Knowledge-Based Agent in Artificial intelligence:

An intelligent agent needs **knowledge** about the real world for taking decisions and **reasoning** to act efficiently.

Knowledge-based agents are those agents who have the capability of **maintaining an internal state of knowledge, reason over that knowledge, update their knowledge after observations and take actions. These agents can represent the world with some formal representation and act intelligently**.

Knowledge-based agents are composed of two main parts:

Knowledge-base and

Inference system.

A knowledge-based agent must able to do the following:

An agent should be able to represent states, actions, etc.

An agent Should be able to incorporate new percepts

An agent can update the internal representation of the world

An agent can deduce the internal representation of the world

An agent can deduce appropriate actions.

Knowledge base: Knowledge-base is a central component of a knowledge-based agent, it is also known as KB. It is a collection of sentences (here 'sentence' is a technical term and it is not identical to sentence in English). These sentences are expressed in a language which is called a knowledge representation language. The Knowledge-base of KBA stores fact about the world.

Operations Performed by KBA: TELL: This operation tells the knowledge base what it perceives from the environment.

ASK: This operation asks the knowledge base what action it should perform.

Perform: It performs the selected action.

- Inference system
- Inference means deriving new sentences from old. Inference system allows us to add a new sentence to the knowledge base. A sentence is a proposition about the world. Inference system applies logical rules to the KB to deduce new information.
- Inference system generates new facts so that an agent can update the KB. An inference system works mainly in two rules which are given as:
- **Forward chaining**
- **Backward chaining**

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

Various levels of knowledge-based agent:

A knowledge-based agent can be viewed at different levels which are given below:

1. Knowledge level

Knowledge level is the first level of knowledge-based agent, and in this level, we need to specify what the agent knows, and what the agent goals are. With these specifications, we can fix its behavior.

- **2. Logical level:**

  At this level, we understand that how the knowledge representation of knowledge is stored. At this level, sentences are encoded into different logics. At the logical level, an encoding of knowledge into logical sentences occurs.
- **3. Implementation level:**
- This is the physical representation of logic and knowledge. At the implementation level agent perform actions as per logical and knowledge level. At this level, an automated taxi agent actually implement his knowledge and logic so that he can reach to the destination.
- **Approaches to designing a knowledge-based agent**:
- There are mainly two approaches to build a knowledge-based agent:
- **1. Declarative approach:** We can create a knowledge-based agent by initializing with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
- **2. Procedural approach:** In the procedural approach, we directly encode desired behavior as a program code. Which means we just need to write a program that already encodes the desired behavior or agent.

- Propositional logic (PL) is the simplest form of logic where all the statements are made by propositions. A proposition is a declarative statement which is either true or false. It is a technique of knowledge representation in logical and mathematical form.
- a) It is Sunday.
- b) The Sun rises from West (False proposition)
- c) 3+3= 7(False proposition)
- d) 5 is a prime number.

- **Following are some basic facts about propositional logic:**
- Propositional logic is also called Boolean logic as it works on 0 and 1.
- In propositional logic, we use symbolic variables to represent the logic, and we can use any symbol for a representing a proposition, such A, B, C, P, Q, R, etc.
- Propositions can be either true or false, but it cannot be both.
- Propositional logic consists of an object, relations or function, and **logical connectives**.
- These connectives are also called logical operators.

- The propositions and connectives are the basic elements of the propositional logic.
- Connectives can be said as a logical operator which connects two sentences.
- A proposition formula which is always true is called **tautology**, and it is also called a valid sentence.
- A proposition formula which is always false is called **Contradiction**.
- A proposition formula which has both true and false values is called
- Statements which are questions, commands, or opinions are not propositions such as "**Where is Rohini**", "**How are you**", "**What is your name**", are not propositions.

- Syntax of propositional logic:
- The syntax of propositional logic defines the allowable sentences for the knowledge representation. There are two types of Propositions:
- **Atomic Propositions**
- **Compound propositions**
- **Atomic Proposition:** Atomic propositions are the simple propositions. It consists of a single proposition symbol. These are the sentences which must be either true or false.
- a) 2+2 is 4, it is an atomic proposition as it is a **true** fa ct.
- b) "The Sun is cold" is also a proposition as it is a **fals e** fact.

- **Compound proposition:** Compound propositions are constructed by combining simpler or atomic propositions, using parenthesis and logical connectives.
- a) "It is raining today, and street is wet."
- b) "Ankit is a doctor, and his clinic is in Mumbai."

- Logical Connectives:
- Logical connectives are used to connect two simpler propositions or representing a sentence logically. We can create compound propositions with the help of logical connectives. There are mainly five connectives, which are given as follows:
- **Negation:** A sentence such as ¬ P is called negation of P. A literal can be either Positive literal or negative literal.
- **Conjunction:** A sentence which has **Λ** connective such as, **P Λ Q** is called a conjunction.
  Example: Rohan is intelligent and hardworking. It can be written as,
  P= Rohan is intelligent,
  Q= Rohan is hardworking. → PΛ Q.

- **Disjunction:** A sentence which has V connective, such as **P V Q**. is called disjunction, where P and Q are the propositions.
  Example: "Ritika is a doctor or Engineer",
  Here P= Ritika is Doctor. Q= Ritika is Doctor, so we can write it as **P V Q**.
- **Implication:** A sentence such as P → Q, is called an implication. Implications are also known as if-then rules. It can be represented as

  If it is raining, then the street is wet.

  Let P= It is raining, and Q= Street is wet, so it is represented as P → Q
- **Biconditional:** A sentence such as **P⇔ Q is a Biconditional sentence, example If I am breathing, then I am alive**

- P= I am breathing, Q= I am alive, it can be represented as P ⇔ Q.

Following is the summarized table for Propositional Logic Connectives:

Connective symbols	Word	Technical term	Example
∧	AND	Conjunction	A ∧ B
∨	OR	Disjunction	A ∨ B
→	Implies	Implication	A → B
⇔	If and only if	Biconditional	A⇔ B
¬ or ~	Not	Negation	¬ A or ¬ B

- Limitations of Propositional logic:
- We cannot represent relations like ALL, some, or none with propositional logic. Example:
  - **All the girls are intelligent.**
  - **Some apples are sweet.**
- Propositional logic has limited expressive power.
- In propositional logic, we cannot describe statements in terms of their properties or logical relationships.

Inference and proofs

- Inference: In artificial intelligence, we need intelligent computers which can create new logic from old logic or by evidence, **so generating the conclusions from evidence and facts is termed as Inference**.
- Inference rules:
- Inference rules are the templates for generating valid arguments. Inference rules are applied to derive proofs in artificial intelligence, and the proof is a sequence of the conclusion that leads to the desired goal.
- In inference rules, the implication among all the connectives plays an important role. Following are some terminologies related to inference rules:

- **Implication:** It is one of the logical connectives which can be represented as $P \rightarrow Q$. It is a Boolean expression.
- **Converse:** The converse of implication, which means the right-hand side proposition goes to the left-hand side and vice-versa. It can be written as $Q \rightarrow P$.
- **Contrapositive:** The negation of converse is termed as contrapositive, and it can be represented as $\neg Q \rightarrow \neg P$.
- **Inverse:** The negation of implication is called inverse. It can be represented as $\neg P \rightarrow \neg Q$.
- From the above term some of the compound statements are equivalent to each other, which we can prove using truth table:

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$\neg Q \rightarrow \neg P$	$\neg P \rightarrow \neg Q$.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Types of Inference rules:

1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\dfrac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q

Statement-2: "I am sleepy" ==> P

Conclusion: "I go to bed." ==> Q.

Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	$P \rightarrow Q$
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

s.

- Modus Tollens:
- The Modus Tollens rule state that if P→ Q is true and **¬ Q is true, then ¬ P** will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q, \ \sim Q}{\sim P}$

- **Statement-1:** "If I am sleepy then I go to bed" ==> P→ Q
  Statement-2: "I do not go to the bed."==> ~Q
  Statement-3: Which infers that "**I am not sleepy**" => ~P

P	Q	~P	~Q	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

- 3. Hypothetical Syllogism:
- The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:
- **Example:**
- **Statement-1:** If you have my home key then you can unlock my home. **P→Q**
  Statement-2: If you can unlock my home then you can take my money. **Q→R**
  Conclusion: If you have my home key then you can take my money. **P→R**
- **<u>Proof by truth table:</u>**

<u>Proof by truth table:</u>

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

- 4. Disjunctive Syllogism:
- The Disjunctive syllogism rule state that if PVQ is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

- **Example:**
- **Statement-1:** Today is Sunday or Monday. ==>PVQ
  Statement-2: Today is not Sunday. ==> ¬P
  Conclusion: Today is Monday. ==> Q
- 5. Addition:The Addition rule is one the common inference rule, and it states that If P is true, then PVQ will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

- **Example:Statement-1:** I have a vanilla ice-cream. ==> P
  Statement-2: I have Chocolate ice-cream.
  Conclusion: I have vanilla or chocolate ice-cream. ==> (PVQ)

- 6. Simplification:
- The simplification rule state that if **P∧ Q** is true, then **Q or P** will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

7. Resolution:

Notation of Resolution $\frac{P \vee Q, \ \neg P \wedge R}{Q \vee R}$

The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. **It can be represented as**

What is FOPL in artificial intelligence?

