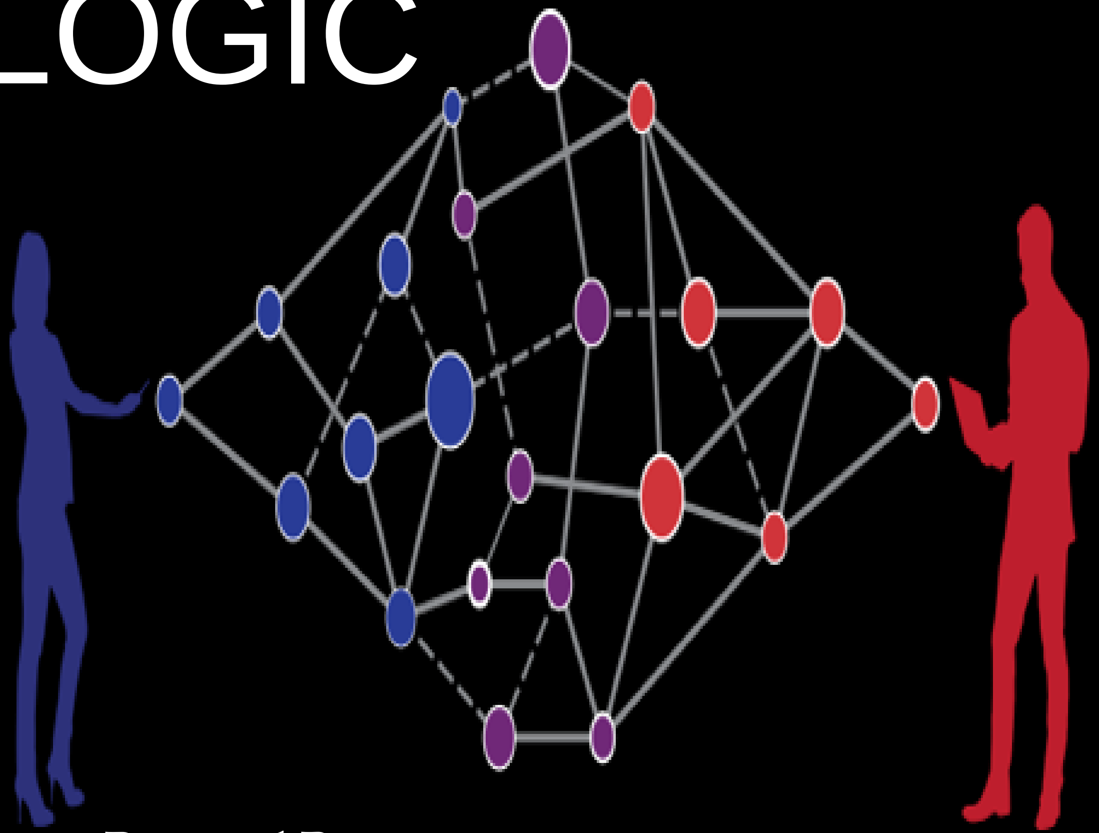
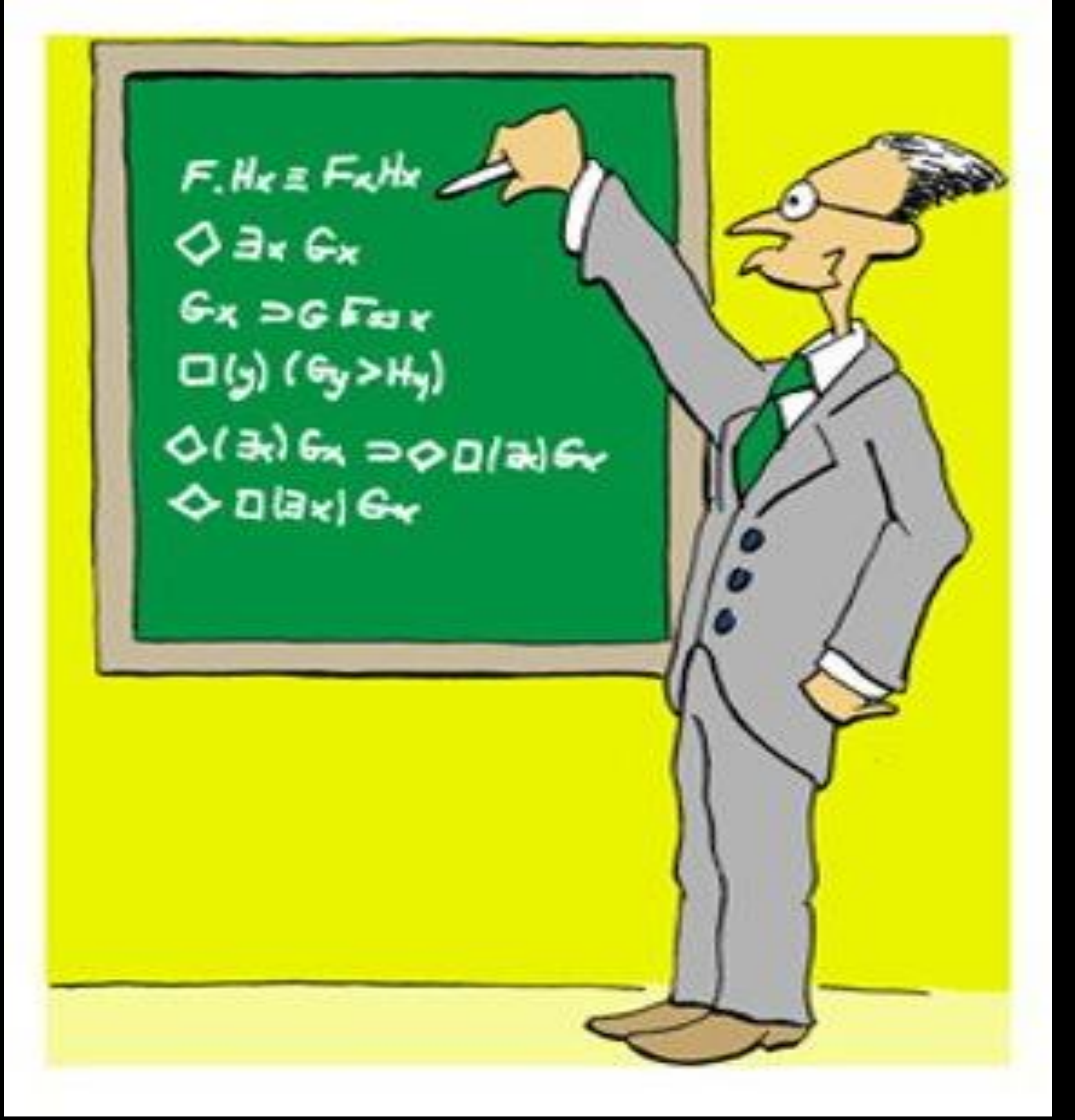
FIRST ORDER LOGIC





