

Artificial Intelligence

FIRST-ORDER LOGIC

Nguyễn Ngọc Thảo – Nguyễn Hải Minh
{nnthao, nhminh}@fit.hcmus.edu.vn

Outline

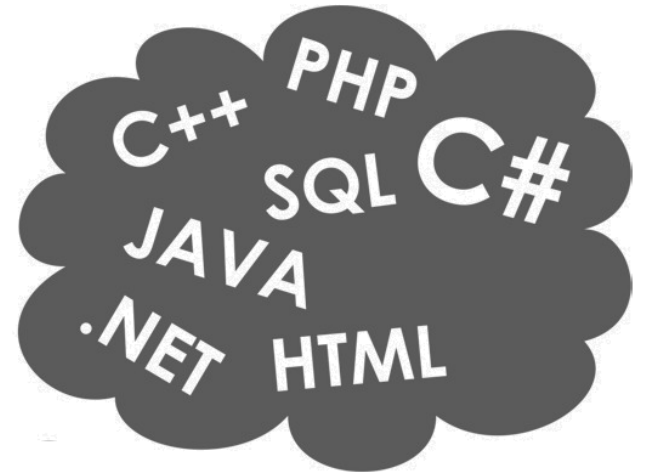
- Representation revisited
- Syntax and semantics of first-order logic (FOL)
- Using first-order logic
- Propositional vs. First-order inference
- Unification and lifting
- Forward chaining
- Backward chaining
- Resolution

Representation revisited



Programming languages

- By far the largest class of formal languages in common use
 - E.g., C++, Java or Lisp, etc.
- **Programs** represent **computational processes** while their **data structures** represent **facts**.
 - E.g., the Wumpus world can be represented by a 4×4 array, “World[2,2] \leftarrow Pit” states that “There is a pit in square [2,2].”



Programming languages

- **Lack of general mechanisms** to derive facts from other facts
 - Update to a data structure is done by a domain-specific procedure.
- **Lack of expressiveness** to handle partial information
 - E.g., to say “There is a pit in [2,2] or [3,1]”, a program stores a single value for each variable and allows the value to be “unknown”, while the propositional logic sentence, $P_{2,2} \vee P_{3,1}$, is more intuitive.

Propositional logic

- Propositional logic is a **declarative language**.
 - Semantics is based on the truth relation between sentences and possible worlds.
- It **handles partial information** using disjunction and negation.
- Propositional logic is **compositional**, which is desirable in representation languages
 - The meaning of a sentence is a function of the meaning of its parts
 - E.g., The meaning of $S_{1,4} \wedge S_{1,2}$ relates the meanings of $S_{1,4}$ and $S_{1,2}$.

Propositional logic

- Meaning in propositional logic is **context-independent**.
 - The natural language, on the other hand, are dependent on context.
- Propositional logic has very **limited expressive power**.
 - E.g., cannot say “Pits cause breezes in adjacent squares”, except by writing one sentence for each square

$$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1}), B_{2,2} \Leftrightarrow (P_{1,2} \vee P_{2,1} \vee P_{3,2} \vee P_{3,1}), \text{ etc.}$$

First-order logic

- **Objects** are referred by nouns and noun phrases.
 - E.g., people, houses, numbers, colors, Bill Gates, games, wars, etc.
- **Relations** can be **unary relations** (properties) or ***n*-ary** relations, representing by verbs and verb phrases
 - Properties: red, round, prime, etc.
 - *n*-ary relations: brother of, bigger than, part of, comes between, etc.
- **Functions** are relations in which there is **only one “value”** for a given “input.”
 - E.g., father of, best friend, one more than, etc.

First-order logic: Some examples

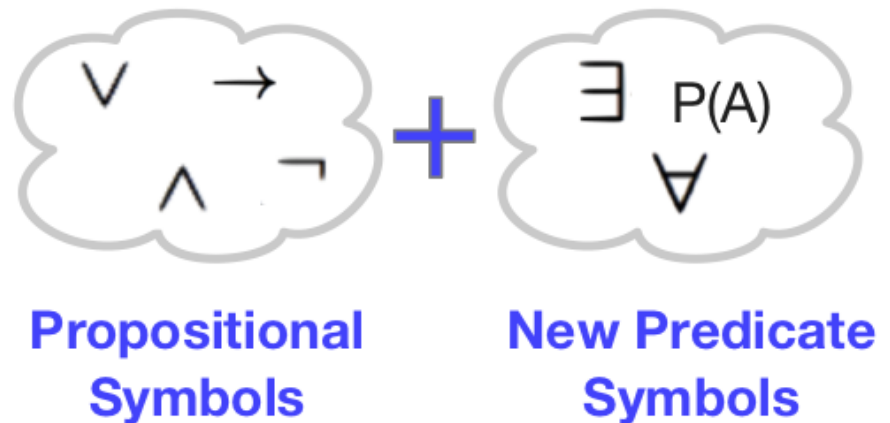
- “One plus two equals three.”
 - Object: one, two, three, one plus two
 - Relation: equal
 - Function: plus
- “Squares neighboring the Wumpus are smelly.”
 - Object: squares, Wumpus
 - Property: smelly
 - Relation: neighboring
- “The intelligent AlphaGo beat the world champion in 2016.”
 - Object: AlphaGo, world champion, 2016
 - Property: intelligent
 - Relation: beat

Types of logics

Language	Ontological Commitment (What exists in the world)	Epistemological Commitment (What an agent believes about facts)
Propositional logic	Facts	True/false/unknown
First-order logic	Facts, objects, relations	True/false/unknown
Temporal logic	Facts, objects, relations, time	True/false/unknown
Probability logic	Facts	Degree of belief $\in [0,1]$
Fuzzy logic	Facts with degree of truth $\in [0,1]$	Known interval value

Syntax and Semantics of FOL

- *Models for First-Order Logic*
- *Symbols and Interpretations*
- *Terms*
- *Atomic Sentences*
- *Complex Sentences*
- *Quantifiers*
- *Equality*



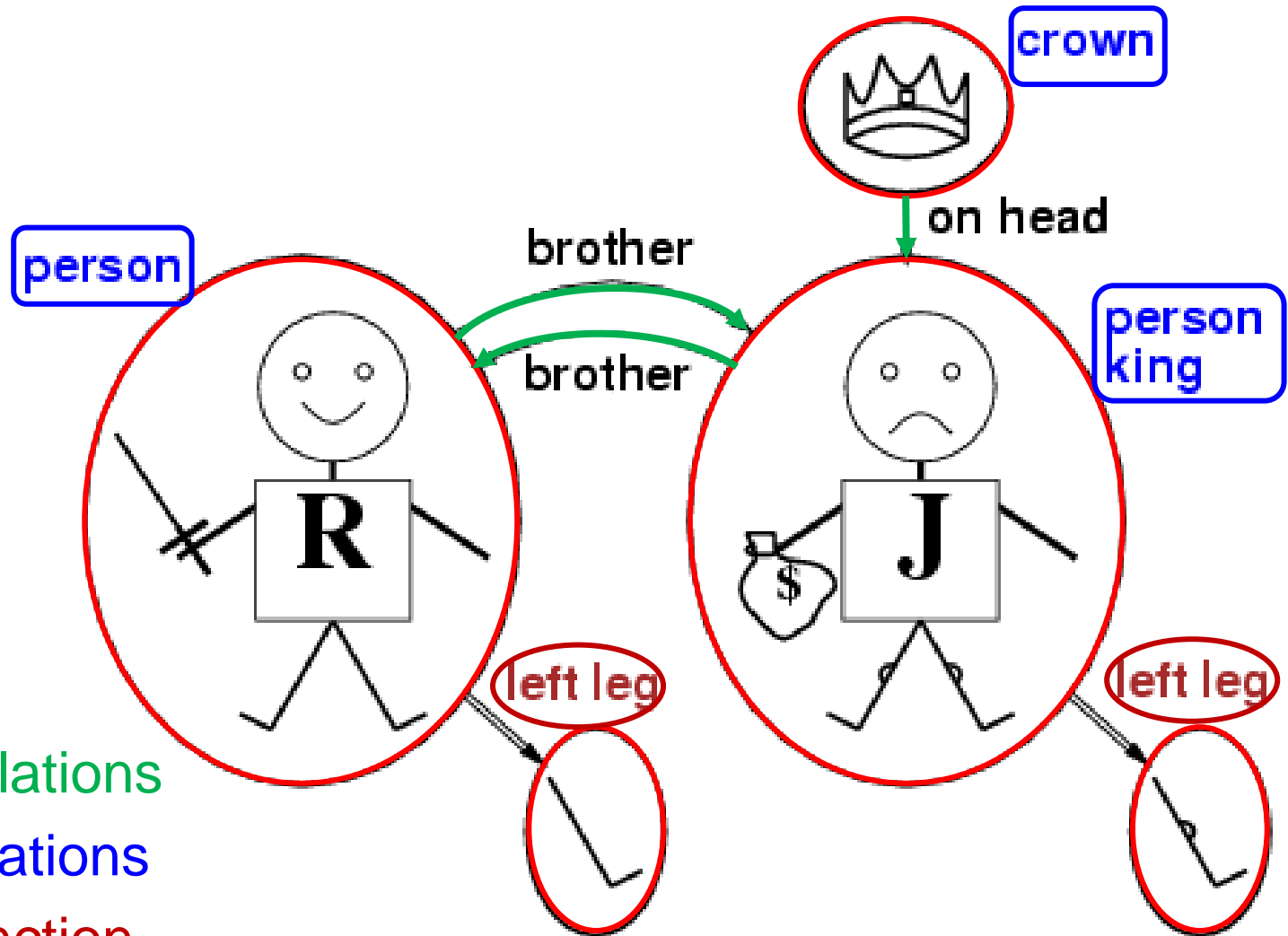
Models for a logic language

- Formal structures that constitute the possible worlds under consideration
 - Each model links the vocabulary of sentences to elements of the possible world → determine the truth of any sentence
- Models for propositional logic link proposition symbols to predefined truth values.
- Models for first-order logic are more interesting with objects.

Models for First-order logic

- The **domain** of a model is the set of objects (or **domain elements**) it contains.
- **Nonempty** — Every possible world must contain at least one object
 - It doesn't matter what these objects are but how many there are in each model.