The first order predicate logic (FOPL) is backbone of AI, as well a method of formal representation of Natural Language (NL) text. The Prolog language for AI programming has its foundations in FOPL

Why do we need FOPL?

According to a new evidence dossier published by the World Obesity Federation, front-of-pack labelling (FOPL) is an effective policy tool to help support consumers in making informed food choices as part of a package of policies

First-order logic

1. The language of thought

The language of thought hypothesis (LOTH) is a foundational and yet largely empirical thesis about how thoughts (as states) and thinking (as processes) are realized in creatures with sufficiently complex minds. It postulates a language-like system of mental representations as the vehicles of thought and thinking.

The modem view of natural language is that it serves a as a medium for communication rather than pure representation. When a speaker points and says

Natural languages also suffer from ambiguity

The famous Sapir–Whorf hypothesis claims that our understanding of the world is strongly influenced by the language we speak

2. Combining the best of formal and natural languages

propositional logic assumes world contains facts,
first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Klaus Iohannis, colors, base- ball 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

. . .

- "One plus two equals three." Objects: one, two, three, one plus two; Relation: equals; Function: plus. ("One plus two" is a name for the object that is obtained by applying the function 'plus" to the objects "one" and "two." "Three" is another name for this object.)

- "Squares neighboring the wumpus are smelly." Objects: wumpus, squares; Property: smelly; Relation: neighboring.

- "Evil King John ruled England in 1200." Objects: John, England, 1200; Relation: ruled; Properties: evil, king.

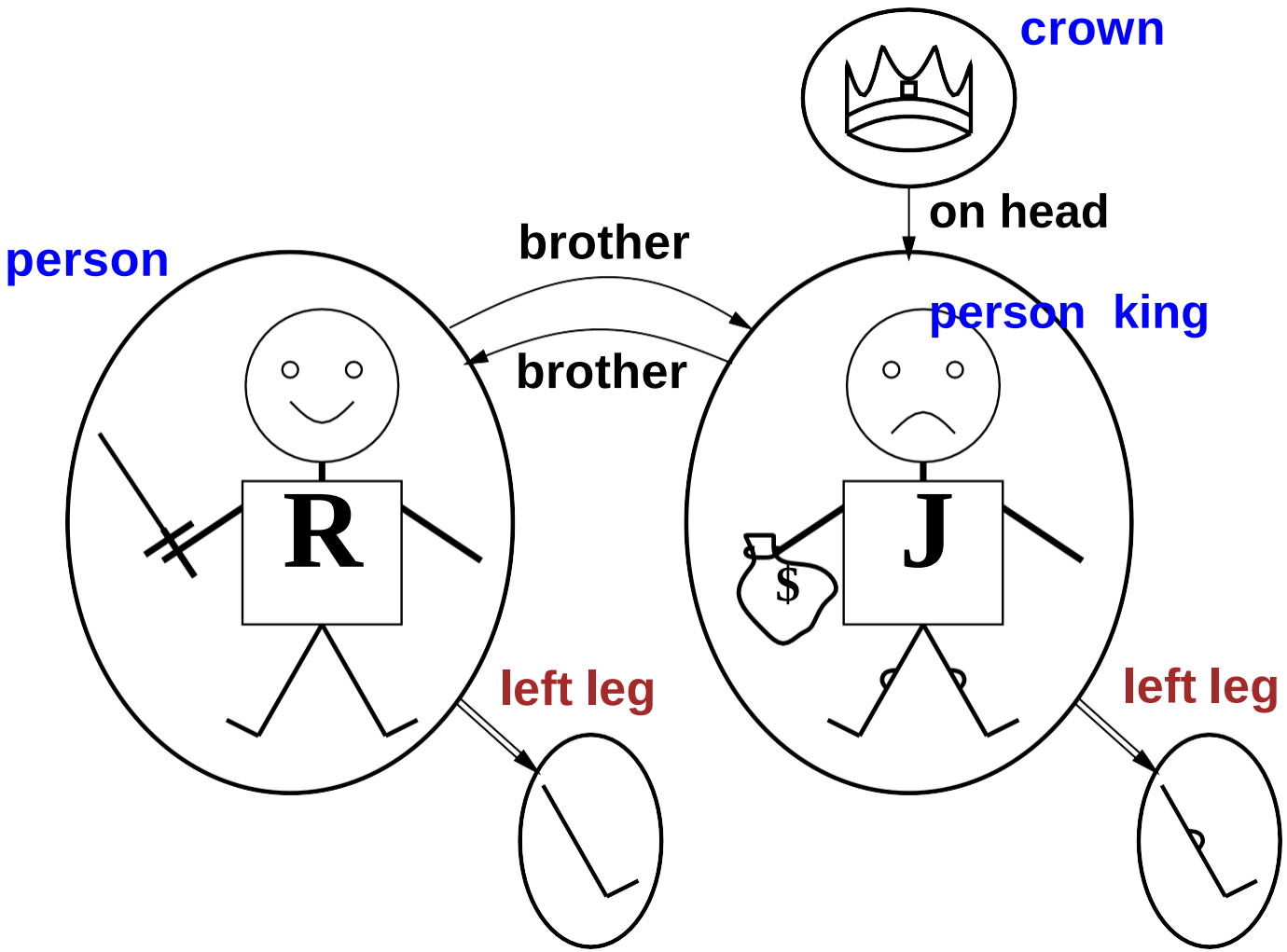
Logics in general

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

Ontological commitment - what is asumed about reality

Epistemological commitment - possbile states of knowledge

Models for FOL: Example



2. Symbols and interpretations

Symbols are things we use to represent other things. Symbols play a vital role in the human thought and reasoning process. If I tell you that I saw a cat up in a tree, your mind will quickly conjure an image.

The basic syntactic elements of first-order logic are the symbols that stand for objects relations, and functions.

The symbols, therefore, come in three kinds: constant symbols which stand for objects; predicate symbols, which stand for relations; and function symbols, which stand for functions.

We adopt the convention that these symbols will begin with uppercase letters

$$
\begin{array}{rcl}
Sentence & \rightarrow & AtomicSentence \mid ComplexSentence \\
AtomicSentence & \rightarrow & Predicate \mid Predicate(Term, \ldots) \mid Term = Term \\
ComplexSentence & \rightarrow & (\ Sentence\ ) \mid [\ Sentence\ ] \\
 & \mid & \neg\ Sentence \\
 & \mid & Sentence \wedge Sentence \\
 & \mid & Sentence \vee Sentence \\
 & \mid & Sentence \Rightarrow Sentence \\
 & \mid & Sentence \Leftrightarrow Sentence \\
 & \mid & Quantifier\ Variable, \ldots\ Sentence \\
 & & \\
Term & \rightarrow & Function(Term, \ldots) \\
 & \mid & Constant \\
 & \mid & Variable \\
 & & \\
Quantifier & \rightarrow & \forall \mid \exists \\
Constant & \rightarrow & A \mid X_1 \mid John \mid \cdots \\
Variable & \rightarrow & a \mid x \mid s \mid \cdots \\
Predicate & \rightarrow & True \mid False \mid After \mid Loves \mid Raining \mid \cdots \\
Function & \rightarrow & Mother \mid LeftLeg \mid \cdots \\
\text{OPERATOR PRECEDENCE} & : & \neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow
\end{array}
$$

The syntax of first-order logic with equality

3. Terms

•A term is a logical expression that refers to an object Constant symbols are therefore terms, but it is not always convenient to have a distinct symbol to name every object.

• For example, in English we might use the expression "King John's left leg" rather than giving a name to his leg. This is what function symbols are for: instead of using a constant symbol, we use Left Leg(John).

• In the general case, a complex term is formed by a function symbol followed by a parenthesized list of terms as arguments to the function symbol.

•It is important to remember that a complex term is just a complicated kind of name. It is not a "subroutine call" that "returns a value

4.Atomic sentences

- we have both terms for referring to objects and predicate symbols for referring to relations, we can put them together to make atomic sentences that state facts.

- An atomic sentence (or atom for short) is formed from a predicate symbol optionally followed by a parenthesized list of terms, such as Brother (Richard „John). This states, under the intended interpretation given earlier, that Richard the Lion heart is the brother of King John.

- Atomic sentences can have complex terms as arguments. Thus, Married(Father (Richard), Mother(John)) states that Richard the Lion heart's father is married to King John's mother (again, under a suitable interpretation).

- An atomic sentence is true in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments

5. Complex sentences

We can use logical connectives to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the model of under our intended interpretation:

–,Brother(LeftLeg(Richard). John)

Brother(Richard.John) A Brother (John, Richard)

King (Richard) V King(John)

–.King(Richard)= King(John)

6. Quantifiers

Once we have a logic that allows objects, it is only natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. Quantifiers let us do this First-order logic contains two standard quantifiers, called universal and existential.

