \Rightarrow Smart(x)$

$\forall x \; P$ 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

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

A common mistake to avoid

Typically, $\Rightarrow$ is the main connective with $\forall$

Common mistake: using $\wedge$ as the main connective with $\forall$:

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

means "Everyone is at Berkeley and everyone is smart"

Existential quantification

$\exists \langle variables \rangle \langle sentence \rangle$

Someone at Stanford is smart:
$\exists x \;\; At(x, Stanford) \wedge Smart(x)$

$\exists x \;\; P$ 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

$$
\begin{aligned}
& (At(KingJohn, Stanford) \wedge Smart(KingJohn)) \\
\vee \;\; & (At(Richard, Stanford) \wedge Smart(Richard)) \\
\vee \;\; & (At(Stanford, Stanford) \wedge Smart(Stanford)) \\
\vee \;\; & \ldots
\end{aligned}
$$

Another common mistake to avoid

Typically, $\wedge$ is the main connective with $\exists$

Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists\, x \;\; At(x, Stanford) \;\Rightarrow\; Smart(x)$$

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

Properties of quantifiers

$\forall x\ \forall y$ is the same as $\forall y\ \forall x$ (why??)

$\exists x\ \exists y$ is the same as $\exists y\ \exists x$ (why??)

$\exists x\ \forall y$ 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

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

$\exists x\ Likes(x, Broccoli) \qquad \neg\forall x\ \neg Likes(x, Broccoli)$

Fun with sentences

Brothers are siblings

Fun with sentences

Brothers are siblings

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

"Sibling" is symmetric

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

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

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

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

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

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

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

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

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*:

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

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: $Yes, \; \{a/Shoot\}$ ← 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:

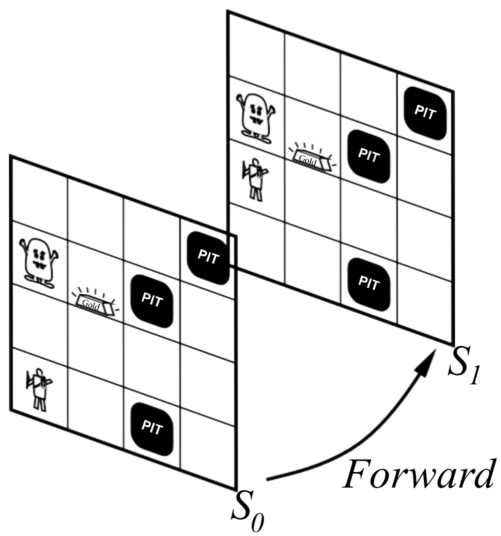
$$\forall y \;\; Breezy(y) \;\Leftrightarrow\; [\exists x \;\; Pit(x) \land Adjacent(x, y)]$$

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:

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

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

Knowledge engineering: the general process of knowledge-base construction.

The steps used in the knowledge engineering process:

1. Identify the questions.
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 problem instance
6. Pose queries to the inference procedure and get answers
7. Debug and evaluate the knowledge base

Knowledge Engineering in FOL

Applications in the electronic circuits domain

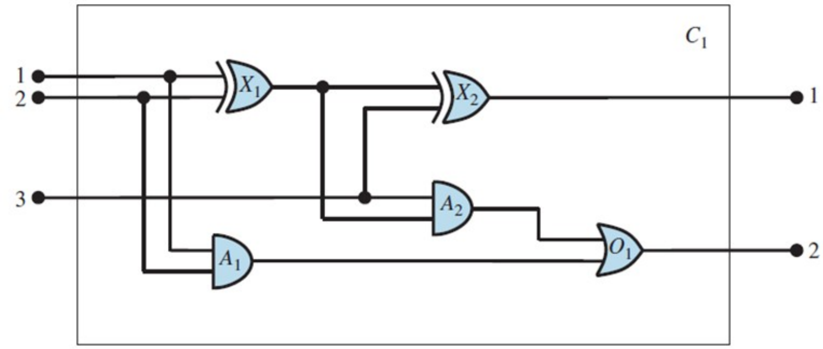


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

1. Identify the questions

- Does the circuit in Figure 8.6 actually add properly?
- If all the inputs are high, what is the output of gate A2?
- Questions about the circuit's structure are also interesting.
- For example, what are all the gates connected to the first input terminal?
- Does the circuit contain feedback loops?

Knowledge Engineering in FOL

Applications in the electronic circuits domain

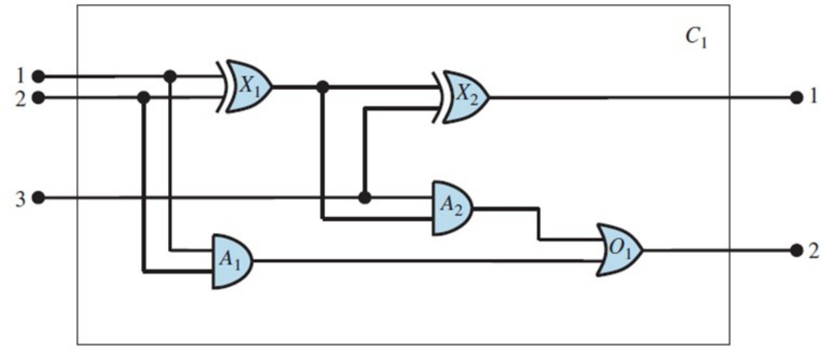


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

2. Assemble the relevant knowledge

- Circuits composed of wires and gates.
- Signals flow along wires to the input terminals of gates
- Each gate produces a signal on the output terminal that flows along another wire.
- There are four types of gates: AND, OR, and XOR gates have two input terminals, and NOT gates have one.