Models for FOL: A concrete example



- 5 objects
- 2 binary relations
- 3 unary relations
- 1 unary function

Models for FOL: A concrete example

- 5 objects

				
Richard the Lionheart (King of England 1189-1199)	King John (King of England 1199-1215)	The left leg of Richard	The left leg of John	A crown

Models for FOL: A concrete example

- Binary relations
 - The brotherhood relation
 $\{ \langle \text{Richard the Lionheart}, \text{King John} \rangle, \langle \text{King John}, \text{Richard the Lionheart} \rangle \}$
 - The “on head” relation
 $\{ \langle \text{The crown}, \text{King John} \rangle \}$
- Unary relations: “person”, “king”, “crown”
- Functions: “left leg”
 - $\langle \text{Richard the Lionheart} \rangle \rightarrow \text{Richard's left leg}$
 - $\langle \text{King John} \rangle \rightarrow \text{John's left leg}$

Symbols

- **Constant symbols** represents **objects**.
 - E.g., Richard, John, etc.
- **Predicate symbols** stand for **relations**.
 - E.g., Brother , OnHead, Person, King, and Crown, etc.
- **Function symbols** stand for **functions**.
 - E.g., LeftLeg
- Each predicate or function symbol comes with an arity that fixes the number of arguments.
 - E.g., Brother(x,y) \rightarrow binary, LeftLeg(x) \rightarrow unary, etc.
- These symbols begins with uppercase letters by convention.

Intended interpretation

- **Interpretation** specifies exactly which objects, relations and functions are referred to by the symbol.
- Each model includes an (intended) interpretation.
- For example,
 - Richard the Lionheart and King John refers two Kings in England.
 - Brother refers to the brotherhood relation, OnHead refers to the “on head” relation that holds between the crown and King John
 - Person, King, and Crown refer to the sets of objects that are persons, kings, and crowns, respectively.
 - LeftLeg refers to the “left leg” function

The syntax of First-order logic

$Sentence \rightarrow AtomicSentence \mid ComplexSentence$

$AtomicSentence \rightarrow Predicate \mid Predicate(Term, \dots) \mid Term = Term$

$ComplexSentence \rightarrow (Sentence) \mid [Sentence]$

$\mid \neg Sentence$

$\mid Sentence \wedge Sentence$

$\mid Sentence \vee Sentence$

$\mid Sentence \Rightarrow Sentence$

$\mid Sentence \Leftrightarrow Sentence$

$\mid Quantifier Variable, \dots Sentence$

$Term \rightarrow Function(Term, \dots)$

$\mid Constant$

$\mid Variable$

The syntax of First-order logic

Quantifier $\rightarrow \forall \mid \exists$

Constant $\rightarrow A \mid X_1 \mid John \mid \dots$

Variable $\rightarrow a \mid x \mid s \mid \dots$

Predicate $\rightarrow True \mid False \mid After \mid Loves \mid Raining \mid \dots$

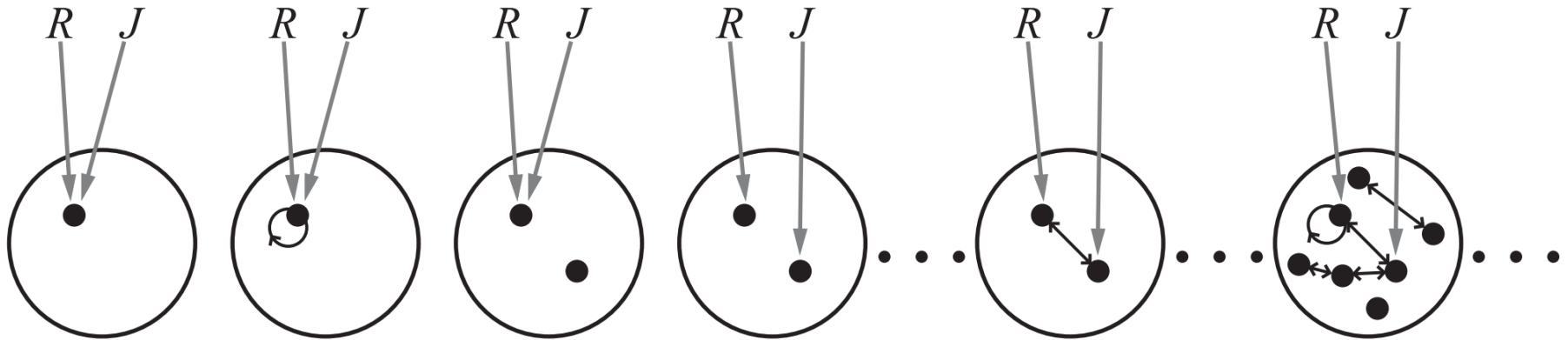
Function $\rightarrow Mother \mid LeftLeg \mid \dots$

OPERATOR PRECEDENCE : $\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

* The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

Possible models in First-order logic

- Similar to propositional logic, entailment, validity, and so on are defined in terms of all possible models.



137,506,194,466 models with six or fewer objects

- The number of possible models is unbounded.
→ checking entailment by the enumeration is **infeasible**

Terms

- A **term** is a logical expression that refers to an object
 - Constant symbols: *John*
 - Function symbols: *LeftLeg(John)*
- A **complex term** is formed by a **function symbol** followed by a **parenthesized list of terms** as arguments.

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

Complex term

The semantics of terms

- Consider a term, $f(t_1, \dots, t_n)$, that refers to some function F
- The arguments refer to objects in the domain, d_1, \dots, d_n .
- The whole term refers to the object that is the value of applying F to d_1, \dots, d_n .
- For example,
 - The *LeftLeg* function symbol refers to the mapping between a person and his left leg, and John refers to King John.
 - *LeftLeg(John)* refers to King John's left leg.
- In this way, the interpretation fixes the referent of every term.

Atomic sentence

- An **atomic sentence** state facts by using a **predicate symbol** followed by a parenthesized list of terms.

$$\text{Atomic sentence} = \text{predicate}(term_1, \dots, term_n)$$

- For example,
 - Brother(Richard, John), Married(Father(Richard), Mother(John))
- It is **true** if the relation referred to by the predicate symbol **holds** among the objects referred to by the arguments

Complex sentences

- A **complex sentences** are made from atomic sentences using **connectives**.
- For example,
 - $\neg \text{Brother}(\text{LeftLeg}(\text{Richard}), \text{John})$
 - $\text{Brother}(\text{Richard}, \text{John}) \wedge \text{Brother}(\text{John}, \text{Richard})$
 - $\text{King}(\text{Richard}) \vee \text{King}(\text{John})$
 - $\neg \text{King}(\text{Richard}) \Rightarrow \text{King}(\text{John})$
 - ...
- Syntax and semantics are the same as in propositional logic.

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**

Quantifiers: Universal quantification

- Expressions of general rules

$\forall \langle \text{variables} \rangle \langle \text{sentence} \rangle$

- E.g., “All kings are persons.”: $\forall x \text{ King}(x) \Rightarrow \text{Person}(x)$
- E.g., “Students of FIT are smart.”: $\forall x \text{ Student}(x, \text{FIT}) \Rightarrow \text{Smart}(x)$

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

- It is equivalent to the conjunction of instantiations of P .

$\text{Student}(\text{Lan}, \text{FIT}) \Rightarrow \text{Smart}(\text{Lan})$

$\wedge \text{Student}(\text{Tuan}, \text{FIT}) \Rightarrow \text{Smart}(\text{Tuan})$

$\wedge \text{Student}(\text{Long}, \text{FIT}) \Rightarrow \text{Smart}(\text{Long})$

$\wedge \dots$

Variables

- A **variable** is a term all by itself and able to serve as the argument of a function
 - E.g., in predicates $King(x)$ or in function $LeftLeg(x)$.
- Usually represented by lowercase letters
- A term with no variables is called a **ground term**.

A common mistake to avoid

- Typically, \Rightarrow is the main connective with \forall
 - The conclusion of the rule **just for** those objects for whom the **premise is true**
 - It says nothing at all about individuals for whom the premise is false.
- **Common mistake:** using \wedge as the main connective with \forall
 - $\forall x \text{ Student}(x, \text{FIT}) \wedge \text{Smart}(x)$
 - It means “Everyone is a student of FIT and Everyone is smart.”
- **Too strong** implication

Quantifiers: Existential quantification

- Expressions of “some cases”

$\exists \langle \text{variables} \rangle \langle \text{sentence} \rangle$

- E.g., “Some students of FIT are smart.”

$$\exists x \text{ Student}(x, \text{FIT}) \wedge \text{Smart}(x)$$

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

- It is equivalent to the **disjunction** of **instantiations** of P .

$$\text{Student}(\text{Lan}, \text{FIT}) \wedge \text{Smart}(\text{Lan})$$

$$\vee \text{Student}(\text{Tuan}, \text{FIT}) \wedge \text{Smart}(\text{Tuan})$$

$$\vee \text{Student}(\text{Long}, \text{FIT}) \wedge \text{Smart}(\text{Long})$$

$$\vee \dots$$

Another common mistake to avoid

- Typically, \wedge is the main connective with \exists
- **Common mistake:** using \Rightarrow as the main connective with \exists
 - $\exists x \text{ Student}(x, \text{FIT}) \Rightarrow \text{Smart}(x)$
 - It is true even with anyone who is not at FIT.
- **Too weak** implication

Nested quantifiers

- Multiple quantifiers enable more complex sentences.
- **Simplest cases:** Quantifiers are of the same type
 - $\forall x \forall y \text{ Brother}(x, y) \Rightarrow \text{Sibling}(x, y)$
 - $\forall x \forall y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$
- **Mixtures**
 - $\forall x \exists y \text{ Loves}(x, y)$ – “Everybody loves somebody.”
 - $\exists x \forall y \text{ Loves}(y, x)$ – “There is someone loved by everyone.”
- The **order of quantification** is therefore **very important**.

Nested quantifiers

- Two quantifiers used with the same variable name leads to confusion.

$$\forall x (Crown(x) \vee (\exists x \text{ Brother}(\text{Richard}, x)))$$

- **Rule:** The variable belongs to the **innermost** quantifier that mentions it.
- **Workaround:** Use different variable names with nested quantifiers, e.g., $\exists z \text{ Brother}(\text{Richard}, z)$

Quantifier duality

- \forall and \exists relate to each other through negation.