Prepared By

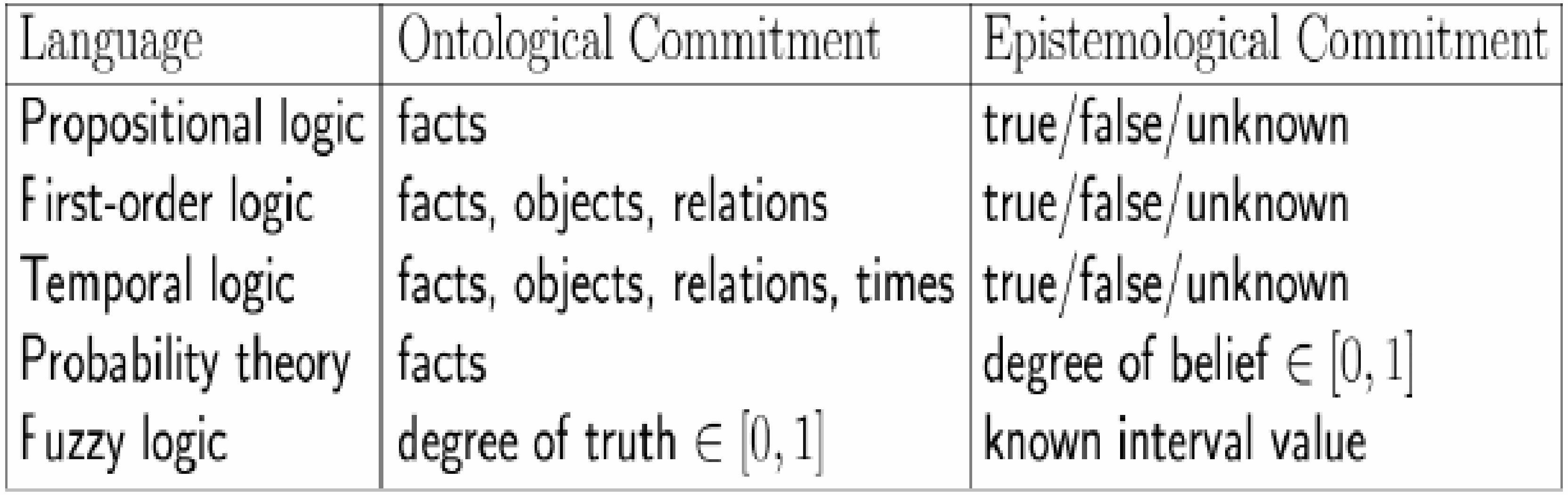
-Anooja Joy

FIRST-ORDER LOGIC

- **Propositional logic** Assumes that the world contains **facts** of from **yes or no.**
- There is a need for a language that is **declarative**(knowledge + inference based on truth relation between sentences), **compositional**(meaning of a sentence is meaning of its parts), **context-independent**, **more expressive**(to deal with partial information using disjunction and negation) and **unambiguous**.
- First-order logic is also called **Predicate logic** and **First-order predicate calculus (FOPL)**. It is a formal representation of logic in the form of **quantifier**s. In predicate logic, the **input** is taken as an **entity**, and the **output** it gives is either **true or false.**
- **First-order logic** Assumes that the world contains
  - **Objects** people, houses, numbers, theories, Donald Duck, colors, centuries, ...
  - **Relations** red, round, prime, multistoried, ... brother of, bigger than, part of, has color, occurred after, owns
  - **Functions** +, middle of, father of, one more than, beginning of, …

FORMAL LANGUAGES: ONTOLOGY AND EPISTEMOLOGY

- **Ontology** is the study of existence claim ie, what it assumes about **nature of reality**. Ie, an inventory of what exists. Ontological commitment means **“WHAT EXISTS IN THE WORLD**”.
- **Epistemology** is a major branch of philosophy that concerns the forms, nature, and preconditions of knowledge. Epistological commitment is a commitment that considers **possible states of knowledge** with respect to each fact. Eepistemological commitment means **“WHAT AN AGENT BELIEVES ABOUT FACTS”.**



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 $\in [0, 1]$
Fuzzy logic	degree of truth $\in [0, 1]$	known interval value

CONCEPTUALIZATION

- One plus two equals three
  - **Objects:** one, two, three, one plus two
  - **Function:** plus
  - **Relation:** equals
- Squares neighboring the wumpus are smelly
  - **Objects:** wumpus, square
  - **Property:** smelly
  - **Relations:** neighboring
- Evil King John ruled England in 1200
  - **Objects:** John, England, 1200
  - **Properties:** evil, king
  - **Relations:** ruled