Universal quantification (V)

Rules such as "Squares neighboring the wumpus are smelly" and "All kings arc persons" are the bread and butter of first-order logic.

The second rule, "All kings are persons," is written in first-order logic as Vx King(x) = Person(x)

Existential quantification (∃)

Universal quantification makes statements about every object. Similarly, we can make a statement about some object in the universe without naming it, by using an existential quantifier. To say, for example, that King John has a crown on his head, we write 3 a:

∃x Crown(x) A OnHead(x, John) .

∃x is pronounced "There exists an x such that ..." or "For some x ..".

Nested quantifiers:

Connections between V and

The two quantifiers are actually intimately connected with each other, through negation. Asserting that everyone dislikes parsnips is the same as asserting there does not exist someone who likes them, and vice versa

7. Equality First-order logic includes one more way to make atomic sentences, other than using a predicate and terms as described earlier. We can use the equality symbol to signify that two terms refer to the same object.

For example, Father (John) = Henry

says that the object referred to by Father(John) and the object referred to by Henry are the same.

8. An alternative semantics?

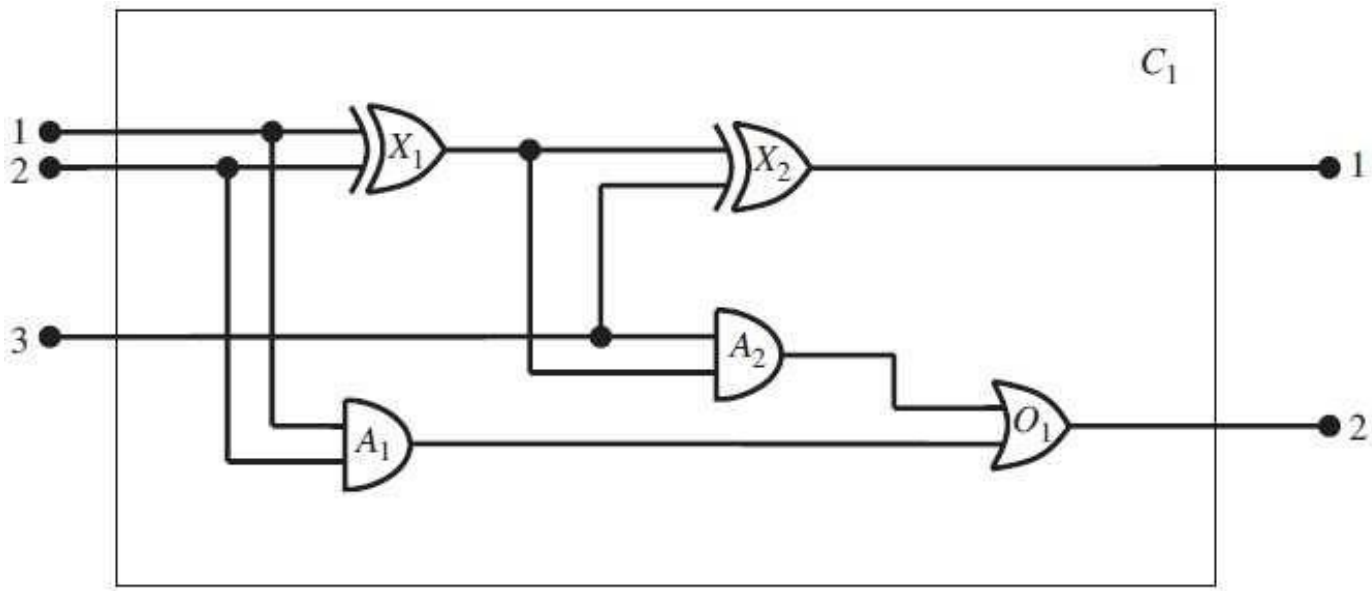
Knowledge engineering in first-order logic

Knowledge engineer: investigates a particular domain, learns what concepts are important, creates a formal representation of the objects and relations in the domain

Knowledge engineering steps

1. Identify the task. - list compentency questions that the KB will respond
2. Assemble the relevant knowledge. - knowledge acuisition
3. Decide on a vocabulary of predicates, functions, and constants - translate the important domain-level concepts into logic-level names - ontology
4. Encode general knowledge about the domain - axioms for all vocabulary terms
5. Encode a description of the specific problem instance - atomic sentences about instances of concepts in the ontology
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

The electronic circuit domain



Step 1 - Identify the task

♦Functionality (e.g., If all the inputs are high, what is the output of gate A2?)

♦Structure (e.g., What are all the gates connected to the first input terminal? Does the circuit contain feedback loops?)

Step 2 - Assemble the relevant knowledge

♦Relevant: types of gates, number of inputs/outputs, signals

♦Irrelevant: wires, size, shape, cost

Step 3 - Decide on the vocabulary

♦Objects: X_1, $Gate(X_1)$ ♦ constants: *AND, OR, XOR, NOT* ♦ functions: $Type(X_1) = XOR$, $In(1, X_1)$, $Arity(c, i, o)$, $Signal(t)$ ♦ predicates: $Circuit(C_1)$, $Terminal(x)$, $Connected(Out1(1, X_1), In(1, X_2))$

Step 4 - Encode general knowledge

1. If two terminals are connected, then they have the same signal
2. The signal at every terminal is either 1 or 0
3. Connected is commutative
4. There are four types of gates
5. An AND gate's output is 0 if and only if any of its inputs is 0
6. An OR gates output is 1 if and only if any of its inputs is 1

7. An XOR gates output is 1 if and only if its inputs are different
8. A NOT gate's output is different from its input
9. The gates (except for NOT) have two inputs and one output.
10. A circuit has terminals, up to its input and output arity, and nothing beyond its arity
11. Gates, terminals, signals, gate types, and Nothing are all distinct
12. Gates are circuits

1. $\forall\, t_1, t_2 \quad Terminal(t_1) \wedge Terminal(t_2) \wedge Connected(t_1, t_2) \Rightarrow Signal(t_1) = Signal(t_2)$

2. $\forall\, t \quad Terminal(t) \quad \Rightarrow \quad Signal(t) = 1 \vee Signal(t) = 0$

3. $\forall\, t_1, t_2 \quad Connected(t_1, t_2) \Leftrightarrow \quad Connected(t_2, t_1)$

4. $\forall\, g \quad Gate(g) \wedge k = Type(g) \Rightarrow k = AND \vee k = OR \vee k = XOR \vee k = NOT$

5. $\forall\, g \quad Gate(g) \wedge Type(g) = AND \Rightarrow Signal(Out(1, g)) = 0 \quad \Leftrightarrow \quad \exists\, n \quad Signal(In(n, g)) = 0$

6. $\forall g \;\; Gate(g) \wedge Type(g) = OR \Rightarrow Signal(Out(1, g)) = 1 \quad \Leftrightarrow \quad \exists n Signal(In(n, g)) = 1$

7. $\forall g \;\; Gate(g) \wedge Type(g) = XOR \Rightarrow Signal(Out(1, g)) = 1 \quad \Leftrightarrow \quad Signal(In(1, g)) \; /= \; Signal(In(2, g))$

8. $\forall g Gate(g) \wedge (Type(g) = NOT) \Rightarrow Signal(Out(1, g)) \; /= \; Signal(In(1, g))$

9. $\forall g \; Gate(g) \wedge Type(g) = NOT \quad \Rightarrow \quad Arity(g, 1, 1)$

$\forall g \quad Gate(g) \wedge k \quad = \quad Type(g) \wedge (k \quad = \quad AND \vee k \quad = \quad OR \vee k \quad =$

$XOR) \quad \Rightarrow \quad Arity(g, 2, 1)$

10. $\forall c, i, j \quad Circuit(c) \wedge Arity(c, i, j) \Rightarrow \forall n \quad (n \leq iTerminal(In(c, n))) \wedge \; (n > i \Rightarrow In(c, n) = Nothing) \wedge \forall n \; (n \leq jTerminal(Out(c, n))) \wedge \; (n > j \quad \Rightarrow \quad Out(c, n) = Nothing)$

11. $\forall g, t \quad Gate(g) \wedge Terminal(t) \quad \Rightarrow \quad g \; /= \; t \; /= \; 1 \; =/ \; 0 \; =/ \; OR \; /= \; AND \; /= \; XOR \; =/ \; NOT \; /= \; Nothing$

12. $\forall g \;\; Gate(g) \quad \Rightarrow \quad Circuit(g)$

Step 5 - Encode the specific problem instance

$Circuit(C_1) \wedge Arity(C_1, 3, 2)$
$Gate(X_1) \wedge Type(X_1) = XOR$
$Gate(X_2) \wedge Type(X_2) = XOR$
$Gate(A_1) \wedge Type(A_1) = AND$
$Gate(A_2) \wedge Type(A_2) = AND$
$Gate(O_1) \wedge Type(O_1) = OR$

$Connected(Out(1, X_1), In(1, X_2))$
$Connected(Out(1, X_1), In(2, A_2))$
$Connected(Out(1, A_2), In(1, O_1))$
$Connected(Out(1, A_1), In(2, O_1))$
$Connected(Out(1, X_2), Out(1, C_1))$
$Connected(Out(1, O_1), Out(2, C_1))$

$Connected(In(1, C_1), In(1, X_1))$
$Connected(In(1, C_1), In(1, A_1))$
$Connected(In(2, C_1), In(2, X_1))$
$Connected(In(2, C_1), In(2, A_1))$
$Connected(In(3, C_1), In(2, X_2))$
$Connected(In(3, C_1), In(1, A_2))$

Step 6 - Pose queries

What combinations of inputs would cause the first output of C1 (the sum bit) to be 0 and the second output of C1 (the carry bit) to be 1?