- For example,

$$\bullet \forall x \text{ Likes}(x, \text{IceCream}) \qquad \neg \exists x \neg \text{Likes}(x, \text{IceCream})$$

$$\bullet \exists x \text{ Likes}(x, \text{Brocoli}) \qquad \neg \forall x \neg \text{Likes}(x, \text{IceCream})$$

- De Morgan's rules

$$\forall x \neg P \equiv \neg \exists x P$$

$$\neg \forall x P \equiv \exists x \neg P$$

$$\forall x P \equiv \neg \exists x \neg P$$

$$\exists x P \equiv \neg \forall x \neg P$$

$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$

$$P \wedge Q \equiv \neg(\neg P \vee \neg Q)$$

$$P \vee Q \equiv \neg(\neg P \wedge \neg Q)$$

Equality symbol =

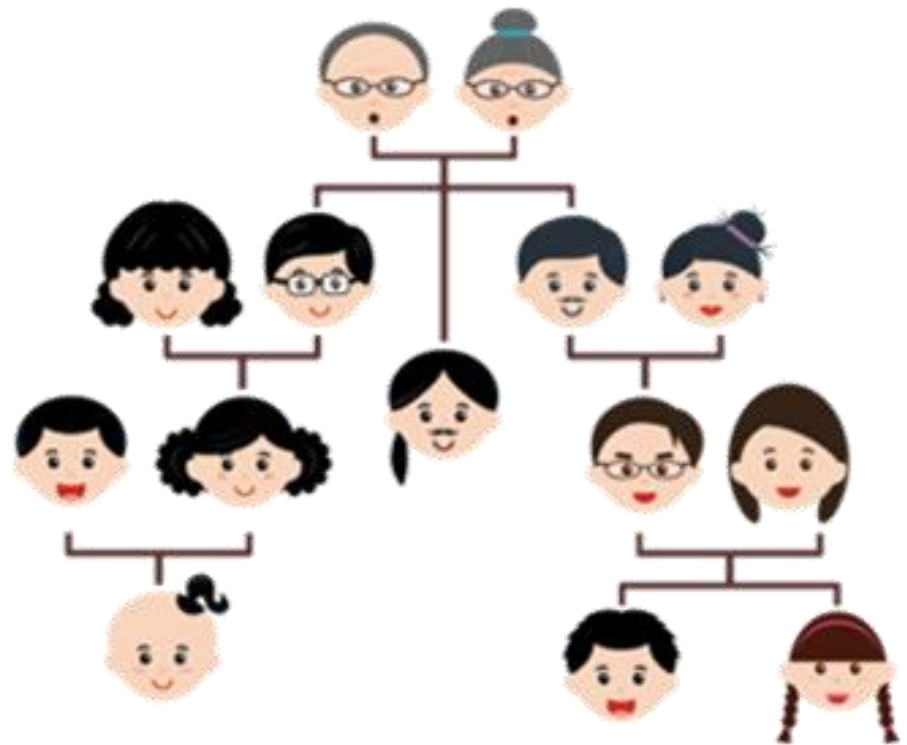
- $term_1 = term_2$ is **true** under a given interpretation iff $term_1$ and $term_2$ refer to the same object
 - E.g., $Father(John) = Henry$ means that $Father(John)$ and $Henry$ refer to the same object.
 - It states facts about a given function
- The **negation** insists that **two terms are not the same**.
 $\exists x, y \text{ Brother}(x, Richard) \wedge \text{Brother}(y, Richard) \wedge \neg(x = y)$

Using First-Order Logic

- *Assertions and Queries in First-Order Logic*
- *The Kinship Domain*
- *Numbers, Sets, and Lists*
- *The Wumpus World*



The kinship domain



- Unary predicates
 - Male and Female
- Binary predicates represent kinship relations.
 - Parenthood, brotherhood, marriage, etc.
 - Parent, Sibling, Brother , Sister, Child, Daughter, Son, Spouse, Wife, Husband, Grandparent , Grandchild , Cousin, Aunt, and Uncle.
- Functions
 - Mother and Father, each person has exactly one of each of these.

The kinship domain: Axioms

- One's mother is one's female parent

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

- One's husband is one's male spouse:

$$\forall w, h \text{ Husband}(h, w) \Leftrightarrow \text{Male}(h) \wedge \text{Spouse}(h, w)$$

- Male and female are disjoint categories:

$$\forall x \text{ Male}(x) \Leftrightarrow \neg \text{Female}(x)$$

- Parent and child are inverse relations:

$$\forall p, c \text{ Parent}(p, c) \Leftrightarrow \text{Child}(c, p)$$

- A grandparent is a parent of one's parent:

$$\forall g, c \text{ Grandparent}(g, c) \Leftrightarrow \exists p \text{ Parent}(g, p) \wedge \text{Parent}(p, c)$$

- A sibling is another child of one's parents:

$$\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \neg(x = y) \wedge \exists p \text{ Parent}(p, x) \wedge \text{Parent}(p, y)$$

The kinship domain: Theorems

- **Theorems:** logical sentences that are entailed by the axioms
 - E.g., $\forall x, y \text{ Sibling}(x, y) \Leftrightarrow \text{Sibling}(y, x)$
- Theorems **reduce the cost of deriving new sentences**.
 - They do not increase the set of conclusions that follow from $KB \rightarrow$ no value from a pure logical point of view
 - They are essential from a practical point of view.

The set domain

- Sets are the empty set and those made by adjoining something to a set.
 - $\forall s \text{ Set}(s) \Leftrightarrow (s = \{ \}) \vee (\exists x, s_2 \text{ Set}(s_2) \wedge s = \{x|s_2\})$
- The empty set has no elements adjoined into it.
 - $\neg \exists x, s \{x|s\} = \{ \}$
- Adjoining an element already in the set has no effect:
 - $\forall x, s \ x \in s \Leftrightarrow s = \{x|s\}$
- The only members of a set are the elements that were adjoined into it.
 - $\forall x, s \ x \in s \Leftrightarrow \exists y, s_2 (s = \{y|s_2\} \wedge (x = y \vee x \in s_2))]$

The set domain

- Can you interpret the following sentences?
 - $\forall s_1, s_2 \quad s_1 \sqsubseteq s_2 \Leftrightarrow (\forall x \quad x \in s_1 \Rightarrow x \in s_2)$
 - $\forall s_1, s_2 \quad s_1 = s_2 \Leftrightarrow (s_1 \sqsubseteq s_2 \wedge s_2 \sqsubseteq s_1)$
 - $\forall x, s_1, s_2 \quad x \in (s_1 \cap s_2) \Leftrightarrow (x \in s_1 \wedge x \in s_2)$
 - $\forall x, s_1, s_2 \quad x \in (s_1 \cup s_2) \Leftrightarrow (x \in s_1 \vee x \in s_2)$

The Wumpus world: Input – Output

- Typical percept sentence
 - `Percept([Stench, Breeze, Glitter, None, None]. 5)`
- Actions
 - `Turn(Right), Turn(Left), Forward, Shoot, Grab, Release, Climb`
- To determine the best action, construct a query
 - `ASKVARS($\exists a \text{ BestAction}(a, 5)$)`
 - Returns a **binding list** such as `{a/Grab}`

The Wumpus world: The KB

- Perception

- $\forall t, s, g, m, c \text{ Percept } ([s, \text{Breeze}, g, m, c], t) \Rightarrow \text{Breeze}(t)$
- $\forall t, s, b, m, c \text{ Percept } ([s, b, \text{Glitter}, m, c], t) \Rightarrow \text{Glitter}(t) \quad \dots$

- Reflex

- $\forall t \text{ Glitter}(t) \Rightarrow \text{BestAction}(\text{Grab}, t)$

- Environment definition:

$$\begin{aligned} \forall x, y, a, b \quad \text{Adjacent}([x, y], [a, b]) \Leftrightarrow \\ (x = a \wedge (y = b - 1 \vee y = b + 1)) \\ \vee (y = b \wedge (x = a - 1 \vee x = a + 1)) \end{aligned}$$

The Wumpus world: Hidden properties

- Properties of squares

$$\forall s, t \text{ } At(Agent, s, t) \wedge Breeze(t) \Rightarrow Breezy(s)$$

- Squares are breezy near a pit

- **Diagnostic** rule --- infer cause from effect

$$\forall s \text{ } Breezy(s) \Leftrightarrow \exists r \text{ } Adjacent(r, s) \wedge Pit(r)$$

- **Causal** rule --- infer effect from cause

$$\forall r \text{ } Pit(r) \Leftrightarrow [\forall s \text{ } Adjacent(r, s) \Rightarrow Breezy(s)]$$

Quiz 01: Writing FOL sentences

- Represent the following sentences with first-order logic using the given predicates
 - $Student(x)$ means x is student.
 - $Smart(x)$ means x is smart.
 - $Loves(x, y)$ means x loves y .
1. All students are smart.
 2. There exists a smart student.
 3. Every student loves some student.
 4. Every student loves some other student
 5. There is a student who is loved by every other student.

First-Order Inference

- *Inference Rules for Quantifiers*
- *Reduction to Propositional Inference*



$$\forall x. Cat(x) \Rightarrow Cute(x)$$

Universal Instantiation (UI)

- It is possible to infer any sentence obtained by **substituting a ground term for the variable**.
- Let $SUBST(\theta, \alpha)$ be the result of applying the substitution θ to the sentence α .
- Then the **rule of Universal Instantiation** is written

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

- for any variable v and ground term g .