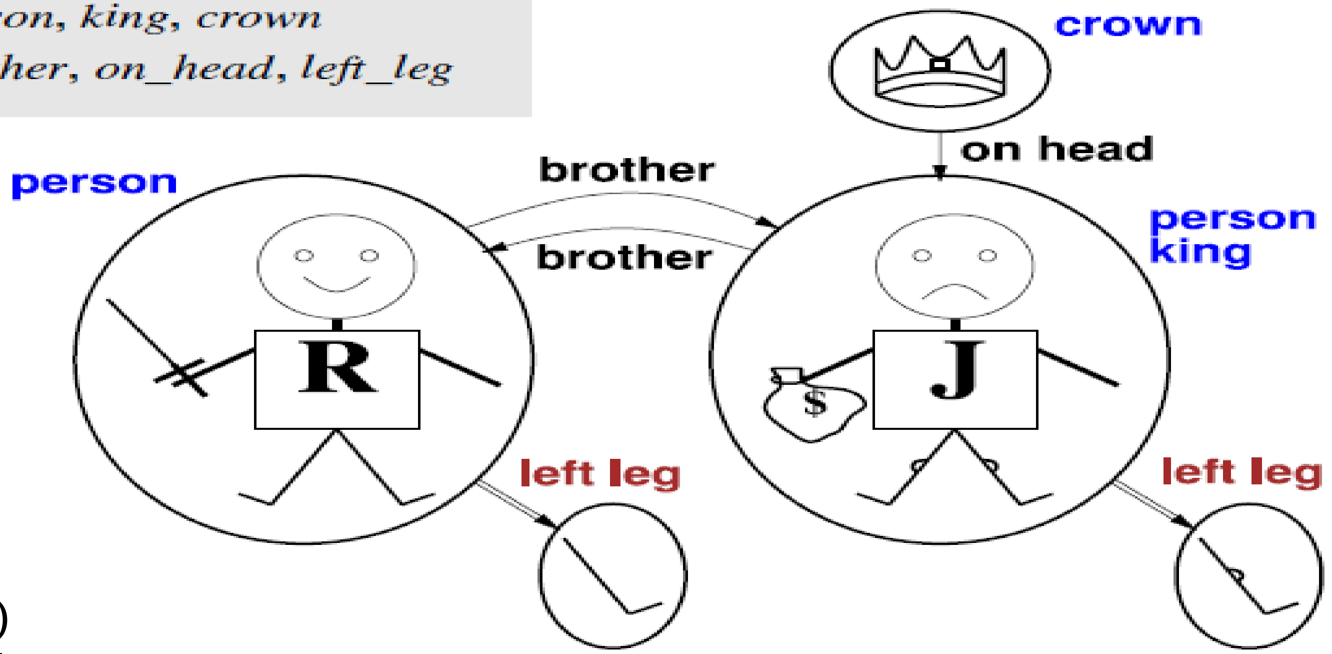
SYNTAX AND SEMANTICS OF FOL

MODELS FOR FOL

- **Model** contains **domain elements** and **relations** among them.
  - Models for **propositional logic** are just **sets of truth values** for the proposition symbols.
  - Models for **first-order logic** have **objects, relations** and **functions**.
- **Syntax:** formal structure of sentences. It defines the way of representing the **given predicates**. As these predicates are represented via **quantifiers**, there are different types of quantifiers used.
- **Semantics:** truth of sentences wrt models. It defines the sense of the given predicate. It allows to make more logical expression by devising its semantics. Semantics allow us to understand the sentence meaning.
- **Predicates:** express relationships between objects
- **Quantifiers:** Quantifiers are used to express properties of entire collections of objects, instead of enumerating the objects by name if a logic that allows object is found. It can restrict the scope of variables
- In FOL, **variables** refer to things in the world and can quantify over the-talk about all or some of them without naming them explicitly.

MODELS FOR FOL

Constants:	*KingJohn, Richard*
Predicates:	*person, king, crown*
Functions:	*brother, on_head, left_leg*



Objects (Ex:- Richard)
Relations (Ex:- King)
functions (Ex:- LeftLeg)

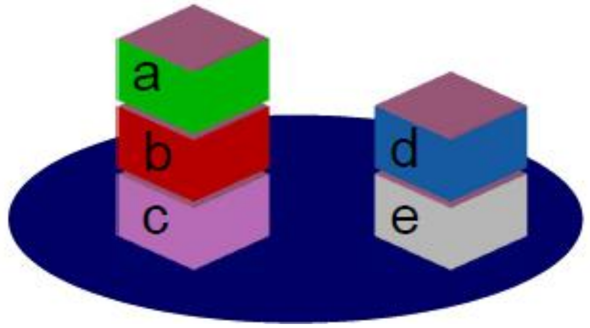
- The picture depicts Richard the Lionheart, King of England from 1189 to 1199; his younger brother, the evil King John, who ruled from 1199 to 1215; the left legs of Richard and John; and a crown.

MODELS FOR FOL: OBJECTS

- The model contain **5 objects**, 2 **binary relations**(brother, onhead), **3 unary relations**(person, person king, crown) and **1 unary function**(left leg)
- **Objects** refers to an entity that exists in the real world.
- The picture shows a model with five **objects:**
  - Richard the Lionheart
  - His younger brother
  - The evil King John
  - The left legs of Richard and John
  - A crown

MODELS FOR FOL: RELATIONS

- The objects in the model may be **related** in various ways, In the figure Richard and John are **brothers**. Formally speaking, a **relation** is just the **set of tuples of objects that are related.**
- A **tuple** is **a collection of Objects arranged in a fixed order and is written with angle brackets surrounding the objects.**
- **Eg:** The **brotherhood** relation in this model is the set {**<Richard the Lionheart, King John>**,**<King John, Richard the Lionheart>**} The crown is on King John's head, so the "**on head**" relation contains just one tuple, **<the crown, King John>**.
- The relation can be:
  - **binary relation** relating pairs of objects (**Eg:** "**Brother**")
  - **unary relation** representing a common object (**Eg:**"**Person**" representing both **Richard** and **John**)



Eg: Set of blocks {a, b, c, d, e}. The "On" relation includes: On = {<a,b>,<b,c> ,<d,e> }<a,b> represents the predicate On(A,B)

MODELS FOR FOL: FUNCTIONS

- Certain kinds of relationships are best considered as functions that relates an object to exactly one object.
- **Eg:**- each person has one left leg, so the model has a **unary** "**left leg**" **function** that includes the following mappings (Richard the Lionheart) ----> Richard's left leg

SYNTAX: ELEMENTS AND SYMBOLS OF FOL

- **The elements for which different symbols are defined are:**
  - **Objects(Constant Symbols):** It refers to an entity that exists in the real world. **For example,** Ram, John, etc. are referred to as Objects.
  - **Functions(Function Symbols):** Any function performed by the object/on the object. **For example,** LeftLeg, writes, eats, Sqrt, etc. are some of the functions.
  - **Relations(Predicate Symbols):** The relation of an object with the other object defines its relation**. For example,** brother, mother, king,>, =, ... etc. are some types of relations which exist in the real world.