Knowledge Engineering in FOL

Applications in the electronic circuits domain

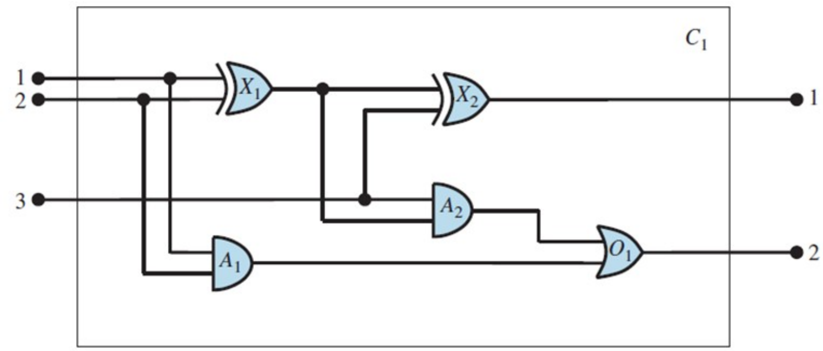


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

3. Decide on a vocabulary

- Each gate is represented as an object named by a constant, about which we assert that it is a gate with
- *Gate*(X_1), eg: *Type*(X_1)=*XOR*
- *Circuit*(C_1)
- *Terminal*(x)

Knowledge Engineering in FOL

Applications in the electronic circuits domain

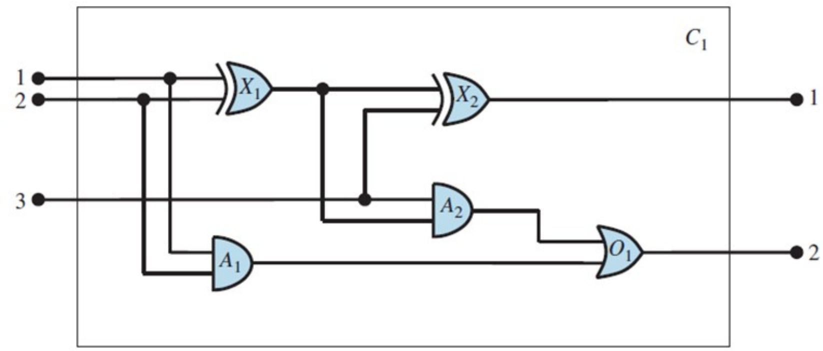


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

4. Encode general knowledge of the domain

- Example:

  If two terminals are connected, then they have the same signal:

$\forall\, t_1, t_2\; Terminal(t_1) \land Terminal(t_2) \land Connected(t_1, t_2) \Rightarrow$
$Signal(t_1)=Signal(t_2)$

Knowledge Engineering in FOL

Applications in the electronic circuits domain

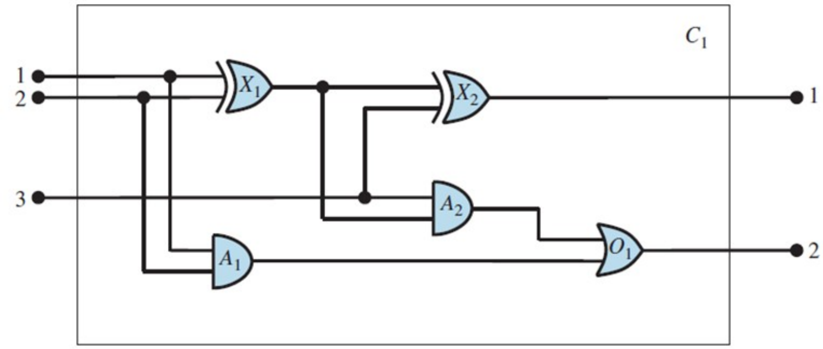


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

5. Encode the specific problem instance

- Categorize the circuit and its component gates & show the connections:

Connected(*Out*$(1,X_1)$, *In*$(1,X_2)$) *Connected*(*In*$(1,C_1)$; *In*$(1,X_1)$)

Knowledge Engineering in FOL

Applications in the electronic circuits domain

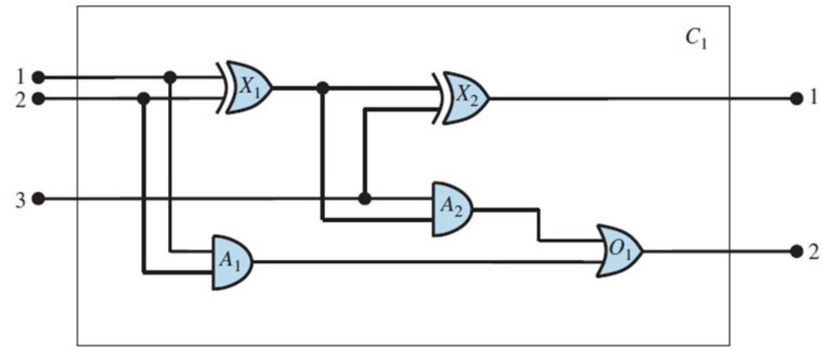


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

6. Pose queries to the inference procedure

- What are the possible sets of values of all the terminals for the adder circuit?
- This final query will return a complete input–output table for the device, which can be used

Knowledge Engineering in FOL

Applications in the electronic circuits domain

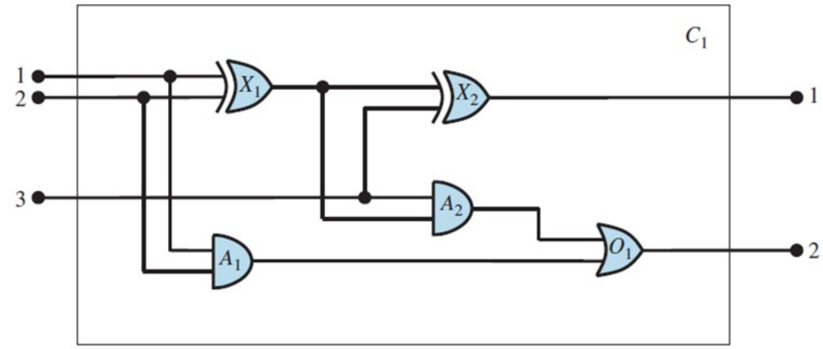


Figure 8.6 A digital circuit C_1, purporting to be a one-bit full adder. The first two inputs are the two bits to be added, and the third input is a carry bit. The first output is the sum, and the second output is a carry bit for the next adder. The circuit contains two XOR gates, two AND gates, and one OR gate.