$\exists\ i_1, i_2, i_3 \quad Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge Signal(In(3, C_1)) = i_3 \wedge Signal(Out(1, C_1)) = 0 \wedge Signal(Out(2, C_1)) = 1$

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

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

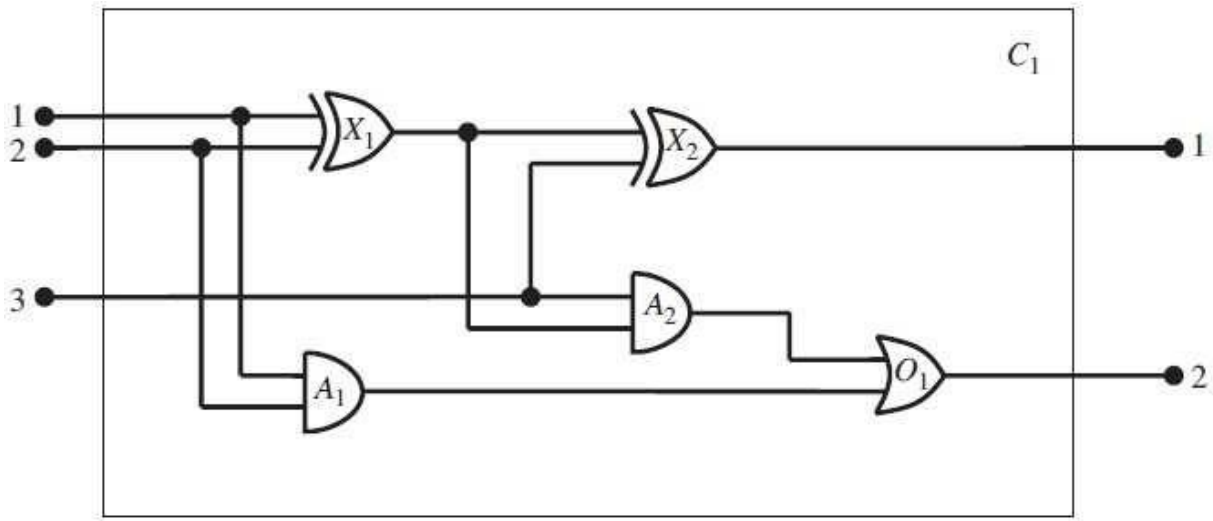
$\exists\ i_1, i_2, i_3, o_1, o_2\ Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge Signal(In(3, C_1)) = i_3 \wedge Signal(Out(1, C_1)) = o_1 \wedge Signal(Out(2, C_1)) = o_2$

Step 7 - Debug the knowledge base

Assume we forget to assert that $1 \neq 0$

$\exists\, i_1, i_2, o \qquad Signal(In(1, C_1)) = i_1 \wedge Signal(In(2, C_1)) = i_2 \wedge Signal(Out(1, X_1))$

No outputs are known at X_1 for the input cases 10 and 01.



$Signal(Out(1, X_1)) = 1 \equiv Signal(In(1, X_1)) \neq Signal(In(2, X_1))$

$Signal(Out(1, X_1)) = 1 \equiv 1 \neq 0$

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

INFERENCE IN FIRST-ORDER LOGIC

PROPOSITIONAL VS. FIRST-ORDER INFERENCE

1. Inference rules for quantifiers

Suppose our knowledge base contains the standard folkloric axiom stating that all greedy kings are evil:

∀ *x* King(x) A Greedy(x) Evil(x).

Then it seems quite permissible to infer any of the following sentences:
King (John) A Greedy(John) = Evil(John)
King(Richard) A Greedy(Richard) Evil(Rirchard)
King(Father(John)) A Greedy(Father(John)) Evil(Father(John)).

2. Reduction to propositional inference

Once we have rules for inferring nonquantified sentences from quantified sentences, it becomes possible to reduce first-order inference to propositional inference.

Suppose the KB contains just the following:

$\forall\ x \quad King(x) \wedge Greedy(x) \quad \Rightarrow \quad Evil(x)$ $King(John)$
$Greedy(John)$
$Brother(Richard, John)$

Instantiating the universal sentence in all possible ways, we have

$King(John) \wedge Greedy(John) \quad \Rightarrow \quad Evil(John)$
$King(Richard) \wedge Greedy(Richard) \quad \Rightarrow \quad Evil(Richard)$ $King(John)$
$Greedy(John)$
$Brother(Richard, John)$

The new KB is propositionalized: proposition symbols are

$King(John)$, $Greedy(John)$, $Evil(John)$, $King(Richard)$ etc.

UNIFICATION AND LIFTING

What is unification and lifting in artificial intelligence?

Unification is a key component of all first-order inference algorithms. It returns fail if the expressions do not match with each other

1.A first-order inference rule

- This rule states that if there is some element in the universe of discourse which has a property P, then we can infer that there exists something in the universe which has the property P.

- inference rules are used to generate proofs, and a proof is a series of conclusions that leads to the intended outcome. The implication among all the connectives is vital in inference rules.

2. Unification

What is unification in AI?

Unification is a process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process. It takes two literals as input and makes them identical using substitution.

Or

finding substitutions that make different logical expressions look identical. This process is called unification and is a key component of all first-order inference algorithms.

The UNIFY algorithm takes two sentences and returns a unifier for them if one exists.

UNIFY (p , q)= 0 where SUBST(0,p)= SUBST(0, q) .

Suppose we have a query AskVars(Knows (John , x)): whom does John know? Answers to this query can be found by finding all sentences in the knowledge base that unify with Knows(John, x). Here are the results of unification with four different sentences that might be in the knowledge base:

UNIFY (Knows (John x).Knows (John, Jane)) = Ix I Jane}
UNIFY (Knows (John , x).Knows (y,Bill)) = {x I Bill, y I John}
UNIFY (Knows(John , x),Knows (y, Mother (y)))= {y,John x I mother(John)}
UNIFY (Knows(John, x).Knows (x , Elizabeth))= fail.

The last unification fails because x cannot take on the values John. and Elizabeth at the same time. Now, remember that Knows(x, Elizabeth) means "Everyone knows Elizabeth," so we should be able to infer that John knows Elizabeth.

3. Storage and retrieval

Underlying the TELL and ASK functions used to inform and interrogate a knowledge base are the more primitive STORE and FETCH functions.

STORE(s)stores a sentence a into the knowledge base and FETCH(q) returns all unifiers such that the query q unifies with some sentence in the knowledge base

FORWARD CHAINING

What is forward chain in AI?

Forward chaining in artificial intelligence is a method in which inference rules are applied to existing data to extract additional data until an end goal is achieved.

1. First-order definite clauses

- They are disjunctions of literals of which exactly one is positive.

- A definite clause either is atomic or is an implication whose antecedent is a conjunction of positive literals and whose consequent is a single positive literal.