1. **Constant Symbols:** These symbols are used to represent the objects. **Eg:** KingJohn, 2, Koblenz, C,…
2. **Function Symbols:** These symbols are used to represent the functions.**Eg:** Sqrt, LeftLegOf,...
3. **Predicate Symbols** are used to represent relations. **Eg:** brother, mother, king,>,
4. **Variable Symbols** x, y, a, b, ...
5. **Connectives** $\wedge \vee \neg \Rightarrow \Leftrightarrow$
6. **Equality:** ==
7. **Quantifiers** $\forall \exists$

SYNTAX: ATOMIC AND COMPLEX SENTENCES

- **Atomic Sentences:** These sentences are formed via predicate symbols may or may not be followed by a list of terms.

Example:

Parents(Ram, Sita) where Ram and Sita are the parents.

Brother(KingJohn,RichardTheLionheart)

Length(LeftLegOf(KingJohn)))

- **Complex Sentences:** These sentences make use of logical connectives to construct more complex sentences from atomic sentences .

Example:

Sibling(KingJohn,Richard) $\Rightarrow$ Sibling(Richard,KingJohn)

$\neg$Brother (LeftLeg(Richard), John)

King (Richard) $\lor$ King (John)

King (Richard)$\Rightarrow$ King(John)

EXAMPLES: ATOMIC SENTENCES

$$\underbrace{\underbrace{Brother}_{\text{predicate}}\,(\,\underbrace{\underbrace{KingJohn}_{\text{constant}}}_{\text{term}},\ \underbrace{\underbrace{RichardTheLionheart}_{\text{constant}}}_{\text{term}}\,)}_{\text{atomic sentence}}$$

$$\underbrace{\underbrace{>}_{\text{predicate}}\ (\underbrace{\underbrace{Length}_{\text{function}}(\underbrace{LeftLegOf}_{\text{function}}(\underbrace{Richard}_{\text{constant}}))}_{\text{term}},\ \underbrace{\underbrace{Length}_{\text{function}}(\underbrace{LeftLegOf}_{\text{function}}(\underbrace{KingJohn}_{\text{constant}}))}_{\text{term}}\,)}_{\text{atomic sentence}}$$

EXAMPLES: COMPLEX SENTENCES

$$\underbrace{\underbrace{\underbrace{Sibling}_{\text{predicate}}(\underbrace{KingJohn}_{\text{term}},\underbrace{Richard}_{\text{term}})}_{\text{atomic sentence}} \Rightarrow \underbrace{\underbrace{Sibling}_{\text{predicate}}(\underbrace{Richard}_{\text{term}},\underbrace{KingJohn}_{\text{term}})}_{\text{atomic sentence}}}_{\text{complex sentence}}$$

SEMANTICS: INTENDED INTERPRETATION

- The semantics must **relate sentences** to models in order to **determine truth**. To do this, an **interpretation** is needed.
- **Interpretation specifying exactly which objects, relations and functions are referred to by the constant, predicate and function symbols.** Interpretation specifies referents for ie, it maps **constant symbols → objects, predicate symbols → relations, function symbols → functional relations**
- One possible interpretation called as the **intended interpretation**- is as follows;
  - **Richard** refers to Richard the Lionheart and **John** refers to the evil King John.
  - **Brother** refers to the brotherhood relation, that is the set of tuples of objects given in equation {(Richard the Lionheart, King John),(King John, Richard the Lionheart)}
  - **OnHead** refers to the **"on head" relation** that holds between the crown and King John; Person, King and Crown refer to the set of objects that are persons, kings and crowns.
  - **Leftleg** refers to the **"left leg" function**, that is, the mapping given in {(Richard the Lionheart, King John),(King John, Richard the Lionheart)}
- There are many other possible interpretations relating these symbols to this particular model.
- The **truth of any sentence** is determined by a **model** and an **interpretation** for the sentence symbols.

SEMANTICS: QUANTIFIERS

- Once we have a logic that allows objects, it is natural to want to express properties of entire collections of objects, instead of enumerating the objects by name. **Quantifiers** is a way for that.
- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- Quantifiers are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. These are thetypes of quantifier:

  1. **Universal Quantifier, (for all, everyone, everything)**
  2. **Existential quantifier, (for some, at least one)**
  3. **Nested Quantifiers**

UNIVERSAL QUANTIFIER

- Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing. The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.
- In general, **∀ x P is true** in a given model under a given interpretation **if P is true in all possible extended interpretations.**
- **Universal quantification (∀) Eg**
  - "**All kings are person**": ∀x King(x) ⇒ Person(x) which means For all x, if x is a king, then x is a person.Here x could be one of the following(5 extended interpretations): Richard, John, Richard's left leg, John's left leg, Crown

EXISTENTIAL QUANTIFIER

- Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for **at least one instance of something**. It is denoted by the logical operator **∃**. When it is used with a predicate variable then it is called as an **existential quantifier**.
- In general, **∃x P is true** in a given model under a given interpretation **if P is true in at least one extended interpretation that assigns x to a domain element.**
- Existential quantification (∃) Eg
  - **∃x Crown(x) ∧ OnHead(x, John)** which means there is an x such that x is a crown and x is on the John's head. Here , "Crown(crown) ∧ OnHead(crown, John)" is true

NESTED QUANTIFIERS

- It is the nesting of the same type of quantifier. One predicate is nested under the other predicate.
- Nested quantifiers Example
  - **$\forall x\ \forall y$ [Brother(x, y) $\Rightarrow$ Sibling(x, y)]**
  - **$\forall x\ \exists y$ Loves(x, y)**
  - **$\exists y\ \forall x$ Loves(x, y)**
- The **order of quantification is important**.
- **$\exists x\ \forall y$** is not the same as **$\forall y\ \exists x$.**

PROPERTIES OF QUANTIFIERS

Quantifier duality: Universal and existential quantification are logically related to each other:

- $\forall x \neg \mathrm{Love}(x,\mathrm{Saddam}) \Leftrightarrow \quad \neg\exists x\ \mathrm{Loves}(x,\mathrm{Saddam})$
- $\forall x \mathrm{Likes}(x, \mathrm{IceCream})$ is the same as $\neg\exists x \neg \mathrm{Likes}(x, \mathrm{IceCream})$
- $\exists x \mathrm{Likes}(x, \mathrm{Broccoli})$ is the same as $\neg\forall x \neg \mathrm{Likes}(x, \mathrm{Broccoli})$
- **De Morgan’s rules for quantifiers**
  - $\forall x \neg P = \neg\exists x\ P$
  - $\neg \forall x\ P = \exists x \neg P$
  - $\forall x\ P = \neg\exists x \ \neg P$
  - $\neg \forall x \neg P = \exists x\ P$
  - $\neg \exists X\ p(X) \equiv \forall X \neg p(X)$
  - $\neg \forall X\ p(X) \equiv \exists X \neg p(X)$
  - $-\forall x P(x) \wedge Q(x) \Leftrightarrow \forall x P(x) \wedge \forall x Q(x)$
  - $-\exists x\ P(x) \vee Q(x) \Leftrightarrow \exists x P(x) \ \vee \exists x Q(x)$

EQUALITY

- $\exists X\ p(X) \equiv \exists Y\ p(Y)$
- $\forall X\ q(X) \equiv \forall Y\ q(Y)$
- $\forall X\ (p(X) \land q(X)) \equiv \forall X\ p(X) \land \forall Y\ q(Y)$
- $\exists X\ (p(X) \lor q(X)) \equiv \exists X\ p(X) \lor \exists Y\ q(Y)$

PROPERTIES OF QUANTIFIERS

Quantifiers of same type commute

- ∀x ∀y is the same as ∀y ∀x
- ∃x ∃y is the same as ∃y ∃x

Quantifiers of different type do NOT commute

- ∃x ∀y is not the same as ∀y ∃x

Example

- ∃x ∀yLoves(x, y): "There is a person who loves everyone in the world"
- ∀y ∃xLoves(x, y): "Everyone in the world is loved by at least one person"

THINGS TO BE TAKEN CARE

- → **is the main connective with ∀** where as ^ **as the main connective with ∃**
- It's a mistake to use ^ as the main connective with ∀ and Using → as the main connective with ∃**.**
- Example
- **Correct:**
  - ∀x (StudiesAt(x,Koblenz))→Smart(x)): Everyone who studies at Koblenz is smart"
  - ∃x (StudiesAt(x,Landau) ^ Smart(x)): There is someone who studies at Landau and is smart"
- **Wrong:**
  - ∀x (StudiesAt(x,Koblenz) ^ Smart(x)): Everyone studies at Koblenz and is smart,"i.e., Everyone studies at Koblenz and everyone is smart"
  - ∃x (StudiesAt(x;Landau))→Smart(x)): There is someone who, if he/she studies at Landau, is smart"This is true if there is anyone not studying at Landau
  - ∀x Whale(x) ∧ Mammal(x): Says that everything in the universe is both a whale and a mammal.

EQUALITY

- **term1 = term2** is true under a given interpretation if and only if **term1 and term2 refer to the same object .**
- Can be used to **state facts** about a given function

E.g., Father(John) = Henry

- Can be **used with negation to insist that two terms are not the same object**

USING FOL:

- **Example 1:** Lipton is a tea.
- **Solution:** Here, the object is Lipton. It will be represented as **Tea(Lipton). There** is no requirement of quantifiers because the quantity is not specified in the given predicate
- **Example 2:** Every man is mortal.
- **Solution:** Here, the quantifier is the universal identifier, and the object is man. Let x be the man. Thus, it will be represented as **∀x: man(x) → mortal(x).**
- **Example 3:** All girls are beautiful.
- **Solution:** Here, we are talking about all girls. It means universal quantifier will be used. The object is girls. Let, y be the girls. Therefore, it will be represented as **∀y: girls(y) → beautiful(y).**
- **Example 4:** All that glitters is not gold.
- **Solution:** Here, we will represent gold as x. Therefore, it will be represented as **∀x: glitters(x) → ¬gold(x).**
- **Example 5:** Some boys are obedient.
- **Solution:** Here, boys are objects. The quantifier used will be existential quantifier. Let x be the boys. Thus, it will be represented as **Ǝx: boys(x) Λ obedient(x).**
- **Example 6:** Some cows are black and some cows are white.
- **Solution:** Let, x be the cows. Therefore, it will be represented as: **Ǝx(black(x) Λcows(x) Λ white(x).**

USING FOL:

1. **Not All birds fly:** $\neg\forall x\ \text{bird}(x) \rightarrow \text{fly}(x)$ **or** $\exists x\ \text{bird}(x) \wedge \neg \text{fly}(x)$

   In this question the predicate is "fly(bird)."

2. **Everybody is male or female** $\forall x.\ \text{Male}(x) \vee \text{Female}(x)$

3. **A male is not a female.** $\forall x.\ \text{Male}(x) \rightarrow \neg\text{Female}(x)$

4. **Some boys play cricket.** $\exists x\ \text{boys}(x) \wedge \text{play}(x, \text{cricket})$.

   In this question, the predicate is "play(x, y)," where x= boys, and y= game. Since there are some boys so we will use ∃**, and it will be represented as**:

5. **Not all students like both Mathematics and Science** $\neg\forall (x)\ [\ \text{student}(x) \rightarrow \text{like}(x, \text{Mathematics}) \wedge \text{like}(x, \text{Science})]$.