7. Debug the knowledge base

- We can perturb the knowledge base in various ways to see what kinds of erroneous behaviors emerge
- Example if no assertion $1 \neq 0$

Summary

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers

Increased expressive power: sufficient to define

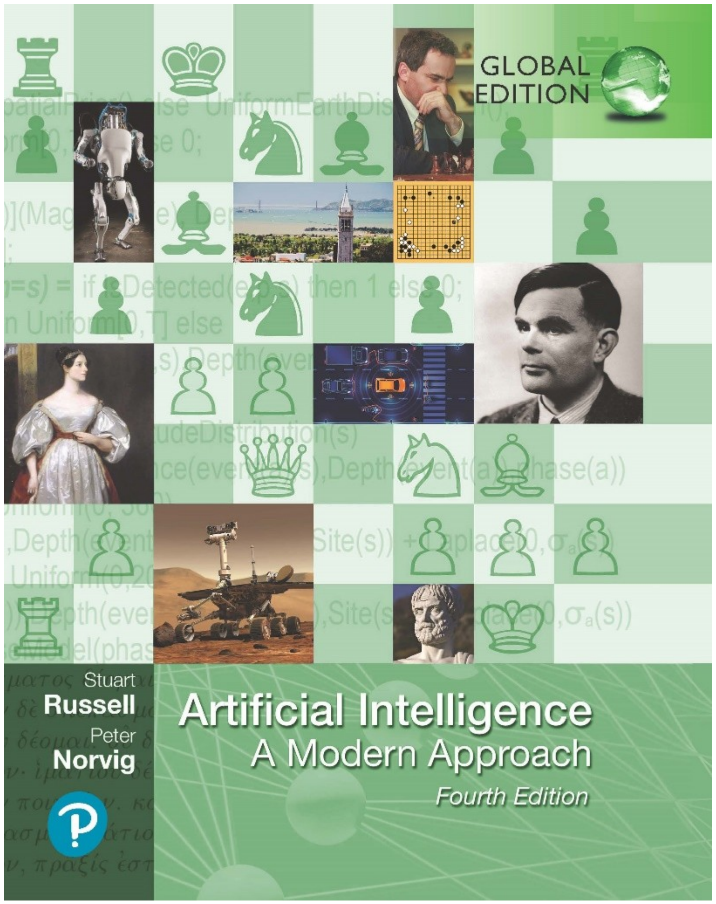
wumpus world

Situation calculus:
- conventions for describing actions and change in FOL
- can formulate planning as inference on a situation calculus KB

Developing a KB in FOL requires a careful process of analyzing the domain, choosing a vocabulary, and encoding the axioms required to support the desired inferences.

Artificial Intelligence: A Modern Approach

Fourth Edition, Global Edition



Chapter 9

Inference in first-order logic

Lecture Presentations: Artificial Intelligence

Adapted from:
"Artificial Intelligence: A Modern Approach, Global Edition", 4th Edition by Stuart Russell and Peter Norvig © 2021 Pearson Education.

Adapted for educational use at ACE Engineering College. Some slides customized by Mr. Shafakhatullah Khan Mohammed, Assistant Professor @ ACE Engineering College. For instructional use only. Not for commercial distribution.

Outline

- Reducing first-order inference to propositional inference
- Unification
- Generalized Modus Ponens
- Forward and backward chaining
- Logic programming
- Resolution

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

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

.

© 2022 Pearson Education Ltd.

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

Another example: from $\exists\, x\ \ d(x^y)/dy = x^y$ we obtain

$$d(e^y)/dy = e^y$$

provided e is a new constant symbol

Existential instantiation contd.

UI can be applied several times to add new sentences;
the new KB is logically equivalent to the old

EI can be applied once to replace the existential sentence;
the new KB is not equivalent to the old,
but is satisfiable iff the old KB was satisfiable

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.

Claim: a ground sentence* is entailed by new KB iff entailed by original KB

Claim: every FOL KB can be propositionalized so as to preserve entailment

Idea: propositionalize KB and query, apply resolution, return result

Problem: with function symbols, there are infinitely many ground terms,
e.g., $Father(Father(Father(John)))$

Theorem: Herbrand (1930). If a sentence α is entailed by an FOL KB,
it is entailed by a finite subset of the propositional 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 in FOL is semidecidable

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

With function symbols, it gets nuch much worse!

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
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	
$Knows(John, x)$	$Knows(x, OJ)$	

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

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 $Greedy(y)$

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

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

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

Unification

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

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

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

Generalized Modus Ponens (GMP)

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

$p_1{}'$ is $King(John)$ $\quad$ p_1 is $King(x)$
$p_2{}'$ is $Greedy(y)$ $\quad$ p_2 is $Greedy(x)$
θ is $\{x/John, y/John\}$ $\quad$ 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 definite clause 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. $p_1', \ldots, p_n' \models p_1' \wedge \ldots \wedge p_n' \models p_1'\theta \wedge \ldots \wedge p_n'\theta$
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:

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

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

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

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

Missiles are weapons:

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

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

Missiles are weapons:

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

An enemy of America counts as “hostile”:

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

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

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 ...

$Enemy(Nono, America)$

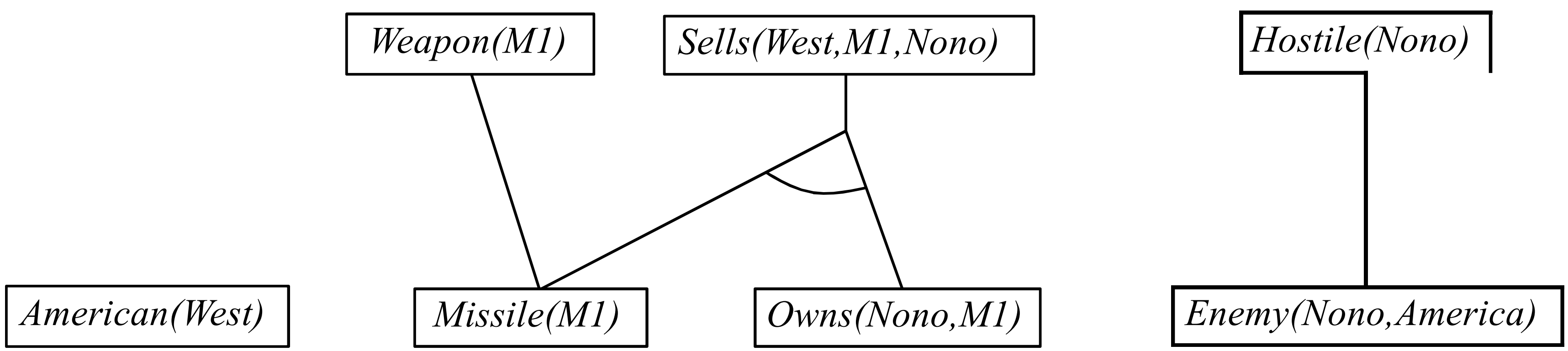
Forward chaining algorithm

```
function FOL-FC-Ask(KB, α) returns a substitution or false
    repeat until new is empty
        new ← { }
        for each sentence r in KB do
            (p1 ∧ . . . ∧ pn ⇒ q) ← Standardize-Apart(r)
            for each θ such that (p1 ∧ . . . ∧ pn)θ = (p'1 ∧ . . . ∧ p'n)θ
                            for some p'1, . . . , p'n in KB
                q' ← Subst(θ, q)
                if q' is not a renaming of a sentence already in KB or new then do
                    add q' to new
                    φ ← Unify(q', α)
                    if φ is not fail then return φ
        add new to KB
    return false
```