Universal Instantiation: An example

- Suppose our knowledge base contains

$$\forall x \textit{ King}(x) \wedge \textit{ Greedy}(x) \Rightarrow \textit{ Evil}(x)$$

- Then it is permissible to infer any of the following sentences

$$\textit{ King}(\textit{ John}) \wedge \textit{ Greedy}(\textit{ John}) \Rightarrow \textit{ Evil}(\textit{ John})$$

$$\textit{ King}(\textit{ Richard}) \wedge \textit{ Greedy}(\textit{ Richard}) \Rightarrow \textit{ Evil}(\textit{ Richard})$$

$$\begin{aligned} \textit{ King}(\textit{ Father}(\textit{ John})) \wedge \textit{ Greedy}(\textit{ Father}(\textit{ John})) \\ \Rightarrow \textit{ Evil}(\textit{ Father}(\textit{ John})) \end{aligned}$$

...

with substitutions $\{x/\textit{ John}\}$, $\{x/\textit{ Richard}\}$, and
 $\{x/\textit{ Father}(\textit{ John})\}$, respectively

Existential Instantiation (EI)

- It is possible to **replace the variable by a single new constant symbol**.
- The **rule of Existential Instantiation** is written

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

- for any sentence α , variable v , and constant symbol k that does not appear elsewhere in KB .
- For example, from $\exists x \text{Crown}(x) \wedge \text{OnHead}(x, \text{John})$
infer $\text{Crown}(C_1) \wedge \text{OnHead}(C_1, \text{John})$
 - As long as C_1 does not appear in KB , called **Skolem constant**.

Universal / Existential Instantiation

- The **UI rule** can be **applied many times** to produce different consequences.
- The **EI rule** can be **applied once**, and then the existentially quantified sentence is discarded.
 - E.g., discard $\exists x \text{ Kill}(x, \text{Victim})$ after adding $\text{Kill}(\text{Murderer}, \text{Victim})$
 - The new *KB* is not **logically equivalent** to the old but shown to be **inferentially equivalent**.

Reduction to propositional inference

- Every first-order *KB* and query can be **propositionalized** in such a way that entailment is preserved.
 - A ground sentence is entailed by new *KB* iff entailed by original *KB*.
- For example, suppose the *KB* contains just the sentences

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\text{King}(\text{John})$

$\text{Greedy}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

- Apply UI with substitutions, $\{x/\text{John}\}$ and $\{x/\text{Richard}\}$

$\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$

$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$

Reduction to propositional inference

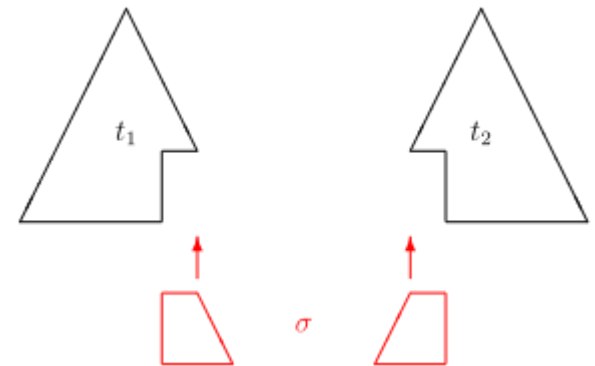
- **Problem:** *When the KB includes a function symbol, the set of possible ground-term substitutions is infinite.*
 - E.g., the *Father* symbol, infinitely many nested terms such as *Father(Father(Father(John)))* can be constructed.
- **Herbrand's Theorem (1930):** *If an original FOL KB $\models \alpha$, α is entailed by a **finite** subset of the propositionalized KB.*
 - For $n = 0$ to ∞ do
 - Create a propositional KB by instantiating with depth- n terms
 - See if α is entailed by this KB
 - $n = 0$: *Richard* and *John*
 - $n = 1$: *Father(Richard)* and *Father(John)*, etc.

Reduction to propositional inference

- **Problem:** *The inference works if sentence α is entailed, but it loops if α is not entailed.*
- **Theorems of Turing (1936) and Church (1936):** *The question of **entailment for first-order logic** is **semidecidable**.*
 - Algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.

Unification and Lifting

- *A First-Order Inference Rule*
- *Unification*
- *Storage and Retrieval*



Problem with propositionalization

- A lots of generated sentences seems to be irrelevant.
- For example, with the following *KB*

$$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$$

King(John)

Greedy(John)

Brother(Richard, John)

- It seems obvious that *Evil(John)*
- However, propositionalization produces lots of irrelevant sentences

$$\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$$

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

A first-order inference rule

- **If** there is some substitution θ making each of the conjuncts of the premise identical to sentences already in the *KB*.
- **Then** the conclusion can be asserted after applying θ .
- For example,

$\forall x \text{ King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$

$\forall y \text{ Greedy}(y)$

$\text{King}(\text{John})$

$\text{Brother}(\text{Richard}, \text{John})$

- Applying the substitution $\{x/\text{John}, y/\text{John}\}$ to $\text{King}(x)$ and $\text{Greedy}(x)$, $\text{King}(\text{John})$ and $\text{Greedy}(y)$ will make them identical in pairs.
- Thus, infer the conclusion of the implication

Generalized Modus Ponens (GMP)

- For atomic sentences p_i , p'_i and q , where there exists θ such that $SUBST(\theta, p'_i) = SUBST(\theta, p_i)$, for all i ,

$$\frac{p'_1, p'_2, \dots, p'_n, \quad (p_1, p_2, \dots, p_n \Rightarrow q)}{SUBST(\theta, q)}$$

- For example, 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)*
 $SUBST(\theta, q)$ is *Evil(John)*
- All variables assumed universally quantified
- A **lifted version** of Modus Ponens \rightarrow **sound** inference rule

Unification

- Find substitutions that make different logical expressions look identical

$$\text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

- For example,

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

Problem is due to use of same variable x in both sentences



Standardizing apart eliminates overlap of variables
Knows(z , Steve)

Most General Unifier (MGU)

- $UNIFY(Knows(John, x), Knows(y, z)) = \theta$
 1. $\theta = \{y/John, x/z\}$
 2. $\theta = \{y/John, x/John, z/John\}$
- 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.

$$MGU = \{y/John, x/z\}$$

MGU: Some examples

ω_1	ω_2	MGU
$A(B, C)$	$A(x, y)$	$\{x/B, y/C\}$
$A(x, f(D, x))$	$A(E, f(D, y))$	$\{x/E, y/E\}$
$A(x, y)$	$A(f(C, y), z)$	$\{x/f(C, y), y/z\}$
$P(A, x, f(g(y)))$	$P(y, f(z), f(z))$	$\{y/A, x/f(z), z/g(y)\}$
$P(x, g(f(A)), f(x))$	$P(f(y), z, y)$	No MGU
$P(x, f(y))$	$P(z, g(w))$	No MGU

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 expression
 y , a variable, constant, list, or compound expression
 θ , the substitution built up so far (optional, defaults to empty)
if $\theta = \text{failure}$ **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(x .ARGS, y .ARGS, UNIFY(x .OP, y .OP, θ))
else if LIST?(x) **and** LIST?(y) **then**
 return UNIFY(x .REST, y .REST, UNIFY(x .FIRST, y .FIRST, θ))
else return failure

The unification algorithm

function UNIFY-VAR(var, x, θ) **returns** a substitution
if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
else if OCCUR-CHECK?(var, x) **then return** failure
else return add $\{var/x\}$ to θ

Atom_mgu($P(a,x,f(g(y)))$, $P(z,f(z),f(w))$) **a is a constant**

↓
Term_list_mgu($\langle a, x, f(g(y)) \rangle$, $\langle z, f(z), f(w) \rangle$, \emptyset)

↓
Term_list_mgu_aux($\langle x, f(g(y)) \rangle$, $\langle f(z), f(w) \rangle$, Term_mgu(a, z, \emptyset), \emptyset)

↓
Variable_mgu(z, a, \emptyset)

↓
Term_list_mgu($\langle x, f(g(y)) \rangle$, $\langle f(a), f(w) \rangle$, $\{a/z\}$)

↓
 $\{a/z\}$

↓
Term_list_mgu_aux($\langle f(g(y)) \rangle$, $\langle f(w) \rangle$, Term_mgu(x, f(a), \emptyset), $\{a/z\}$)

↓
Variable_mgu(x, f(a), \emptyset)

↓
 $\{f(a)/x\}$



Term_list_mgu($\langle \rangle$, $\langle \rangle$, Term_mgu($f(g(y))$, $f(w)$, $\{a/z, f(a)/x\}$))



Term_list_mgu($\langle g(y) \rangle$, $\langle w \rangle$, $\{a/z, f(a)/x\}$))



Term_list_mgu_aux($\langle \rangle$, $\langle \rangle$, Term_mgu($g(y)$, w , \emptyset) $\{a/z, f(a)/x\}$))



Variable_mgu(w , $g(y)$, \emptyset)



$\{g(y)/w\}$



Term_list_mgu($\langle \rangle$, $\langle \rangle$, $\{a/z, f(a)/x, g(y)/w\}$))



$\{a/z, f(a)/x, g(y)/w\}$

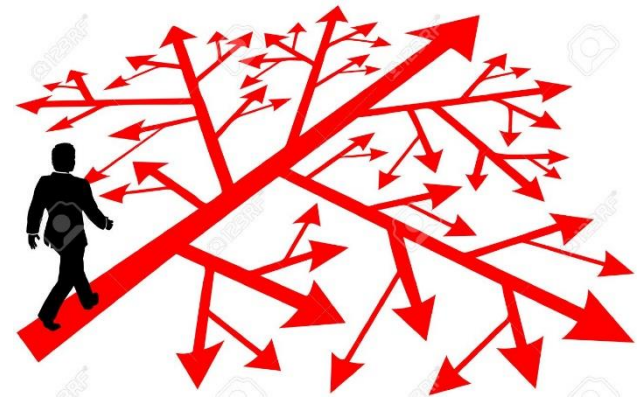
Quiz 02: Find the MGU

- Find the MGU when performing $UNIFY(p, q)$

ω_1	ω_2	MGU
$P(f(A), g(x))$	$P(y, y)$?
$P(A, x, h(g(z)))$	$P(z, h(y), h(y))$?
$P(x, f(x), z)$	$P(g(y), f(g(b)), y)$?
$P(x, f(x))$	$P(f(y), y)$?
$P(x, f(z))$	$P(f(y), y)$?

Forward Chaining

- *First-order Definite Clauses*
- *A Simple Forward Chaining Algorithm*
- *Efficient Forward Chaining*



First-order definite clause

- A **definite clause** is a disjunction of literals of which exactly one is positive.
 - It is either **atomic** or an **implication** whose antecedent is a conjunction of positive literals and consequent is a positive literal.
 - E.g., $King(x) \wedge Greedy(x) \Rightarrow Evil(x), King(John), Greedy(y)$
- **First-order literals** can include variables, which are assumed to be **universally quantified**.
- Not every KB can be converted into a set of definite clauses due to the **single-positive-literal restriction**.