   In this question, the predicate is "like(x, y)," where x= student, and y= subject. Since there are not all students, so we will use ∀ **with negation, so** following representation for this:

6. **Some thing that is white is not always milk, whereas the milk is always white.**

   W hite(x): x is white. , M ilk(x): x is milk. $\exists x(\text{W hite}(x) \wedge \neg\text{M ilk}(x)) \wedge \forall x(\text{M ilk}(x) \rightarrow \text{W hite}(x))$

USING FOL

1. **The father of a person is male** In FOL objects of the domain may be denoted by functions applied to (other) objects: **∀x. Male(father(x))**
2. **Some students of DM course has cleared JEE main and the rest cleared SAT.** ClearJEE(x): x clears JEE main. ClearSAT(x): x clears SAT. **∃x[Stud(x) ^ ClearJEE(x) ^ ∀y(y ≠x→ ClearSAT(y))]**
3. **Every student loves some student. ∀ x (Student(x) ⇒ ∃ y (Student(y) ∧ Loves(x,y)))**
4. **Every student loves some other student. ∀ x (Student(x) ⇒ ∃ y (Student(y) ∧ ¬ (x = y) ∧ Loves(x,y)))**
5. **Every student takes at least one course. ∀ x (Student(x) ⇒ ∃ y (Course(y) ∧ Takes(x,y)))**
6. **Only one student failed History. ∃ x (Student(x) ∧ Failed(x, History) ∧ ∀ y (Student(y) ∧ Failed(y, History) ⇒ x = y))**
7. **No student can fool all the other students. ¬ ∃ x (Student(x) ∧ ∀ y (Student(y) ∧ ¬ (x = y) ⇒ Fools(x,y)))**
8. **Every student who walks talks. ∀x (student(x) → (walk (x) → talk (x)))**

USING FOL: KINSHIP DOMAIN

- One's mom is one's female parent: **∀x, y (Mother(x, y) ⇔ (Female(x)^Parent(x, y)))**
- One's husband is one's male spouse: **∀w,h Husband(h, w)⇔ Male(h) ΛSpouse(h, w)**
- "Brothers are siblings." **∀x, y (Brother(x, y))⇔ Sibling(x, y))**
- Parent and child are inverse relations: **∀ p, c Parent(p, c)⇔ Child(c, p)**
- A Grandparent is a Parent of one's Parent: **∀ g, c Grandparent(g,c) ⇔ ∃p Parent(g,p) ΛParent(p,c)**
- "A first cousin is a child of a parent's sibling." **∀x, y (FirstCousin(x, y)⇔ ∃p, ps (Parent(p, x)^Sibling(ps, p)^Parent(ps, y)))**

Practice:

- Male and female are disjoint categories
- A sibling is another child of one's parents

SOLUTION

- Male and female are disjoint categories:
- $\forall x\ \mathbf{Male}(x) \Leftrightarrow \neg \mathbf{Female}(x)$
- A sibling is another child of one’s parent:
- $\forall x,y\ \mathbf{Sibling}(x, y) \Leftrightarrow x \neq y \wedge \exists p\ \mathbf{Parent}(p,x) \wedge \mathbf{Parent}(p, y)$
- Definition of (full) sibling in terms of Parent
- $\forall x, y\ \mathbf{Sibling}(x, y) \Leftrightarrow (\neg(x = y) \wedge \exists m, f\ (\neg(m = f) \wedge \mathbf{Parent}(m, x) \wedge \mathbf{Parent}(f, x) \wedge \mathbf{Parent}(m, y) \wedge \mathbf{Parent}(f, y)))$

FREE AND BOUND VARIABLES

- The quantifiers interact with variables which appear in a suitable way. There are two types of variables in First-order logic which are given below:
- **Free Variable:** A variable is said to be a free variable in a formula if it occurs outside the scope of the quantifier.

Example: ∀x ∃(y)[P (x, y, z)], where z is a free variable.

- **Bound Variable:** A variable is said to be a bound variable in a formula if it occurs within the scope of the quantifier.

Example: **∀x [A (x) B(y)], here x and y are the bound variables.**

INFERENCE IN FOL

INFERENCE IN FOL

- Inference in First-Order Logic is used to deduce new facts or sentences from existing sentences.
- Two ideas:
  - convert the KB to propositional logic and use propositional inference
  - a shortcut that manipulates on first-order sentences directly

FOL INFERENCE RULES FOR QUANTIFIER:

- As propositional logic we also have inference rules in first-order logic, so following are some basic inference rules in FOL:
    1. **Universal Generalization**
    2. **Universal Instantiation**
    3. **Existential Instantiation**
    4. **Existential introduction**

1. UNIVERSAL GENERALIZATION:

- Universal generalization is a valid inference rule which states that if premise P(c) is true for any arbitrary element c in the universe of discourse, then we can have a conclusion as ∀ x P(x).
- It can be represented as: **P(c)**

  ∀ x P(x)
- This rule can be used if we want to show that every element has a similar property.
- In this rule, x must not appear as a free variable.
- **Example:** Let's represent, P(c): "**A byte contains 8 bits**", so for **∀ x P(x)** "**All bytes contain 8 bits**.", it will also be true.

2. UNIVERSAL INSTANTIATION (UI):

- In this, we can infer any sentence by substituting a ground term (a term without variables) for the variables. In short, when we create the FOPL of the given statement, we can easily infer any other statement using that FOPL . It can be represented as:

∀ x P(x)

P(c)

- **Notation:** Let, **SUBST(θ , α)=** ∀v α /Subst({v/g}, α)
- means replace All occurrences of "v" with "g" in expression "α" be the result θ, where **v** is the variable and **g** is the ground term.
- **For example**: Every man is mortal. It is represented as ∀ **x: man(x) → mortal(x).** In UI, we can infer different sentences as: **man(John) → mortal(John)** also **man(Aakash) → mortal(Aakash), etc.**
- **Universal instantiation** is also called as **universal elimination** or **UI** is a valid inference rule. It can be applied multiple times to add new sentences. The new KB is logically equivalent to the previous KB. As per UI, **we can infer any sentence obtained by substituting a ground term for the variable**. The UI rule state that we can infer any sentence P(c) by substituting a ground term c (a constant within domain x) from **∀ x P(x) for any object in the universe of discourse**.