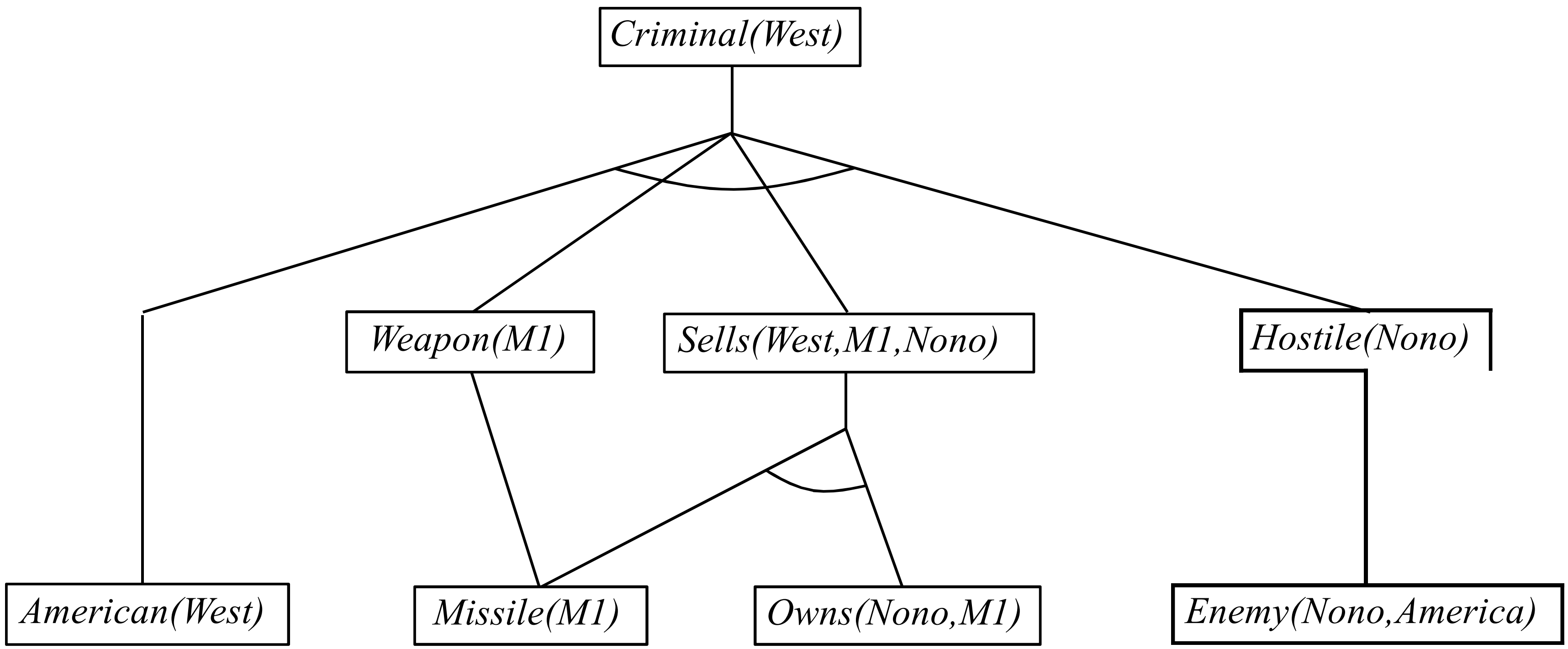
Forward chaining proof

American(West) *Missile(M1)* *Owns(Nono,M1)* *Enemy(Nono,America)*

Forward chaining proof



Forward chaining proof



Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + no functions (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if a is not entailed

This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

Simple observation: 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 literal

Matching itself can be expensive

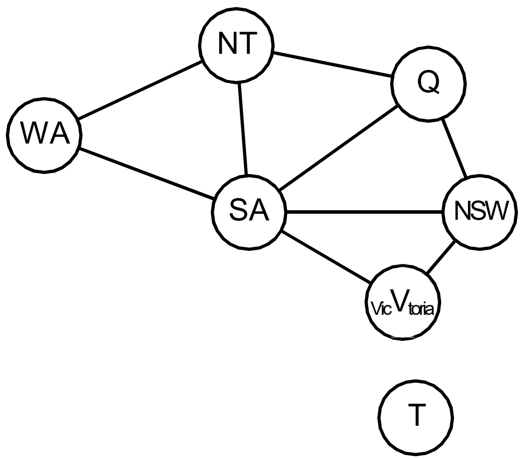
Database indexing allows $O(1)$ retrieval of known facts