FOL definite clause: An example

- Consider the following problem

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.

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

$\exists x Owns(Nono, x) \wedge Missile(x)$

$Owns(Nono, M_1)$

$Missile(M_1)$

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

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

$American(West)$

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

$Enemy(Nono, America)$

A simple forward chaining algorithm

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

inputs: KB , the knowledge base, a set of first-order definite clauses
 α , the query, an atomic sentence

local variables: new , the new sentences inferred on each iteration

repeat until new is empty

$new \leftarrow \{ \}$

for each $rule$ **in** KB **do**

$(p_1 \wedge \dots \wedge p_n \Rightarrow q) \leftarrow \text{STANDARDIZE-VARIABLES}(rule)$

for each θ such that $\text{SUBST}(\theta, p_1 \wedge \dots \wedge p_n) = \text{SUBST}(\theta, p'_1 \wedge \dots \wedge p'_n)$
for some p'_1, \dots, p'_n in KB

$q' \leftarrow \text{SUBST}(\theta, q)$

if q' does not unify with some sentence already in KB or new **then**

add q' to new

$\varphi \leftarrow \text{UNIFY}(q', \alpha)$

if φ is not *fail* **then return** φ

add new to KB

return *false*

Forward chaining: An example

American(West)

Missile(M1)

Owns(Nono,M1)

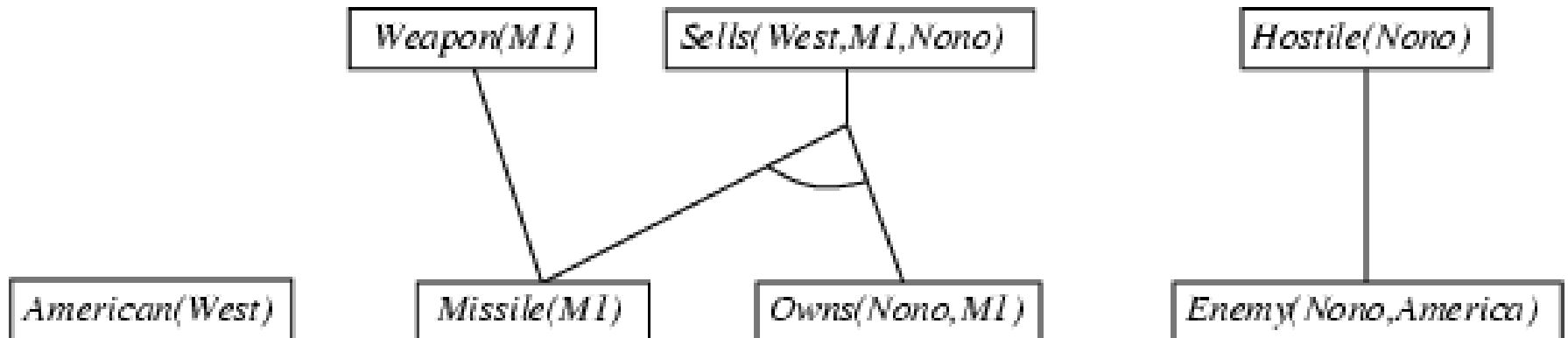
Enemy(Nono,America)

Forward chaining: An example

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

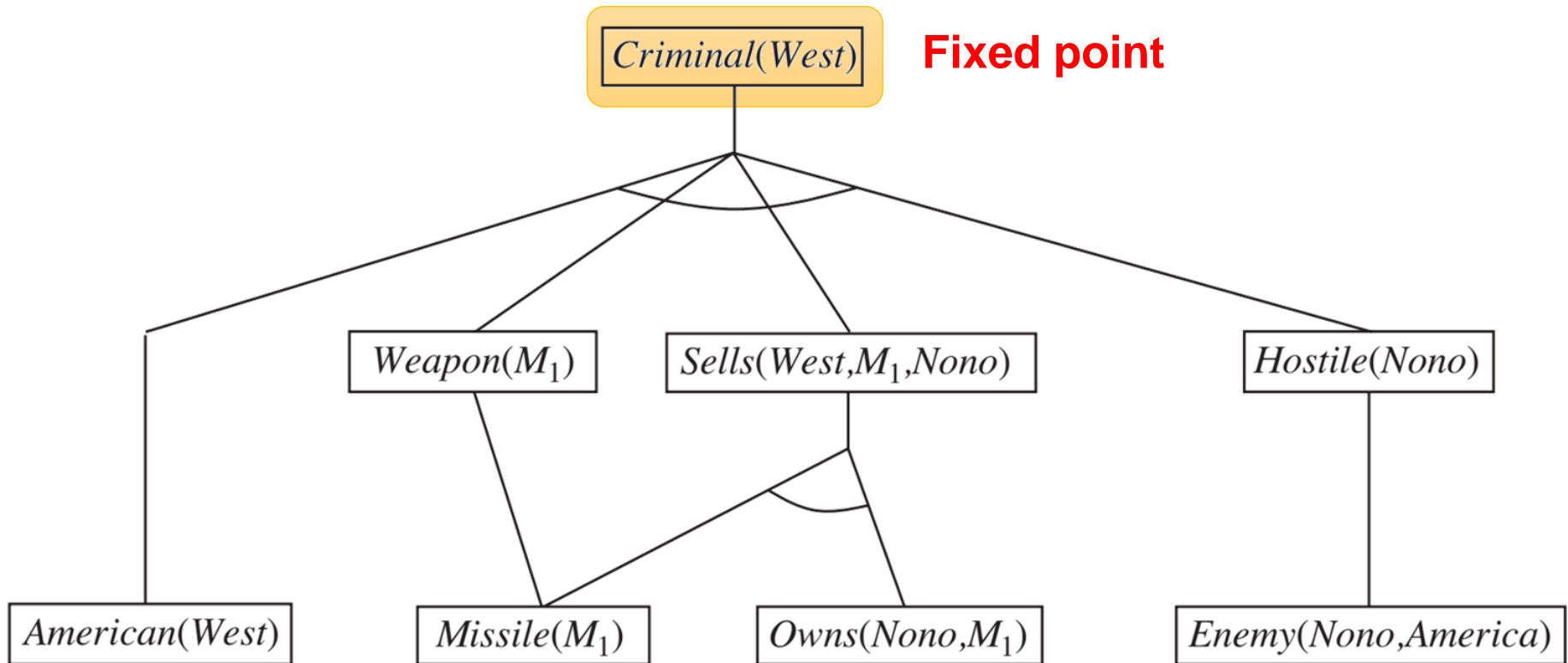
$Missile(x) \Rightarrow Weapon(x)$

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



Forward chaining: An example

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



Forward chaining

- Soundness?

- YES, every inference is just an application of GMP.

- Completeness?

- YES for definite clause knowledge bases.
- It answers every query whose answers are entailed by any *KB* of definite clauses.

- Terminate for Datalog in finite number of iterations

- **Datalog** = first-order definite clauses + **no functions**

- Entailment with definite clauses is **semidecidable**.

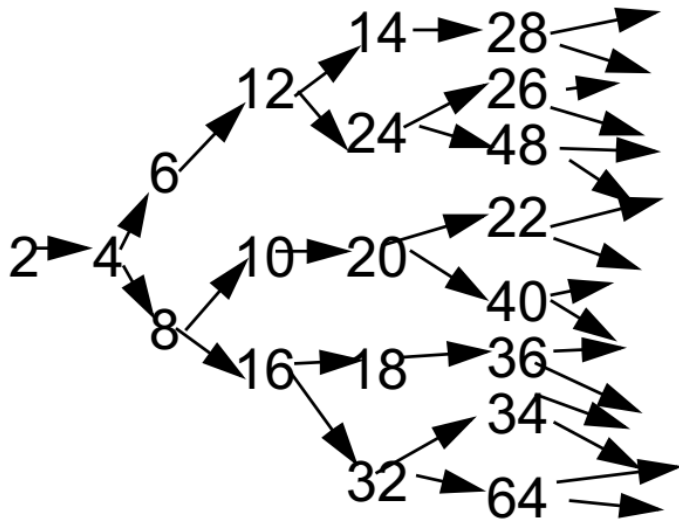
- May not terminate in general if α is not entailed, unavoidable.

Renaming

- A fact is not “**new**” if it is just a renaming of a known fact.
- One sentence is a renaming of another if they are identical except for the names of the variables.
 - E.g., Likes(x, IceCream) vs. Likes(y, IceCream)

Definite clauses with function symbols

- Inference can explode forward and may never terminate.



$Even(x) \Rightarrow Even(plus(x, 2))$

$Integer(x) \Rightarrow Even(times(2, x))$

$Even(x) \Rightarrow Integer(x)$

$Even(2)$

Efficient forward chaining

- Incremental forward chaining
 - No need to match a rule on iteration k if a premise was not 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 $Missile(M_1)$
- Widely used in deductive databases

Quiz 03: Forward chaining

- Given a KB containing the following sentence
 1. $Parent(x, y) \wedge Male(x) \Rightarrow Father(x, y)$
 2. $Father(x, y) \wedge Father(x, z) \Rightarrow Sibling(y, z)$
 3. $Parent(Tom, John)$
 4. $Male(Tom)$
 5. $Parent(Tom, Fred)$
- Perform the forward chaining until a fixed point is reached.