2. A simple forward-chaining algorithm.

The first forward-chaining algorithm we consider is a simple one. Starting from the known facts, it triggers all the rules whose premises are satisfied, adding their conclusions to the known facts. The process repeats until the query is answered (assuming that just one answer is required) or no new facts are added.

```
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

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

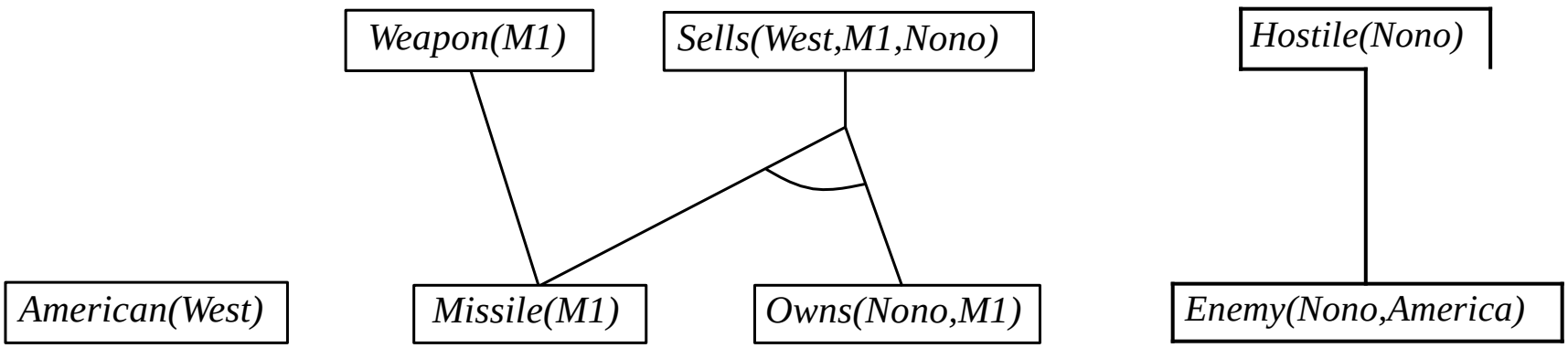
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

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

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

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

Forward chaining proof



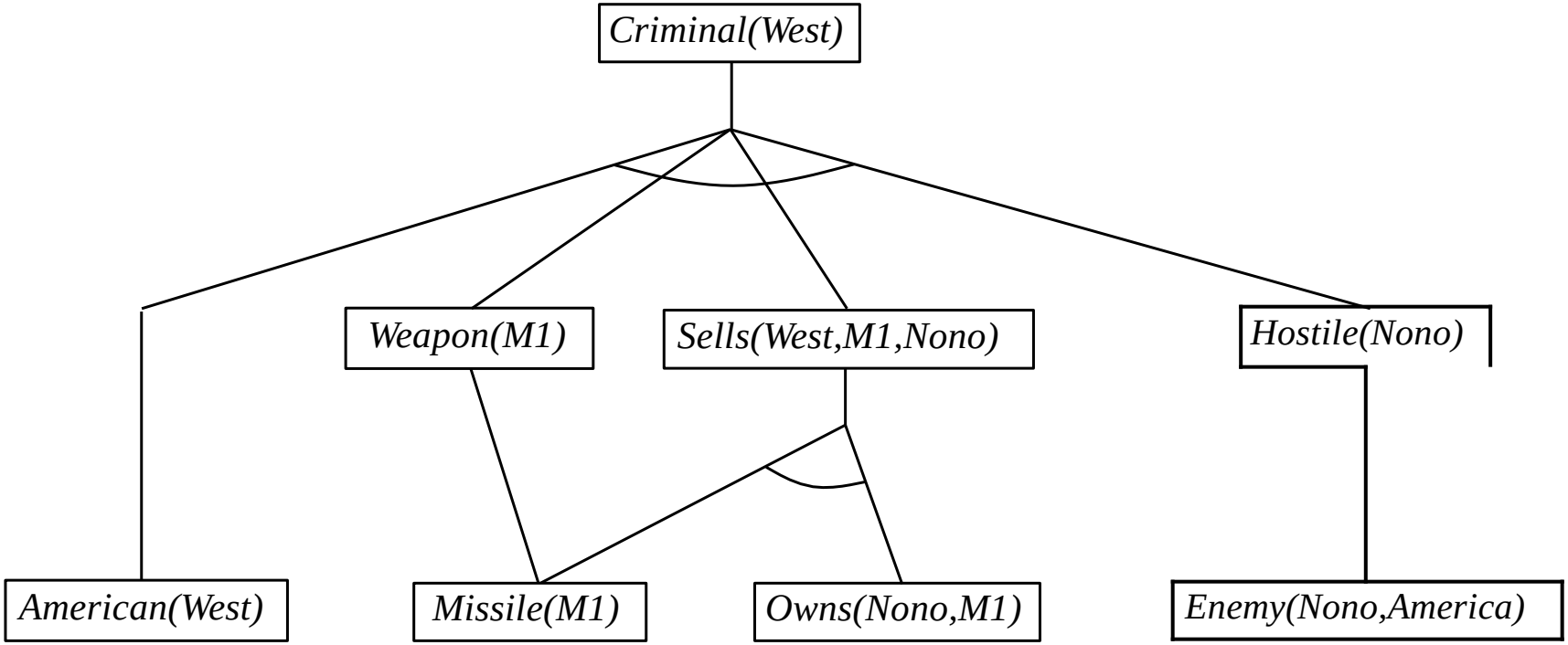
$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

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

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

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

Forward chaining proof



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

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

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

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

3. Efficient forward chaining

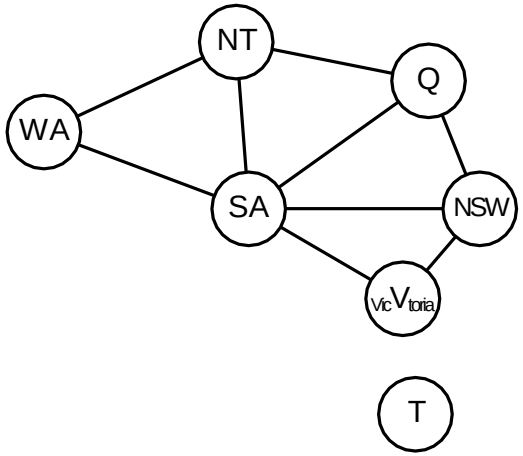
The forward-chaining algorithm is designed for ease of understanding rather than for efficiency of operation. There are three possible sources of inefficiency.

- First, the "inner loop" of the algorithm involves finding all possible unifiers such that the premise of a rule unifies with a suitable set of facts in the knowledge base. This is often called pattern matching and can be very expensive.

- Second, the algorithm rechecks every rule on every iteration to see whether its premises are satisfied, even if very few additions are made to the knowledge base on each iteration.

- Finally, the algorithm might generate many facts that are irrelevant to the goal. We address each of these issues in turn.

Hard matching example



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

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

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

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