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

Matching conjunctive premises against known facts is NP-hard

Forward chaining is widely used in deductive databases

Hard matching example



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

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

$Colorable()$ is inferred iff the CSP has a solution

CSPs include 3SAT as a special case, hence matching is NP-hard

Backward chaining algorithm

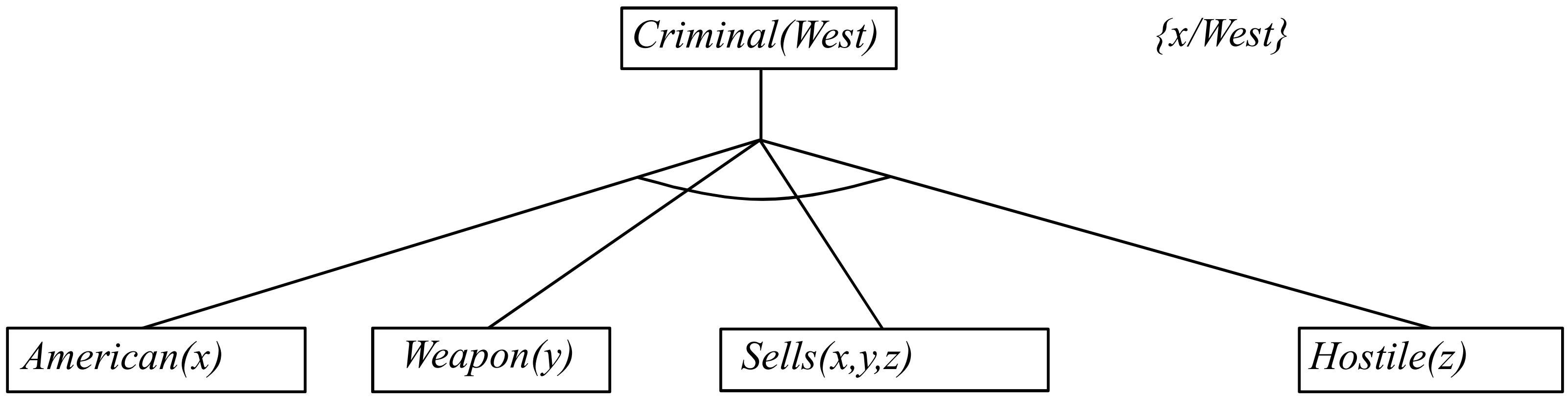
```
function FOL-BC-Ask(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query (θ already applied)