Backward Chaining



A backward chaining algorithm

function FOL-BC-ASK($KB, query$) **returns** a generator of substitutions
return FOL-BC-OR($KB, query, \{ \}$)

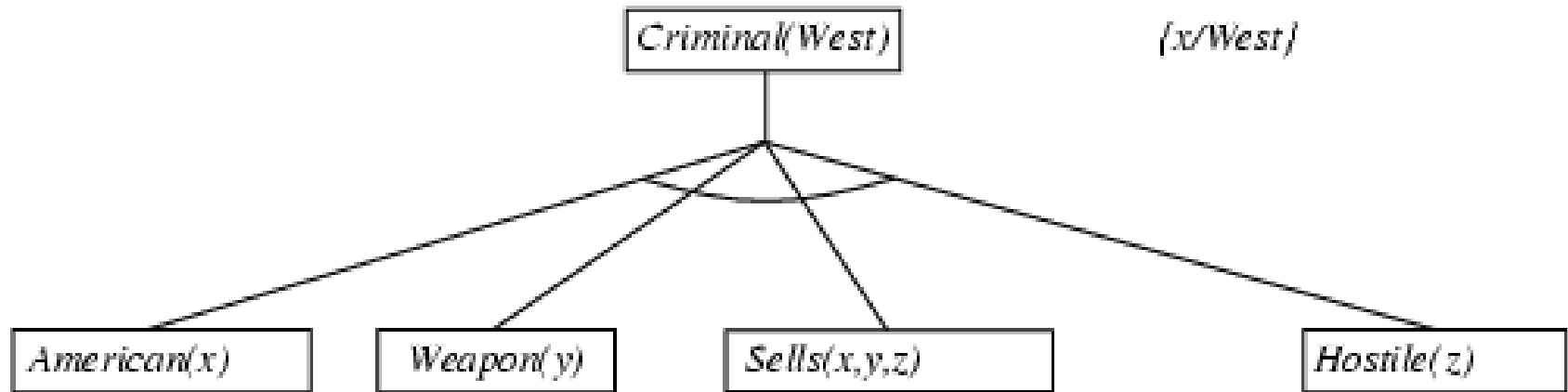
generator FOL-BC-OR($KB, goal, \theta$) **yields** a substitution
for each rule ($lhs \Rightarrow rhs$) in FETCH-RULES-FOR-GOAL($KB, goal$) **do**
 (lhs, rhs) \leftarrow STANDARDIZE-VARIABLES((lhs, rhs))
 for each θ' in FOL-BC-AND($KB, lhs, UNIFY(rhs, goal, \theta)$) **do**
 yield θ'

generator FOL-BC-AND($KB, goals, \theta$) **yields** a substitution
if $\theta = \text{failure}$ **then return**
else if LENGTH($goals$) = 0 **then yield** θ
else do
 $first, rest \leftarrow$ FIRST($goals$), REST($goals$)
 for each θ' in FOL-BC-OR($KB, SUBST(\theta, first), \theta$) **do**
 for each θ'' in FOL-BC-AND($KB, rest, \theta'$) **do**
 yield θ''

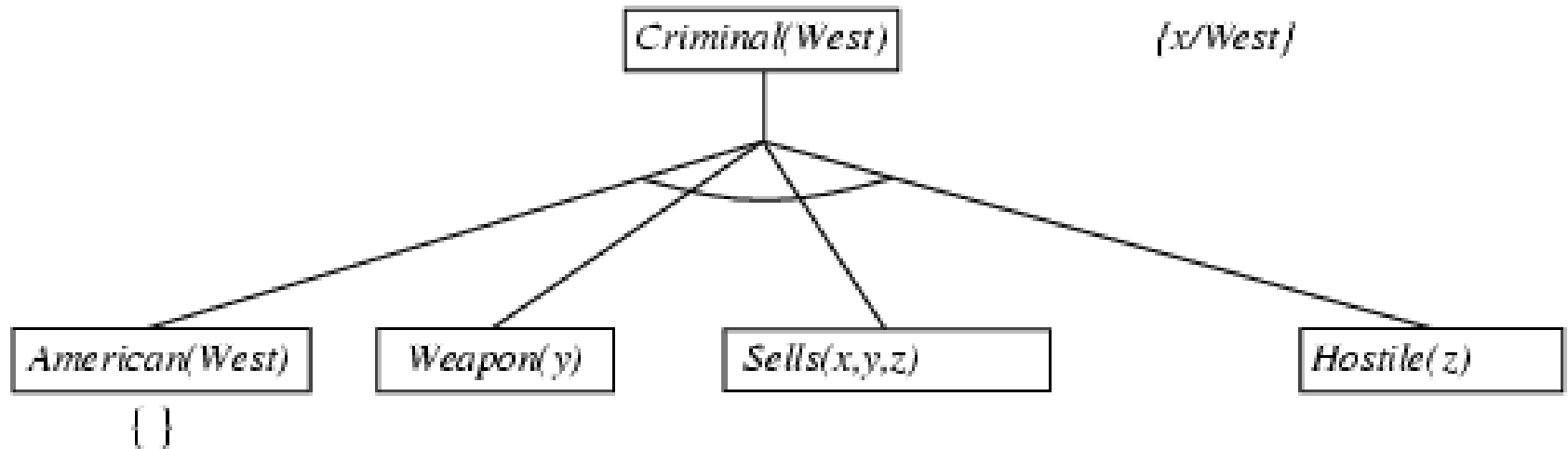
Backward chaining: An example

Criminal(West)