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

BACKWARD CHAINING

These algorithms work backward from the goal, chaining through rules to find known facts that support the proof

1. A 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

$\boxed{Criminal(West)}$

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

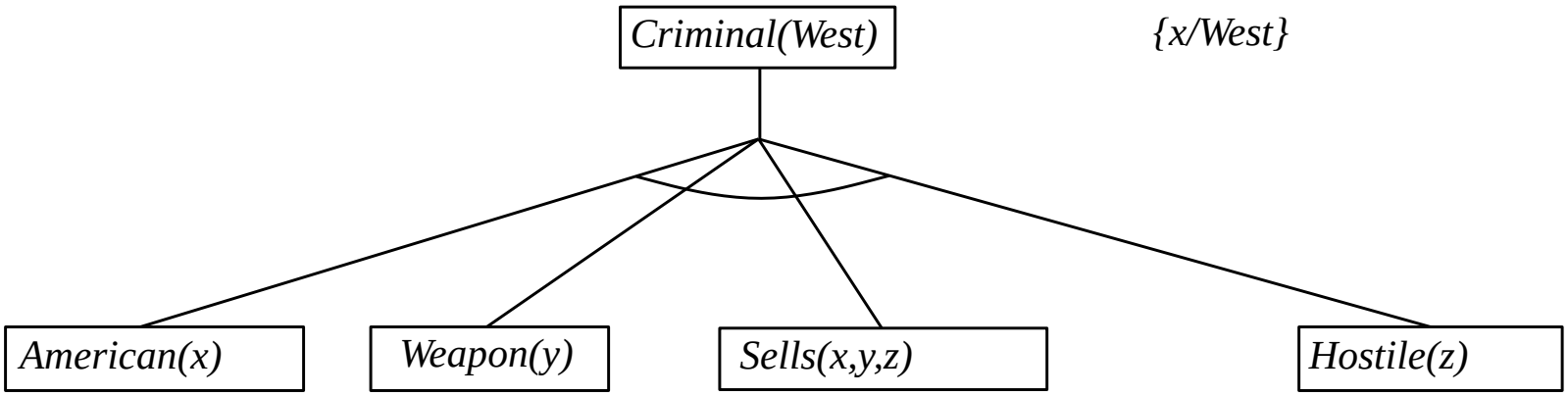
$\forall\ x \quad Missile(x) \wedge Owns(Nono, x) \quad \Rightarrow \quad Sells(West, x, Nono)$

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

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

$American(West),\ Enemy(Nono, America),\ Owns(Nono, M_1),\ Missile(M_1)$

Backward chaining example



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

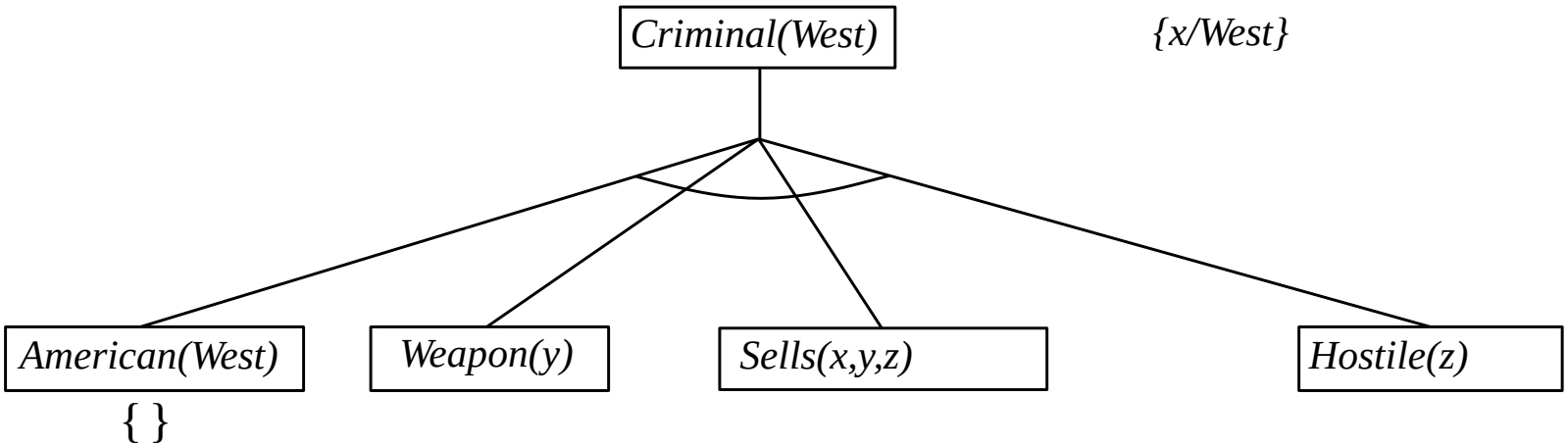
$\forall\ x \quad Missile(x) \wedge Owns(Nono, x) \quad \Rightarrow \quad Sells(West, x, Nono)$

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

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

$American(West),\ Enemy(Nono, America),\ Owns(Nono, M_1),\ Missile(M_1)$

Backward chaining example



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

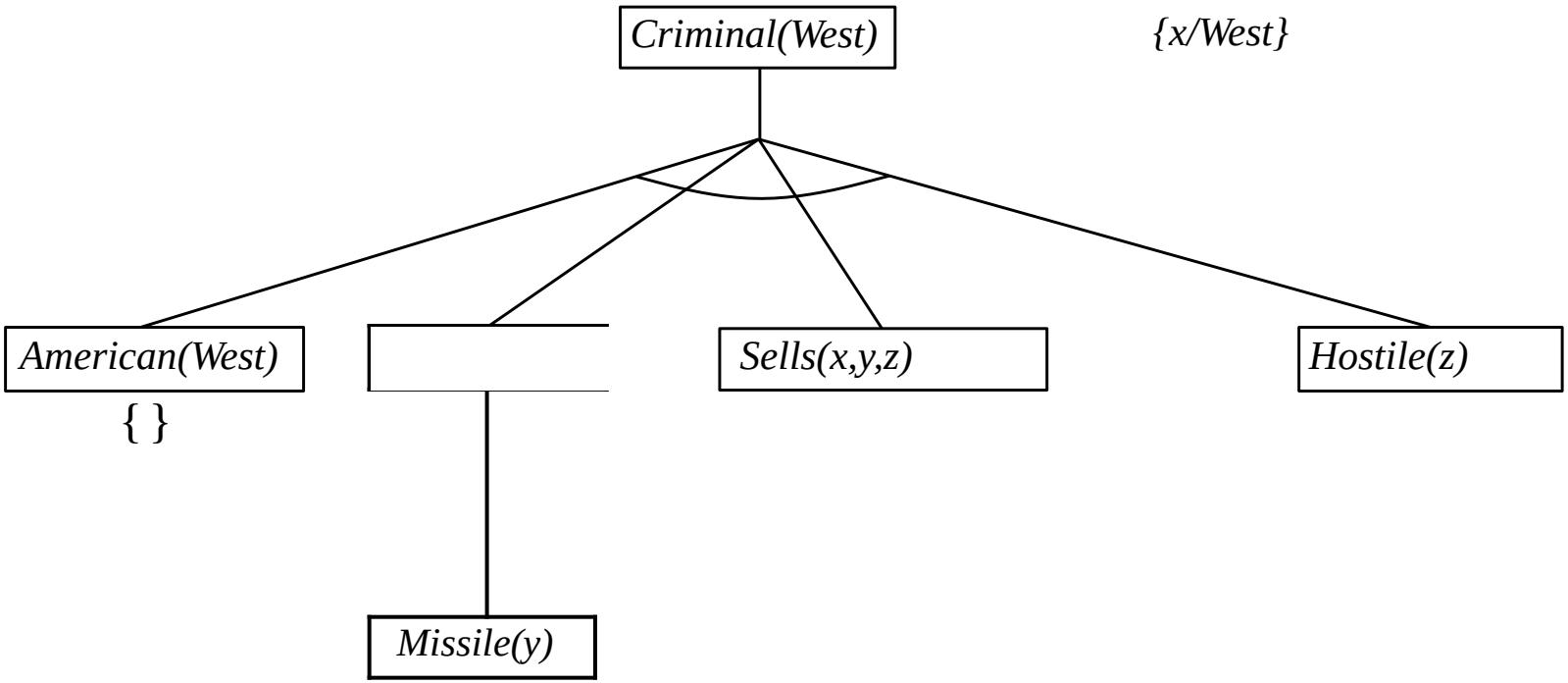
$\forall\ x \quad Missile(x) \wedge Owns(Nono, x) \quad \Rightarrow \quad Sells(West, x, Nono)$

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

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

$American(West),\ Enemy(Nono, America),\ Owns(Nono, M_1),\ Missile(M_1)$

Backward chaining example



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

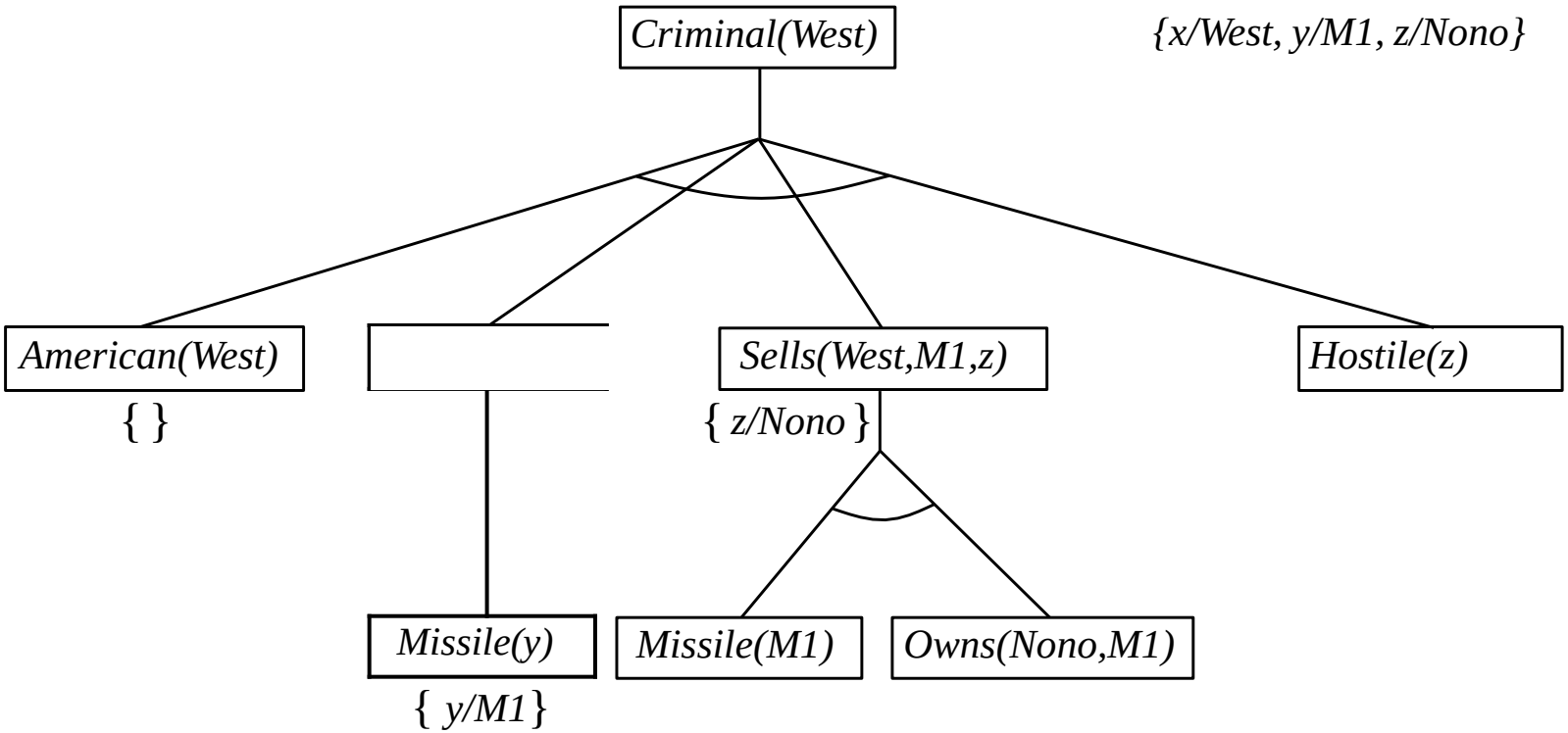
$\forall\ x \quad Missile(x) \wedge Owns(Nono, x) \quad \Rightarrow \quad Sells(West, x, Nono)$

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

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

$American(West), Enemy(Nono, America), Owns(Nono, M_1), Missile(M_1)$

Backward chaining example



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

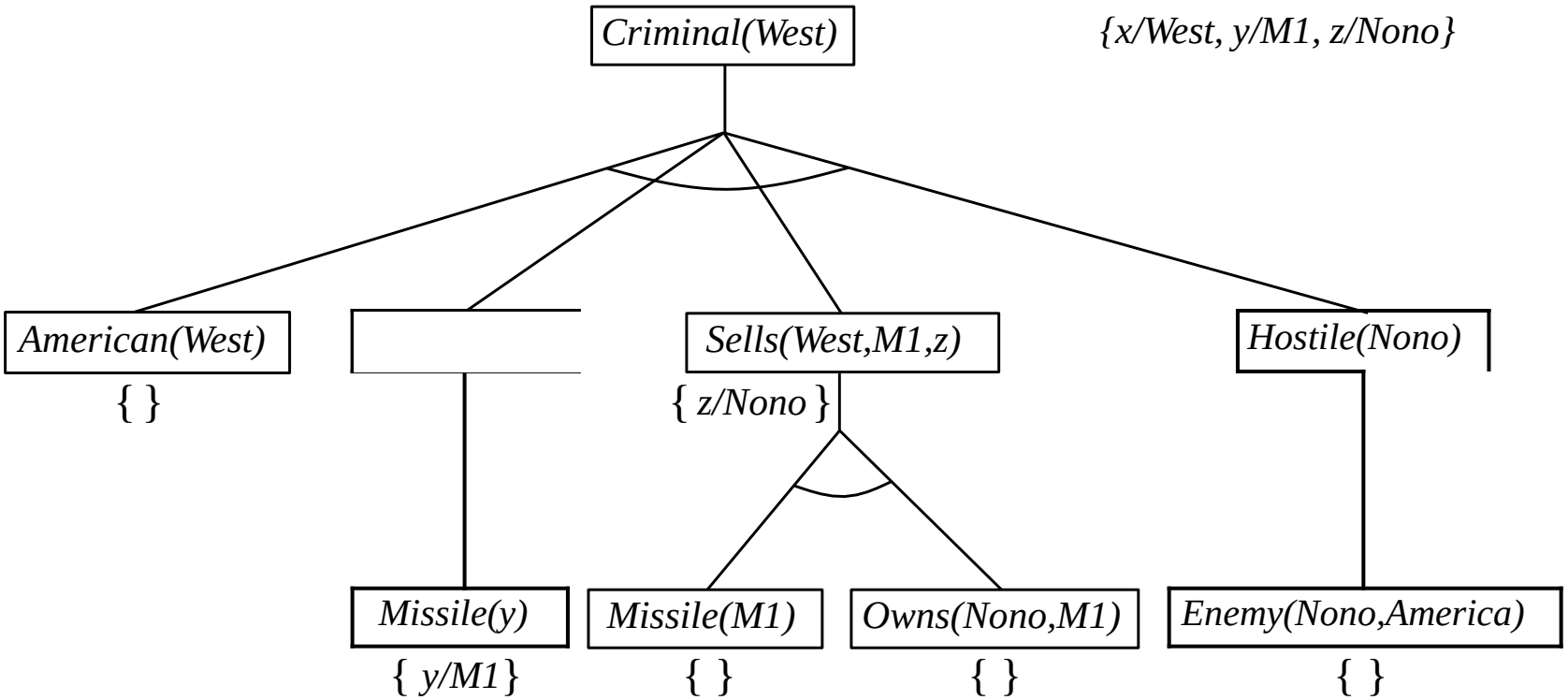
$\forall\ x \quad Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

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

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

$American(West), Enemy(Nono, America), Owns(Nono, M_1), Missile(M_1)$