           θ, the current substitution, initially the empty substitution { }
   local variables: answers, a set of substitutions,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

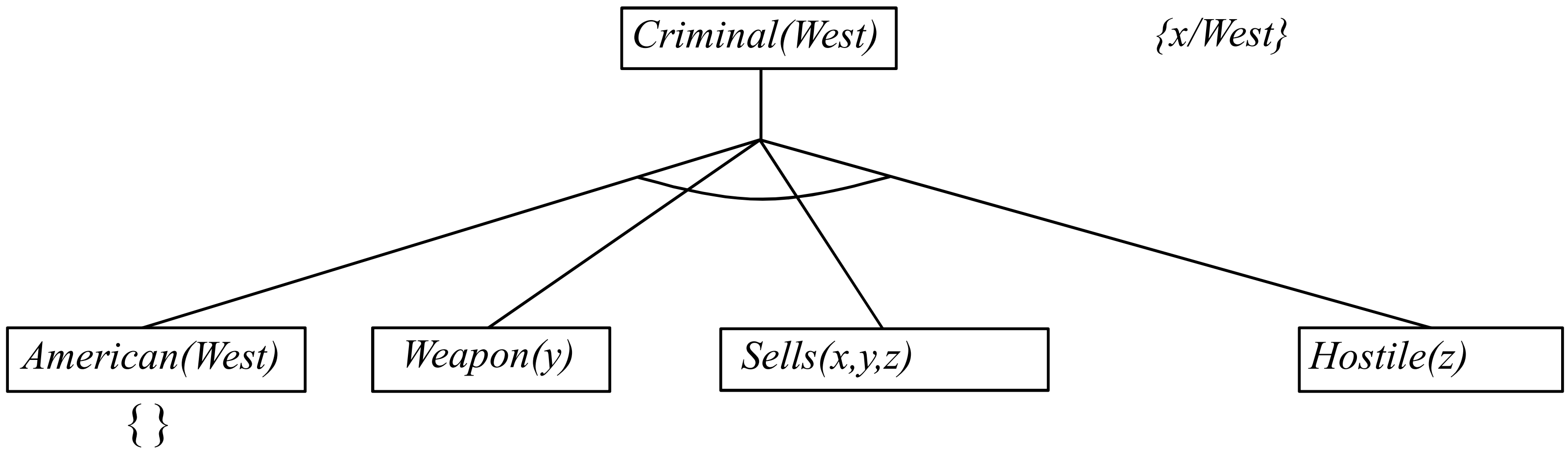
Backward chaining example

$\boxed{Criminal(West)}$

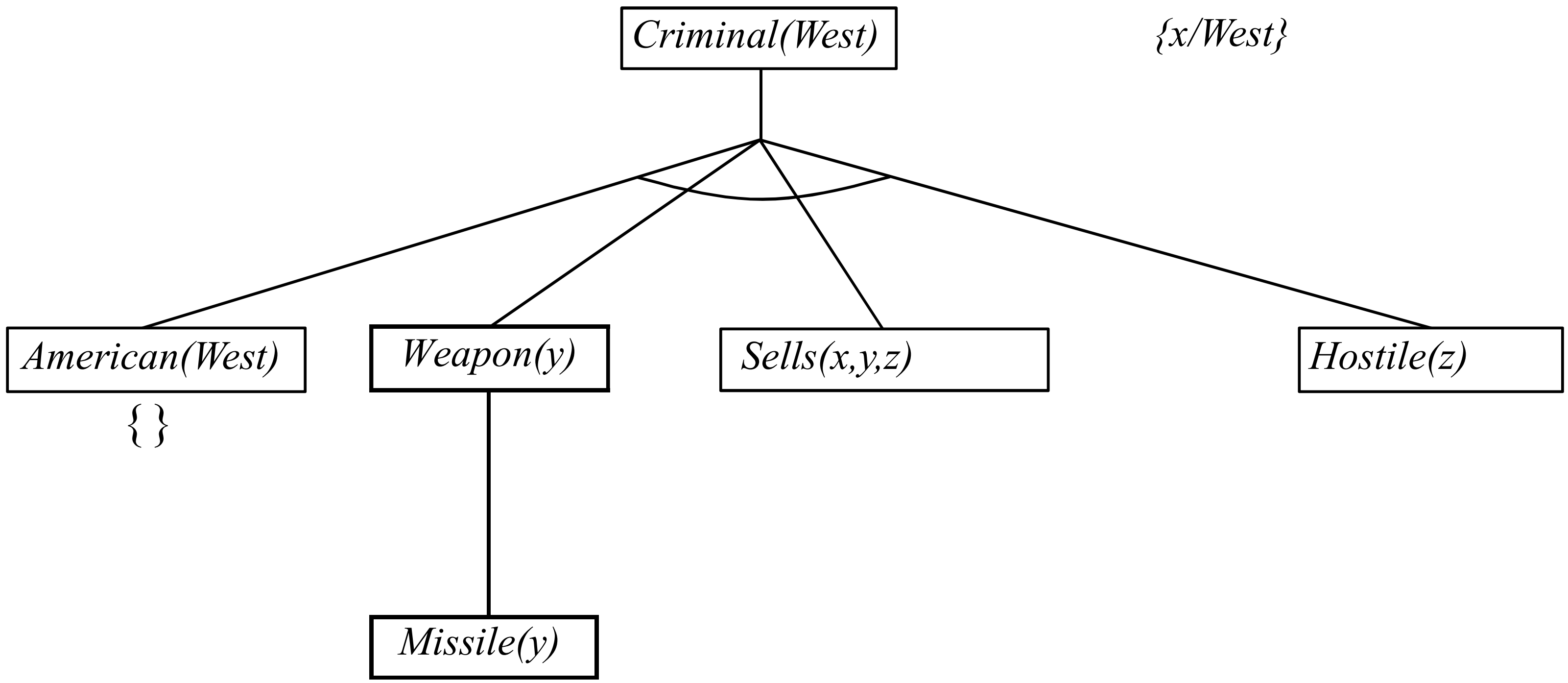
Backward chaining example



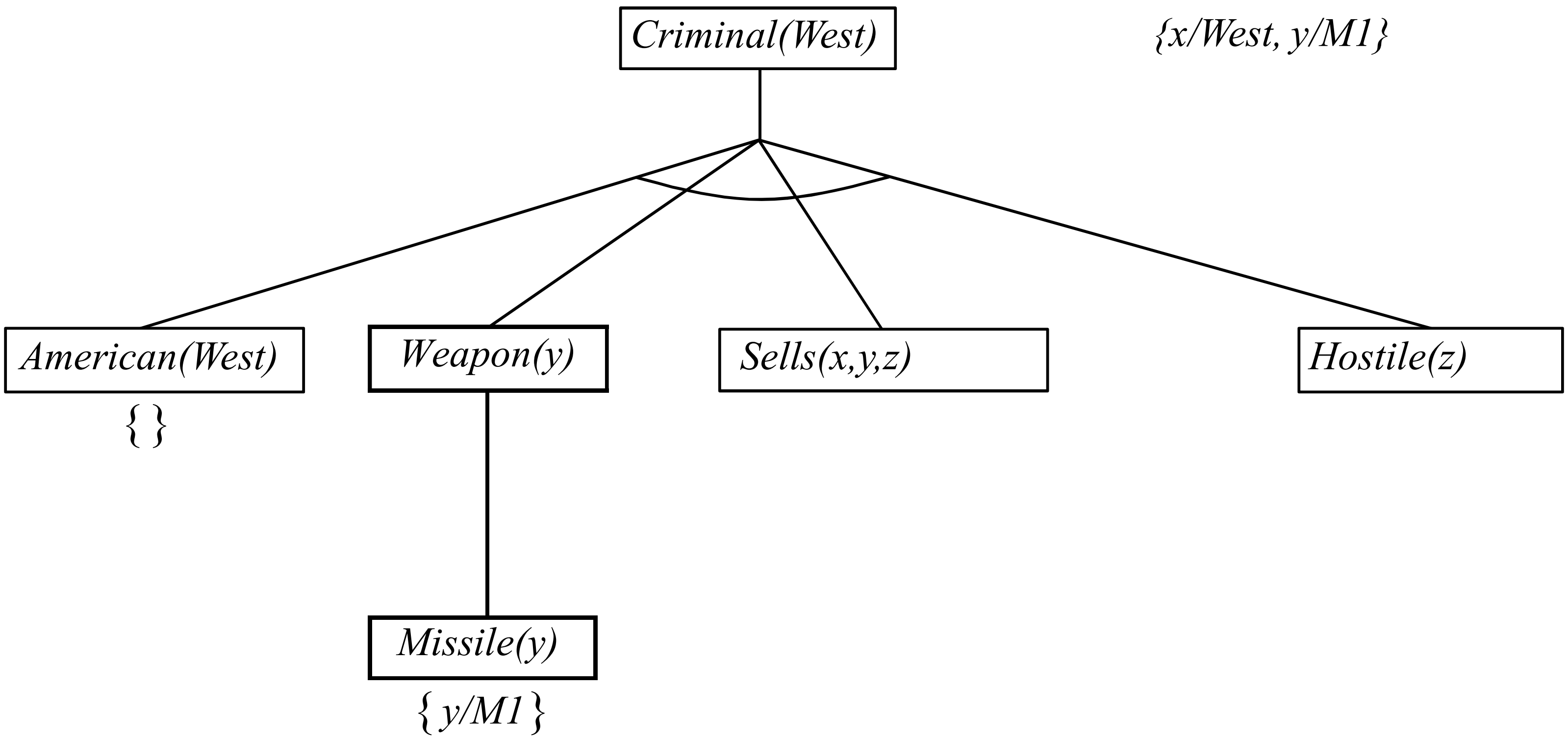
Backward chaining example



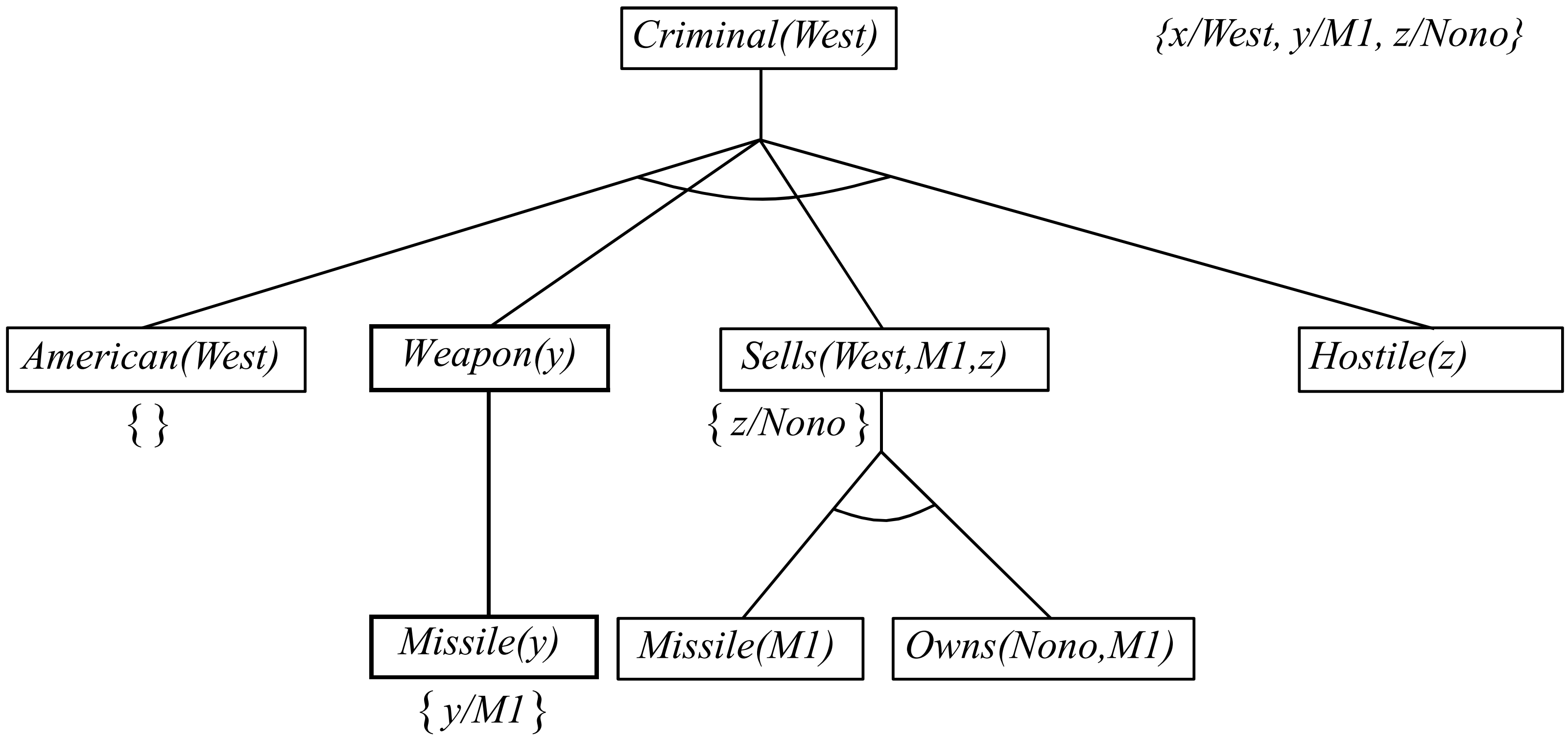
Backward chaining example



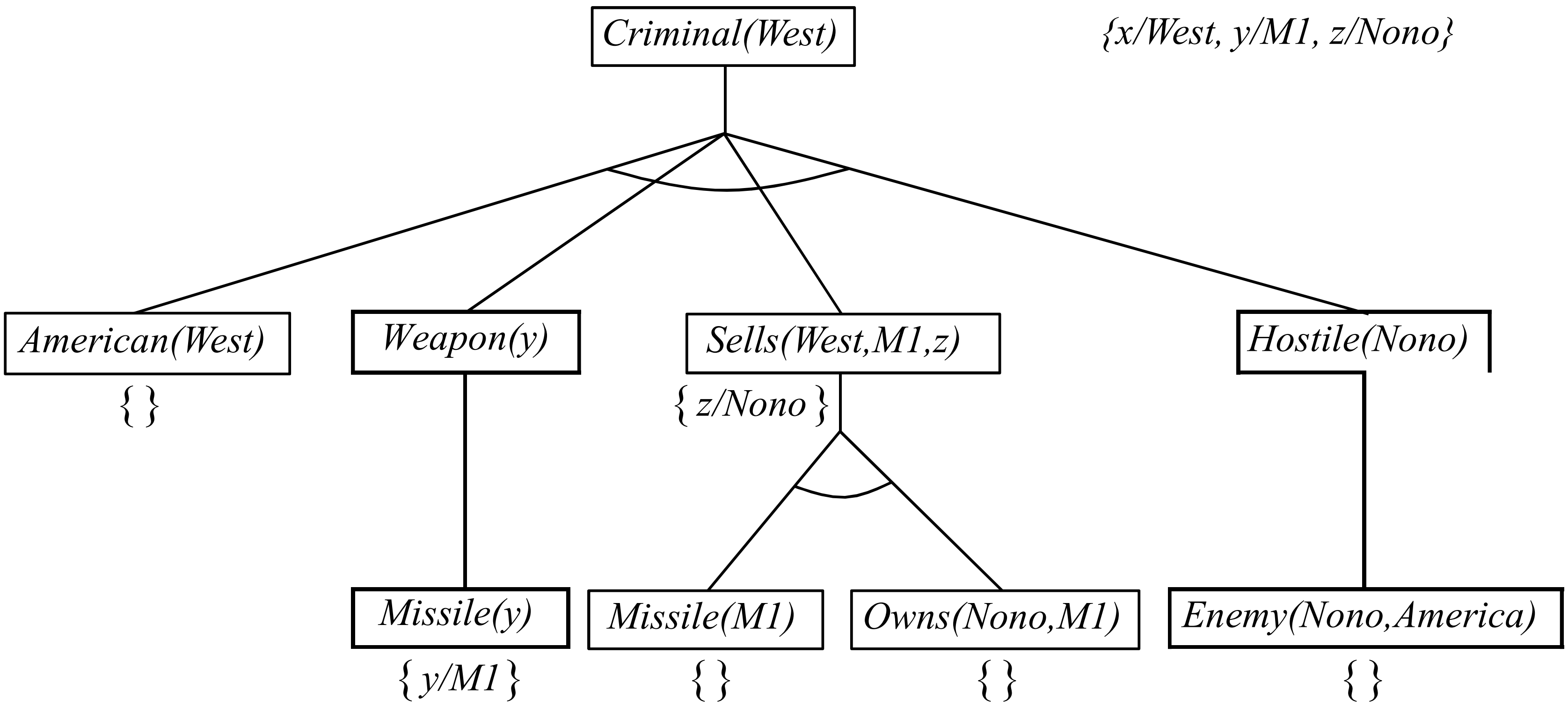
Backward chaining example



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 (without improvements!) for logic programming

Logic programming

Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

Should be easier to debug $Capital(NewYork, US)$ than $x := x + 2$!

Prolog systems

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

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

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

Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
Closed-world assumption ("negation as failure")

e.g., given `alive(X) :- not dead(X).`
`alive(joe)` succeeds if `dead(joe)` fails

Prolog examples

Depth-first search from a start state X:

```
dfs(X) :- goal(X).
dfs(X) :- successor(X,S),dfs(S).
```

No need to loop over S: `successor` succeeds for each

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: brief summary

Full first-order version:

$$\frac{{}_1 \vee \cdots \vee {}_k, \qquad m_1 \vee \cdots \vee m_n}{({}_1 \vee \cdots \vee {}_{i-1} \vee {}_{i+1} \vee \cdots \vee {}_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\mathrm{Unify}({}_i, \neg m_j) = \theta$.

For example,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

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

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

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

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

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

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

Conversion to CNF contd.

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

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

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

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

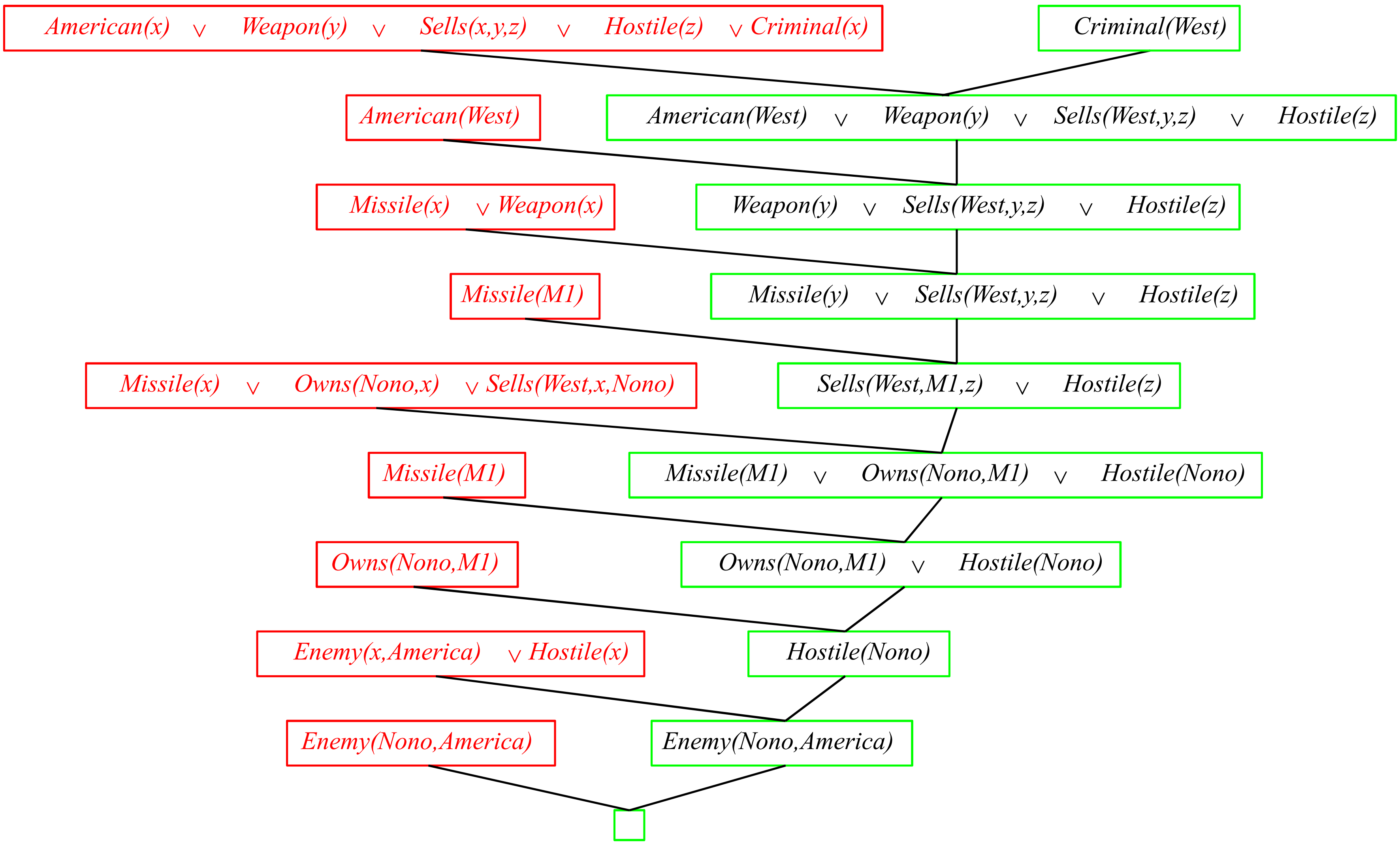
5. Drop universal quantifiers:

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

6. Distribute $\land$ over $\lor$:

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

Resolution proof: definite clauses



Gödel's Incompleteness Theorem

- There are true arithmetic sentences that cannot be proved

- For any set of true sentences of number theory, and in particular any set of basic axioms, there are other true sentences that cannot be proved from those axioms.

- We can never prove all the theorems of mathematics within any given system of axioms.

Resolution strategies

- **Unit preference:** prefers to do resolutions where one of the sentences is a single literal (unit clause)
- **Set of support:** every resolution step involve at least one element of a special set of clauses
- **Input resolution:** every resolution combines one of the KB input sentences with other sentences
- **Subsumption:** eliminates all sentences that are subsumed by KB sentences
- **Learning:** learning from experience (machine learning)

Summary

- **Unification** identify appropriate substitutions for variables eliminates the instantiation step in first-order proofs, making the process more efficient in many cases
- **Forward chaining** is used in deductive databases, where it can be combined with relational database operations. It is also used in production systems
- **Backward chaining** is used in logic programming systems, which employ sophisticated compiler technology to provide very fast inference
- **Prolog**, unlike first-order logic, uses a closed world with the unique names assumption and negation as failure.
- The generalized **resolution** inference rule provides a complete proof system for first order logic, using knowledge bases in conjunctive normal form.