EXAMPLES

- **Example:1.**
- IF "Every person like ice-cream"=> $\forall x\ P(x)$ so we can infer that "John likes ice-cream" => $P(c)$
- **Example: 2.**
- Let's take a famous example,
- "All kings who are greedy are Evil." So let our knowledge base contains this detail as in the form of FOL:
- **$\forall x\ king(x) \land greedy\ (x) \rightarrow Evil\ (x)$,**
- So from this information, we can infer any of the following statements using Universal Instantiation:
- **$King(John) \land Greedy\ (John) \rightarrow Evil\ (John)$,**
- **$King(Richard) \land Greedy\ (Richard) \rightarrow Evil\ (Richard)$,**
- **$King(Father(John)) \land Greedy\ (Father(John)) \rightarrow Evil\ (Father(John))$,**

3. EXISTENTIAL INSTANTIATION(EI):

- In EI, the **variable is substituted** by a **single new constant symbol**:

  ∃ x P(x)

  P(c)

- This rule states that one can infer P(c) from the formula given in the form of ∃x P(x) for a new constant symbol c.
- **Notation:** Let, the variable be **v** which is replaced by a constant symbol **k** for any sentence α. ∃v α /Subst({v/k}, α)
- The value of **k** is unique as it does not appear for any other sentence in the knowledge base. Such type of constant symbols are known as **Skolem constant.** As a result, EI is a special case of **Skolemization process.**
- **Note: UI can be applied several times to produce many sentences**, whereas **EI can be applied once, and then the existentially quantified sentences can be discarded.**
  - **For example: ∃x:steal(x, Money).** We can infer from this: **steal(Thief, Money)**
- Existential instantiation is also called as **Existential Elimination**, which is a valid inference rule in first-order logic.

EXAMPLE

- From the given sentence: **$\exists x$ Crown(x) $\land$ OnHead(x, John),**
- So we can infer: **Crown(K) $\land$ OnHead(K, John),** as long as K does not appear in the knowledge base.
- The above used K is a constant symbol, which is called **Skolem constant**.
- The Existential instantiation is a special case of **Skolemization process**.

4. EXISTENTIAL INTRODUCTION

- An existential introduction is also known as an existential generalization, which is a valid inference rule in first-order logic. This rule states that if there is some element c 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.
- It can be represented as:

P(c)

∃ x P(x)

- **Example: Let's say that,**
  "Priyanka got good marks in English."
  "Therefore, someone got good marks in English."

EXAMPLE

- For example, the following argument can be proven correct using the Universal Instantiation:**"No humans can fly. John Doe is human. Therefore John Doe can not fly."**
  First let us express this using the following notation:
  F(x) means **"x can fly."**
  H(x) means **"x is human."**
  d is a symbol representing **John Doe.**
  Then the argument is: $\forall x\, [H(x) \rightarrow \neg F(x)] \land H(d)\,] \rightarrow \neg F(d)$

1. $\forall x\, [H(x) \rightarrow \neg F(x)]$	**Hypothesis**
2. $H(d)$	**Hypothesis**
3. $H(d) \rightarrow \neg F(d)$	**Universal instantiation on 1.**
4. $\neg F(d)$	**Modus ponens on 2 and 3.**

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)

- **Discading quantifiers and applying inference rules to make KB propositional is known as propositionalization**.
- Proposition symbols are : King(John), Greedy(John), Evil(John), King(Richard), etc.
- What conclusion can you get?

PROBLEMS WITH PROPOSITIONALIZATION

- Propositionalization seems to generate lots of irrelevant sentences
- E.g., from:

  x King(x) ∧ Greedy(x) ⇒ Evil(x)

  King(John)

  ∀y Greedy(y)

  Brother(Richard,John)
- It seems obvious that Evil(John) is true, but propositionalization produces lots of facts such as Greedy(Richard) that are irrelevant
- When the KB includes a function symbol, the set of possible ground term substitutions is infinite like Father(Father(…(John)…))
- Theorem:
  - If a sentence is entailed by the original, first-order KB, then there is a proof involving just a finite subset of the propositionalized KB
  - First instantiation with constant symbols
  - Then terms with depth 1 (Father(John)) Then terms with depth 2 … Until the sentence is entailed

PROBLEMS WITH PROPOSITIONALIZATION

- Every FOL KB can be propositionalized so as to preserve entailment
- (A ground sentence is entailed by new KB iff entailed by original KB)
- Idea: propositionalize KB and query, apply resolution, return result
- Here the issue is **set of possible ground-term substitution is infinite.** This causes halting problem in Turing Machine
- ***Entailment of FOL is semidecidable-ie Algorithm says yes to every entailed sentences, but no algorithm exists that says no to every non-entailed sentence***
- **Hence propositionalization is inefficient.**

PROPOSITIONAL VS. PREDICATE LOGIC

- In propositional logic, each possible atomic fact requires a separate unique propositional symbol.
- If there are n people and m locations, representing the fact that some person moved from one location to another requires nm2 separate symbols.
- Predicate logic includes a richer ontology:

    -objects (terms)

    -properties (unary predicates on terms)

    -relations (n-ary predicates on terms)

    -functions (mappings from terms to other terms)

- Allows more flexible and compact representation of knowledge

  Move(x, y, z) for person x moved from location y to z.

UNIFICATION AND LIFTING

FIRST ORDER INFERENCE RULE: GENERALIZED MODUS PONEN

- For the inference process in FOL, **we have a single inference rule** which is called **Generalized Modus Ponens.** Generalized Modus Ponens can be summarized as, " P implies Q and P is asserted to be true, therefore Q must be True."
- According to Modus Ponens, for atomic sentences **pi, pi', q**. Where there is a substitution θ such that SUBST **(θ, pi',) = SUBST(θ, pi)**, it can be represented as:

$$\frac{p1', p2', \ldots, pn', (p1 \land p2 \land \ldots \land pn \Rightarrow q)}{\mathrm{SUBST}(\theta, q)}$$

- Here the **conclusion** is the result of **applying substitution** to **consequent q.**
- GMP is **lifted version of Modus ponens-**it raises Modus ponens from **propositional logic to FOL.**

GENERALIZED MODUS PONEN EXAMPLE

- If there is some substitution θ that makes the premise of the implication identical to sentences already in the KB, then we assert the conclusion of the implication, after applying θ. Given query **Evil(x)**?

  ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)

  King(John)

  Greedy(John) What's θ here? **θ=(x/John}**

- Instead of Greedy(John) ,we know every one is greedy

  ∀x King(x) ∧ Greedy(x) ⇒ Evil(x)

  King(John)

  ∀y Greedy(y) What's θ here? **θ=(x/John, y/john}**

- **Example: We will use this rule for Kings are evil, so we will find some x such that x is king, and x is greedy so we can infer that x is evil.**
- Here let say, p1' is king(John)   p1 is king(x)
- p2' is Greedy(y)   p2 is Greedy(x)
- θ is {x/John, y/John}   q is evil(x)
- SUBST(θ,q) is **Evil(John)**.