Backward chaining example



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

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

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

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

$American(West),\ Enemy(Nono, America),\ Owns(Nono, M_1),\ Missile(M_1)$

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

2.Logic programming

Sound bite: computation as inference on logical KBs

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

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

3. Efficient implementation of logic programs

- The execution of a Prolog program can happen in two modes: interpreted and compiled. Interpretation essentially amounts to running the FOL-BC-AsK algorithm, with the program as the knowledge base.

- Prolog interpreters contain a variety of improvements designed to maximize speed. Here we consider only two.

  - First, manage the iteration over possible results generated by each of the sub functions.

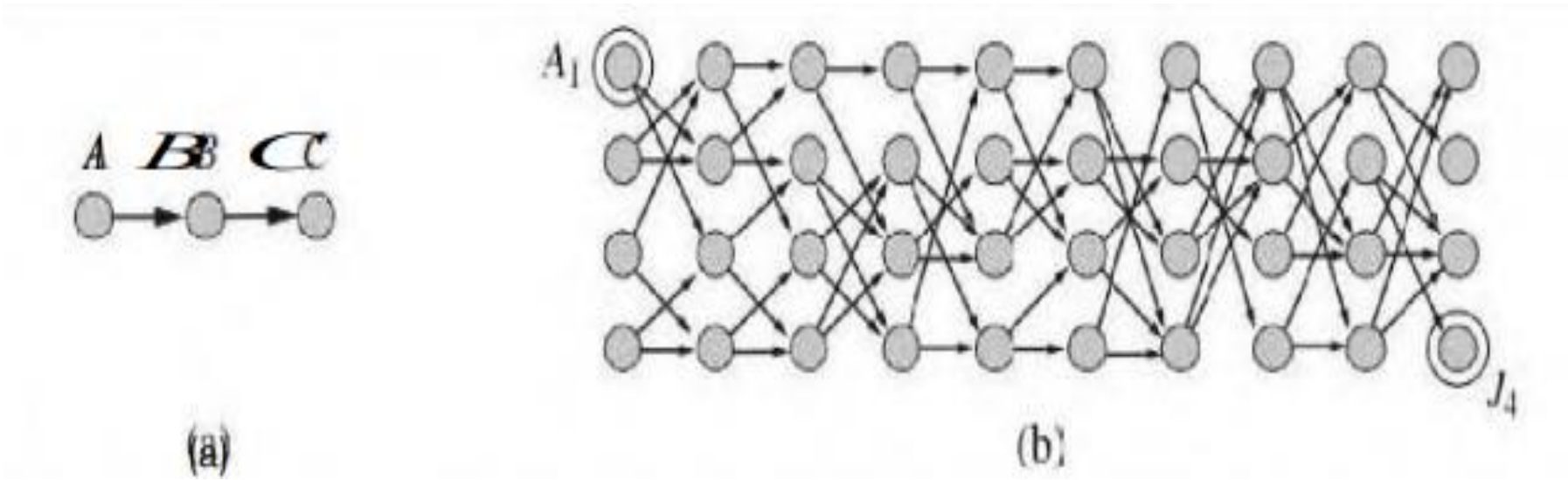
  - Second, FOL-BC-ASK spends a good deal of time generating substitutions.

4. Redundant inference and infinite loops

The mismatch between depth-first search and search trees that include repeated states and infinite paths. Consider the following logic program that decides if a path exists between two points on a directed graph:

path(X,Z) link(X,Z).

path(X,Z) path(X,Y), link(Y,Z).



(a) Finding a path from A to C can lead Prolog into an infinite loop. (b) A graph in which each node is connected in the next layer. finding a path from A t to J4 requires 877inferences.

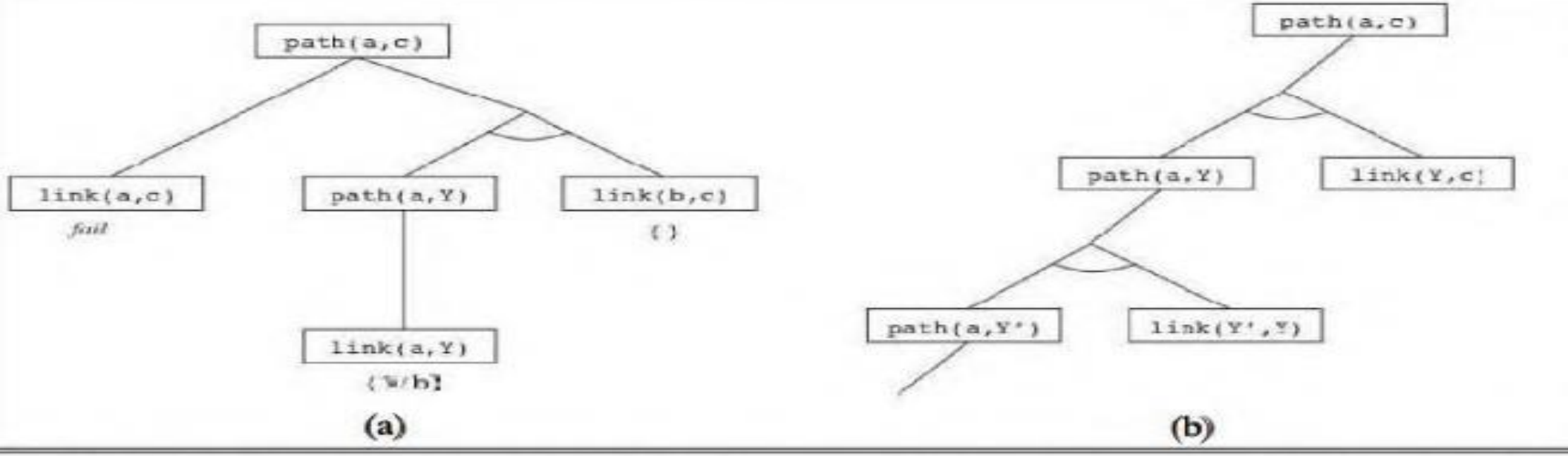


Figure (a) Proof that a path exists from A to C. (b) Infinite proof tree generated when the clauses are in the "wrong" order.

5. Database semantics of Prolog

- Prolog uses database semantics. The unique names assumption says that every Prolog constant and every ground term refers to a distinct object, and the closed world assumption says that the only sentences that are true are those that are entailed by the knowledge base.
- There is no way to assert that a sentence is false in Prolog. This makes Prolog less expressive than first-order logic, but it is part of what makes Prolog more efficient and more concise