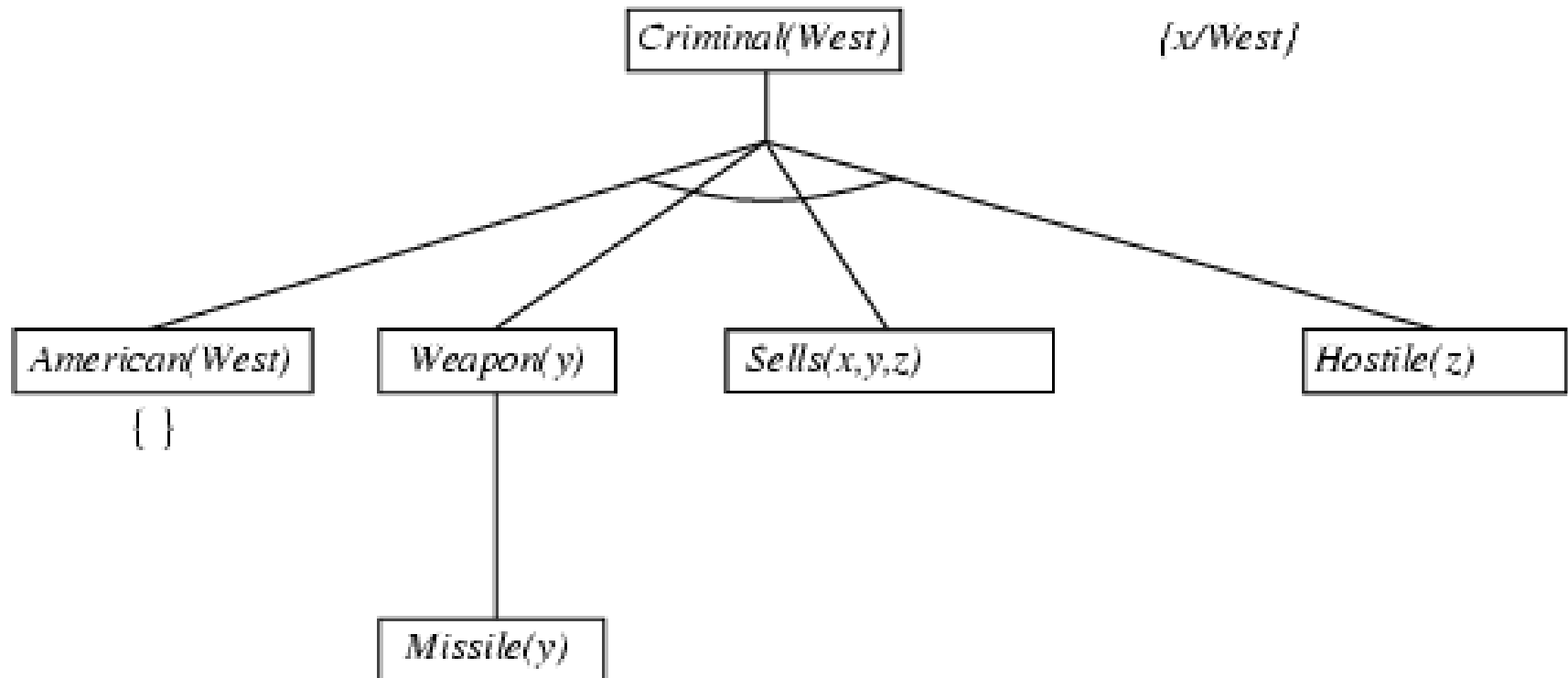
Backward chaining: An example



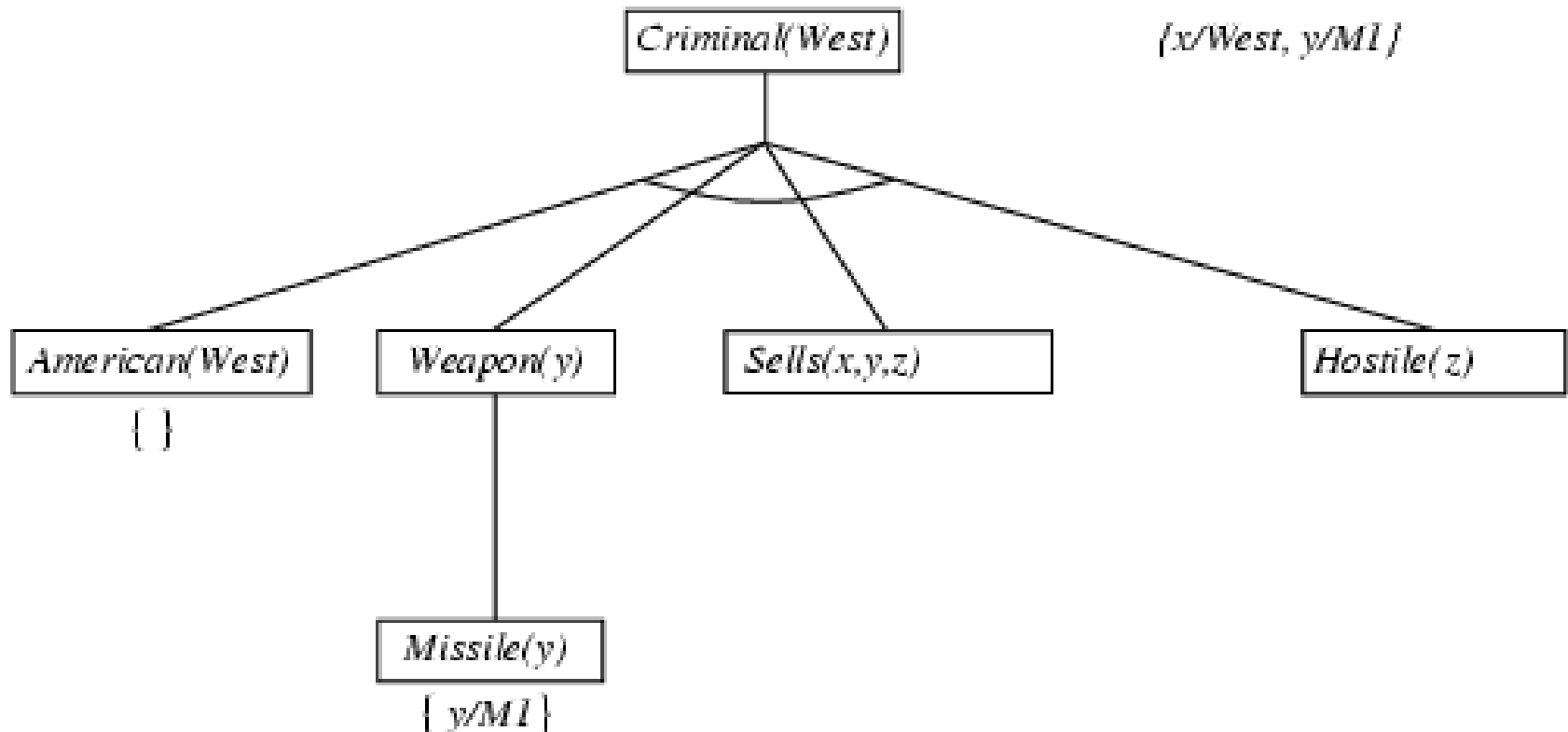
Backward chaining: An example



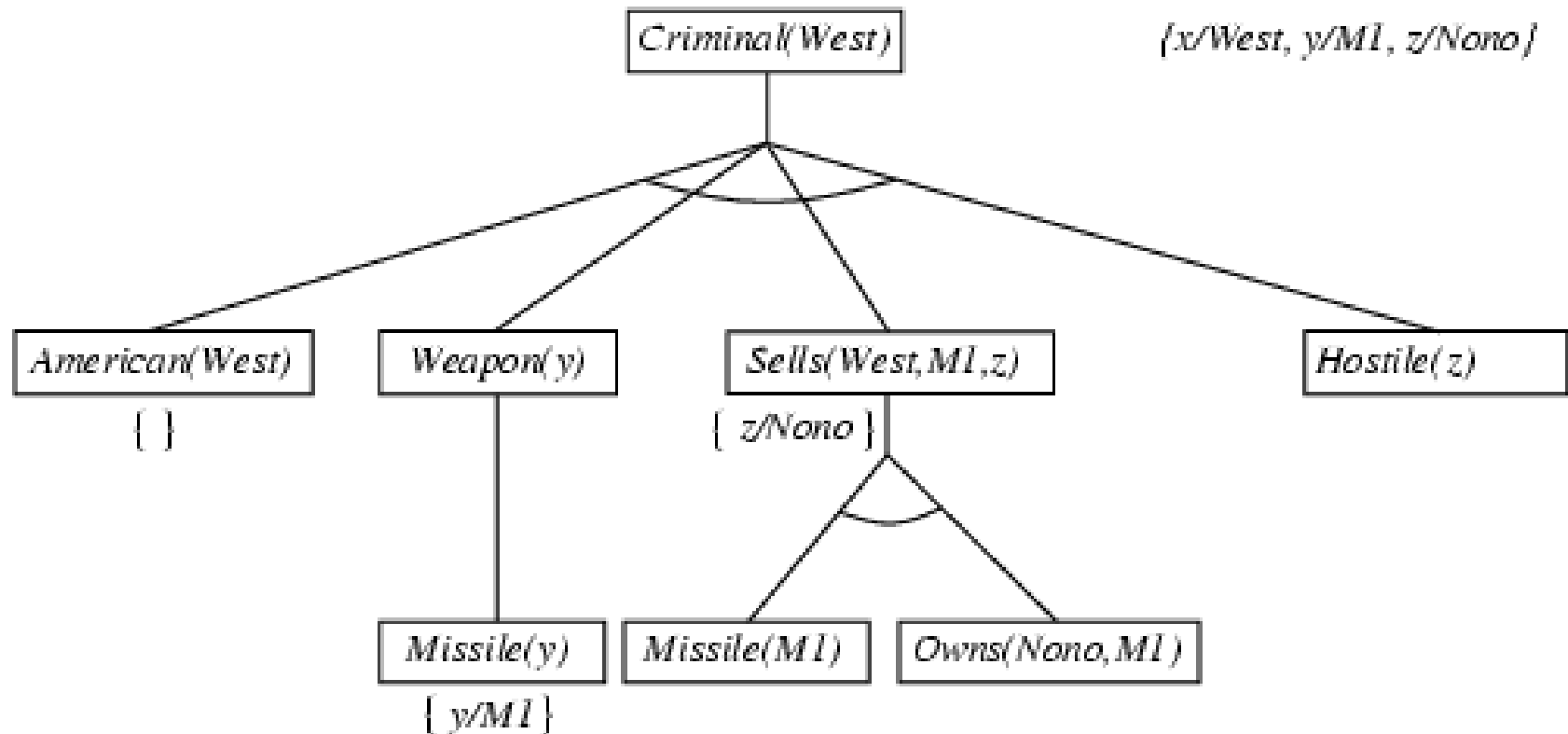
Backward chaining: An example



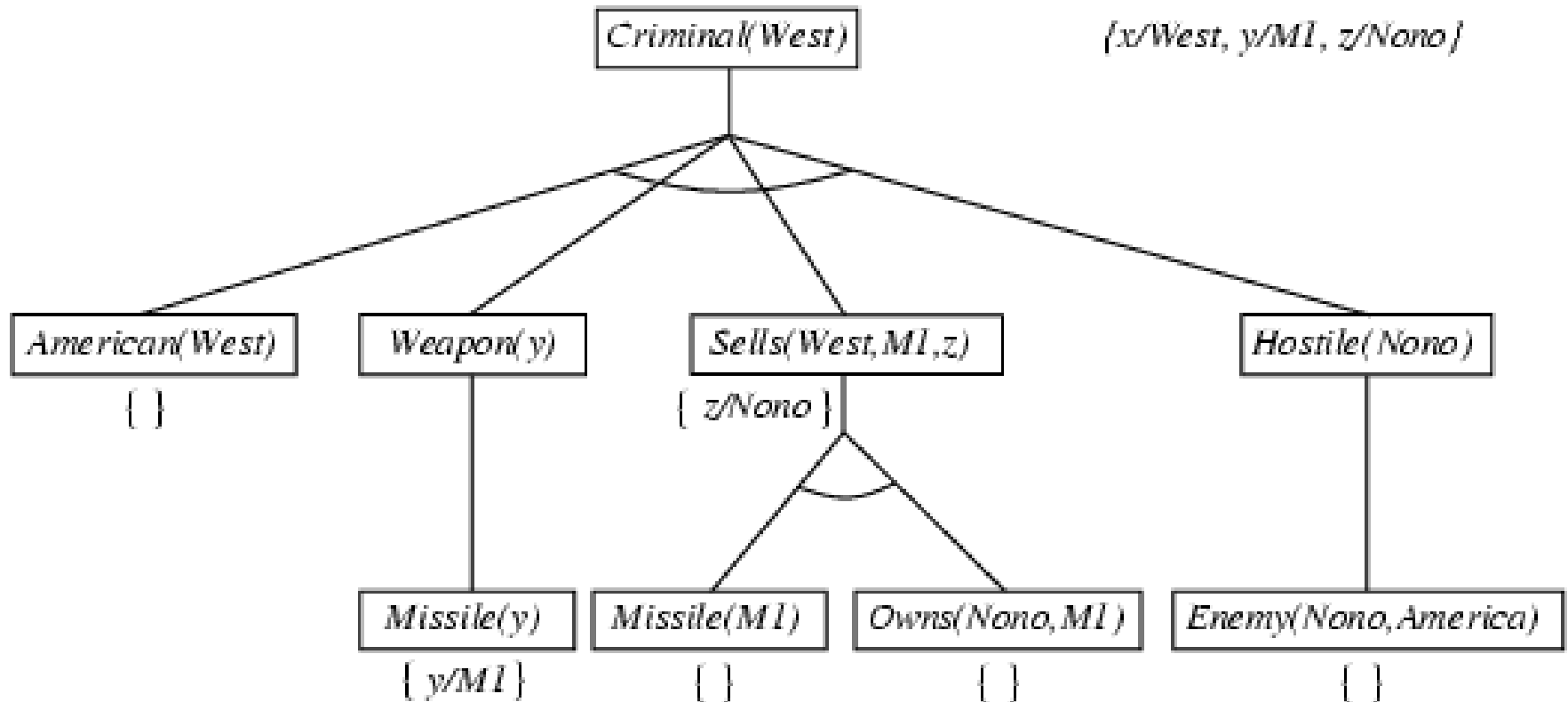
Backward chaining: An example



Backward chaining: An example



Backward chaining: An example



Properties of backward chaining

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

Quiz 04: Backward chaining

- Given a KB containing the following sentence
 1. $Parent(x, y) \wedge Male(x) \Rightarrow Father(x, y)$
 2. $Father(x, y) \wedge Father(x, z) \Rightarrow Sibling(y, z)$
 3. $Parent(Tom, John)$
 4. $Male(Tom)$
 5. $Parent(Tom, Fred)$
- Find solution(s) to each of the following queries
 - $Parent(Tom, x)$
 - $Father(f, s)$
 - $Father(Tom, s)$
 - $Sibling(a, b)$

Quiz 04: Backward chaining

- Query: $\text{Parent}(\text{Tom}, x)$
 - Answers: $(\{x/\text{John}\}, \{x/\text{Fred}\})$
- Query: $\text{Father}(\text{Tom}, s)$
 - Subgoal: $\text{Parent}(\text{Tom}, s) \wedge \text{Male}(\text{Tom})$
 - $\{s/\text{John}\}$
 - Subgoal: $\text{Male}(\text{Tom})$
 - Answer: $\{s/\text{John}\}$
 - $\{s/\text{Fred}\}$
 - Subgoal: $\text{Male}(\text{Tom})$
 - Answer: $\{s/\text{Fred}\}$
 - Answers: $(\{s/\text{John}\}, \{s/\text{Fred}\})$

Resolution



CNF for First-order logic

- **First-order resolution** requires that **sentences be in CNF**.
- For example, the sentence

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

becomes, in CNF,

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

- *Every sentence of first-order logic can be converted into an inferentially equivalent CNF sentence.*
 - The CNF sentence will be unsatisfiable just when the original sentence is unsatisfiable \rightarrow perform **proofs by contraction**.

Conversion to CNF

Everyone who loves all animals is loved by someone.

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

1. Eliminate implications

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

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

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

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

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

Conversion to CNF

Everyone who loves all animals is loved by someone.

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

3. **Standardize variables:** each quantifier uses a different one

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

4. **Skolemize:** remove existential quantifiers by elimination

- Simple case: translate $\exists x P(x)$ into $P(A)$, where A is a new constant.
 - However, $\forall x [\text{Animal}(A) \wedge \neg \text{Loves}(x, A)] \vee [\text{Loves}(B, x)]$ has an entirely different meaning.
- The arguments of the **Skolem function** are all universally quantified variables in whose scope the existential quantifier appears.

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

Conversion to CNF

Everyone who loves all animals is loved by someone.

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

5. Drop universal quantifiers

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

6. Distribute \vee over \wedge

$$\begin{aligned} & [\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)] \wedge \\ & [\neg \text{Loves}(x, F(x)) \vee \text{Loves}(G(x), x)] \end{aligned}$$

The resolution inference rule

- Simply a lifted version of the propositional resolution rule

- Formulation
$$\frac{l_1 \vee \cdots \vee l_k}{m_1 \vee \cdots \vee m_n}$$

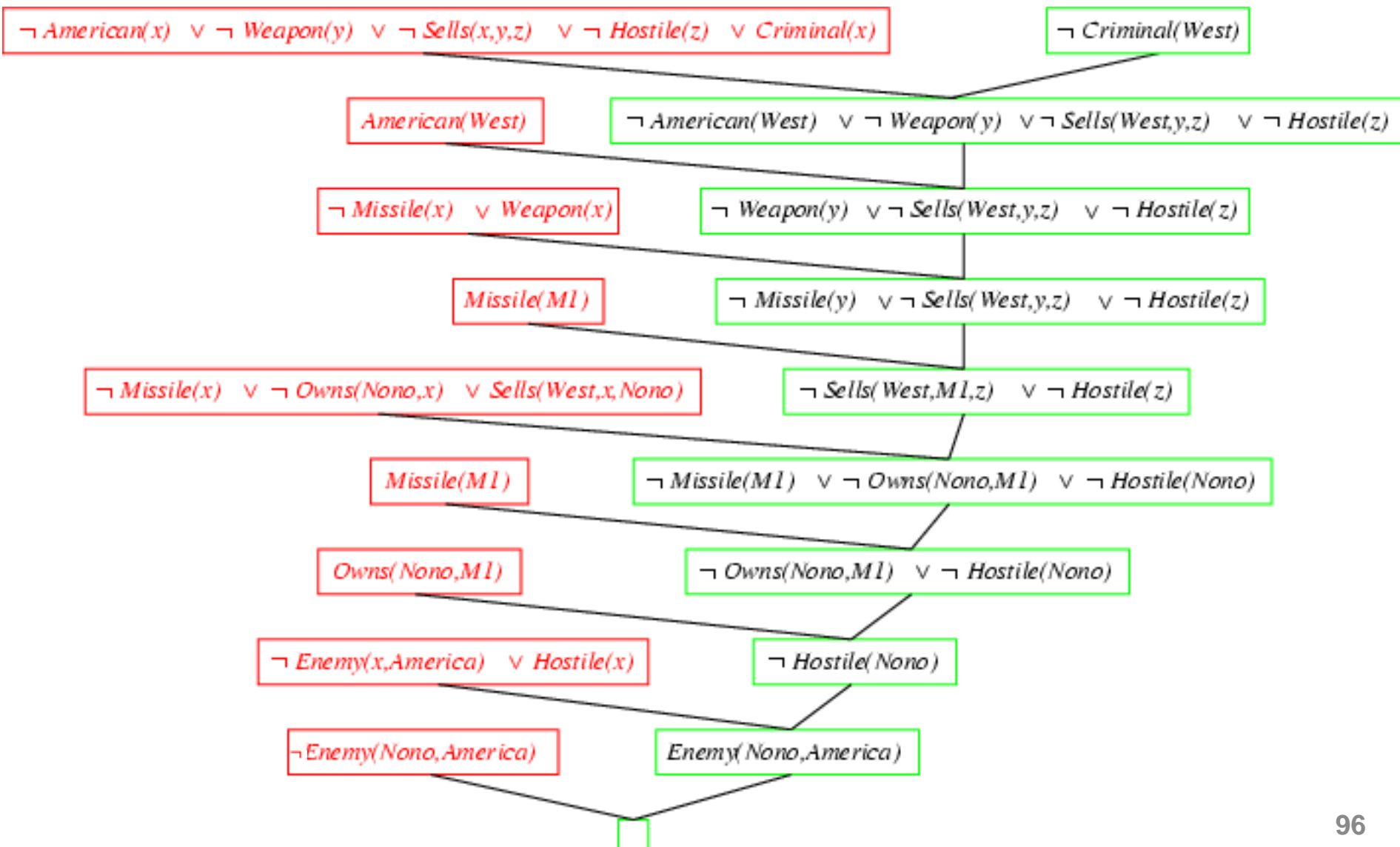
SUBST(θ , $l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n$)

- where $UNIFY(l_i, \neg m_j) = \theta$

- For example,

$$\frac{\begin{array}{l} \textit{Animal}(F(x)) \vee \textit{Loves}(G(x), x) \\ \neg \textit{Loves}(u, v) \vee \neg \textit{Kills}(u, v) \end{array}}{\textit{Animal}(F(x)) \vee \neg \textit{Kills}(G(x), x)} \quad \theta = \{u/G(x), v/x\}$$

Resolution: An example



Resolution: Another example

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?

A. $\forall x [\forall y \text{ Animal}(y) \Rightarrow \text{Loves}(x, y)] \Rightarrow [\exists y \text{ Loves}(y, x)]$

B. $\forall x [\exists z \text{ Animal}(z) \wedge \text{Kills}(x, z)] \Rightarrow [\forall y \neg \text{Loves}(y, x)]$

C. $\forall x \text{ Animal}(x) \Rightarrow \text{Loves}(\text{Jack}, x)$

D. $\text{Kills}(\text{Jack}, \text{Tuna}) \vee \text{Kills}(\text{Curiosity}, \text{Tuna})$

E. $\text{Cat}(\text{Tuna})$

F. $\forall x \text{ Cat}(x) \Rightarrow \text{Animal}(x)$

G. $\neg \text{Kills}(\text{Curiosity}, \text{Tuna})$

Resolution: Another example

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?

A. $Animal(F(x) \vee Loves(G(x), x))$

$\neg Loves(x, F(x)) \vee Loves(G(x), x)$

B. $\neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kills(x, z)$

C. $\neg Animal(x) \vee \neg Loves(Jack, x)$

D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

E. $Cat(Tuna)$

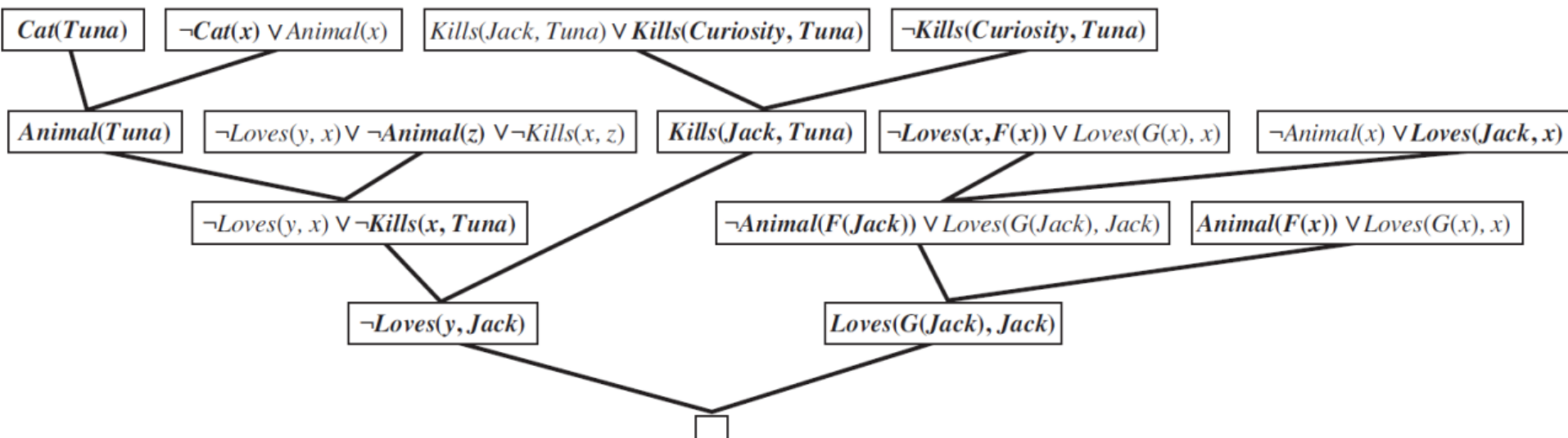
F. $\neg Cat(x) \vee Animal(x)$

G. $\neg Kills(Curiosity, Tuna)$

Resolution: Another example

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.



Quiz 05: Resolution

- Given a KB of the following sentences
 - Anyone whom Mary loves is a football star.
 - Any student who does not pass does not play.
 - John is a student.
 - Any student who does not study does not pass.
 - Anyone who does not play is not a football star.
- Prove that If John does not study, Mary does not love John.
- Write the FOL sentences using only the given predicates

Loves(x, y): “x loves y”

Star(x): “x is a football star”

Student(x): “x is a student”

Pass(x): “x passes”

Play(x): “x plays”

Study(x): “x studies”



THE END