6. Constraint logic programming

What is Constraint Logic Programming used for?

Constraint programming is a method of solving highly combinatorial problems given a declarative problem description.

RESOLUTION

This method is basically used for proving the satisfiability of a sentence

1. Conjunctive normal form for first-order logic

   first-order resolution requires that sentences is in conjunctive normal form (CNF)—that is, a conjunction of clauses, where each clause is a disjunction of literals.

2.The resolution inference rule

The resolution inference rule takes two premises in the form of clauses $(A \vee x)$ and $(B \vee \neg x)$ and gives the clause $(A \vee B)$ as a conclusion.

3. Example proofs

Resolution proof: definite clauses

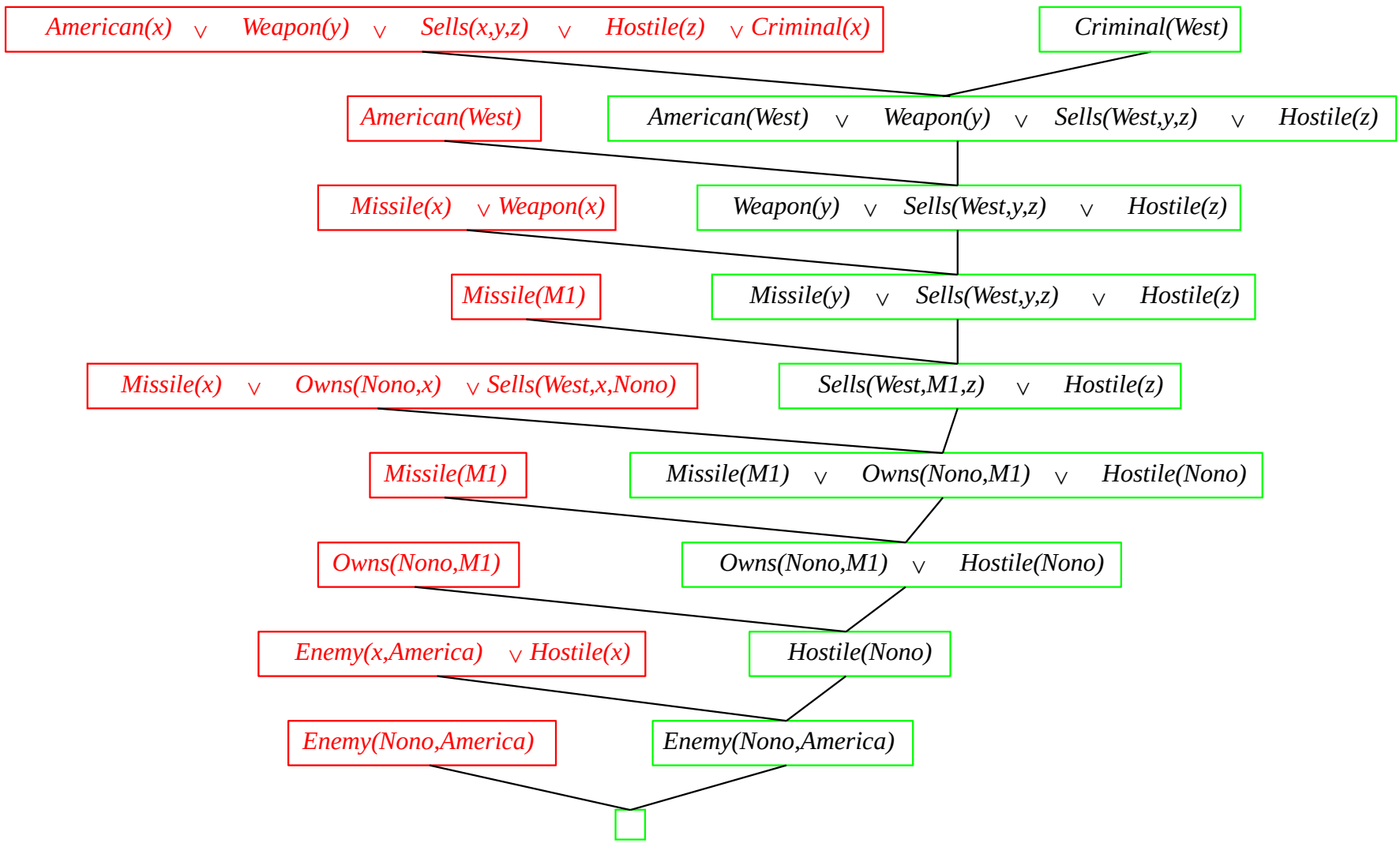
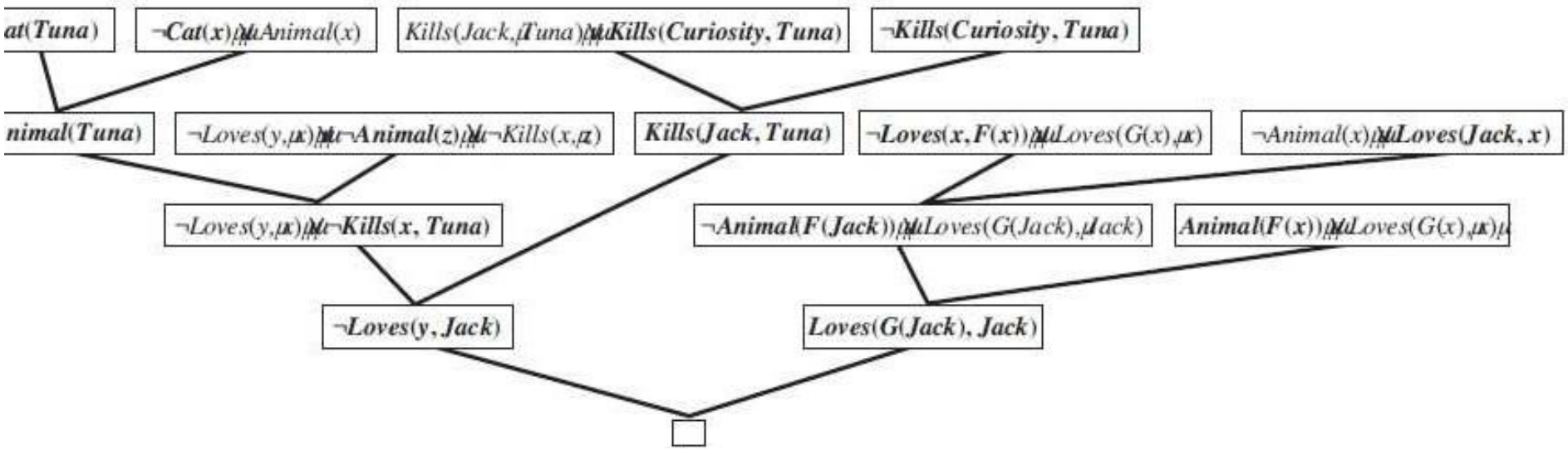


Fig. A resolution proof that West is a criminal. At each step, the literals that unify are in bold.

Proof by resolution



4. Completeness of resolution

- Resolution is refutation-complete, which means that if a set of sentences is unsatisfiable, then resolution will always be able to derive a contradiction.

- Resolution cannot be used to generate all logical consequences of a set of sentences, but it can be used to establish that a given sentence is entailed by the set of sentences.

5. Equality

Distinct approaches can be taken.

We need to say that equality is reflexive. symmetric, and transitive, and we also have to say that we can substitute equals for equals in any predicate or function

6. Resolution strategies

Resolution inference rule will eventually find a proof if one exists. we examine strategies that help find proofs efficiently.

- Unit preference: This strategy prefers to do resolutions where one of the sentences is a single literal (also known as a unit clause).

- Set of support: Preferences that try certain resolutions first are helpful, but in general it is more effective to try to eliminate some potential resolutions altogether.

Input resolution: In this strategy, every resolution combines one of the input sentences with some other sentence

Subsumption: The subsumption method eliminates all sentences that are subsumed by an existing sentence in the KB