UNIFICATION AND LIFTING

- Unification is a process of making **two different logical atomic expressions identical** by **finding a substitution**. Unification depends on the substitution process and is the **key component of First-order inference algorithms**.
- Unification is the process used by the lifted inference rules to find substituents that could give identical but different logical expressions. It means the meaning of the sentence should not be changed, but it should be expressed in multiple ways. The **UNIFY** algorithm in unification takes two sentences as input and then returns a unifier if one exists:

  UNIFY(p,q)= θ **where SUBST(** θ **, p) = SUBST(** θ, **q)**

- Unification is a lifted version of Modus Ponen as it uplifts the Modus Ponens from ground propositions to FOPL. It takes two literals as input and makes them identical using substitution.
- **Unification** is needed for **resolution** to work.

UNIFICATION

- **Unifier:** a substitution that makes two clauses resolvable
- **E.g.** Let's say there are two different expressions, **P(x, y), and P(a, f(z))**.
- In this example, we need t**o make both above statements identical to each other.** For this, we will **perform the substitution.**

  P(x, y)......... (i)
  P(a, f(z))......... (ii)

- Substitute **x** with **a**, and **y** with **f(z)** in the first expression, and it will be represented as **a/x** or **x/a** and **f(z)/y** or **y/f(z).** There are different interpretation for this notation. In **Russel and Norwig** the representation is **x/a, y/f(z)**
- With both the substitutions, the first expression will be identical to the second expression and the substitution set will be: **[x/a, y/f(z)]**

UNIFICATION EXAMPLE

- **Given KB:**
  - **Knows(John, Jane)**
  - **Knows(y, Bill)**
  - **Knows(y, Mother(y))**
  - **Knows(x, Elizabeth)**

p	**q**	**θ**
UNIFY(Knows(John, x),	Knows(John, Jane))	{x/Jane}
UNIFY(Knows(John, x)	Knows(y, Bill))	{x/Bill, y/John}
UNIFY(Knows(John, x)	Knows(y, Mother(y)))	{y/John, x/Mother(John)}
UNIFY(Knows(John, x)	Knows(x, Elizabeth))	fail

- **a query Knows(John, x)** means **Every one knows John** The question is- **Whom does John knows? Ie like Does John know Jane? Does John know Bill?**
- **Answer to this query can be found by finding all sentences in KB that unifies Knows(John,x).** The UNIFY algorithm will search all the related sentences in the knowledge base, which could unify with **Knows(John,x)**.
- UNIFY (Knows(John, x), Knows(John, Jane))≡{x/Jane}
- UNIFY (Knows{John,x}, Knows{y, Bill})≡{x/Bill, y/John}
- UNIFY (Knows{John, x}, Knows{y, Mother(y)})≡{ y/John, x/Mother(John)}

UNIFICATION EXAMPLE

- **UNIFY (Knows{John,x}, Knows{x, Elizabeth})≡fails.** The last one failed because we have used the **same variable for two persons at the same time** ie **x cannot take both values John and Elizabeth.**
- It can be avoided by standardizing apart ie renaming
- UNIFY (Knows{John,x}, Knows{x, Elizabeth})≡ UNIFY (Knows{John,x}, Knows{x17, Elizabeth})≡ { x/Elizabeth, x17/ John }
- Consider the following two unifications
    - **UNIFY (Knows (John, x), Knows (y, z)) ={ y/John, x/ z }**
    - **UNIFY (Knows (John, x), Knows (y, z)) ={ y/John, x/John , z/John }**
- We say the first unifier is more general than the second . **It places fewer restrictions on the values of variables.** For every unifiable pairs of expressions, there is a single **most generalized unifier (MGU)** – E.g., the former unifier, { y/John, x/ z}, shown above

UNIFICATION RULES

- **Following are some basic conditions for unification:**
  1. **Predicate symbol must be same, atoms or expression with different predicate symbol can never be unified.**
  2. **Number of Arguments in both expressions must be identical.**
  3. **Unification will fail if there are two similar variables present in the same expression.**
- Rules for substitutions:
  1. **Can replace a variable by a constant.**
  2. **Can replace a variable by a variable.**
  3. **Can replace a variable by a function expression, as long as the function expression does not contain the variable.**

UNIFICATION EXAMPLE

1. P(x) and P(y): substitution = (x/y)
2. P(x,x) and P(y,z): (z/y)(y/x)
3. P(x,f(y)) and P(Joe,z): (Joe/x, f(y)/z)
4. P(f(x)) and P(x): can’t do it!
5. P(x) $\vee$ Q(Jane) and P(Bill) $\vee$ Q(y): (Bill/x, Jane/y)

MOST GENERAL UNIFIER

- In cases **where there is more than one substitution choose the one that makes the least commitment (most general) about the bindings. The substitution variables are called Most General Unifier or MGU.**

= {y / John, x / z} not {y / John, x / John, z / John} not {y / John, x / z, z / Freda} …

- For each pair of atomic sentences, give the most general unifier if it exists.

1. Find the MGU of {p(f(a), g(Y))} and p(X, X)}

- Sol: **S_0** =>{} Here, **Ψ_1** = **p(f(a), g(Y))**, and **Ψ_2** = **p(X, X)**
  SUBST θ= {X/ f(a) }
  S1 => **Ψ1** = **p(f(a), g(Y))**, and **Ψ2** = **p(f(a), f(a))**
  SUBST θ= {g(y)/f(a) }, **Unification failed**.
- Unification is not possible for these expressions.

MGU EXAMPLES

2. Find the MGU of {p(b, X, f(g(Z))) and p(Z, f(Y), f(Y))}

- Here, Ψ_1 = **p(b, X, f(g(Z)))** , and **Ψ2** = **p(Z, f(Y), f(Y))**
  S0 => { **p(b, X, f(g(Z))); p(Z, f(Y), f(Y))**}
  SUBST θ={Z/b}

- **S1** => { **p(b, X, f(g(b))); p(b, f(Y), f(Y))**}
  SUBST θ={X/f(Y)}

- **S2** => { **p(b, f(Y), f(g(b))); p(b, f(Y), f(Y))**}
  SUBST θ= {Y/g(b)}

- **S2** => { **p(b, f(g(b)), f(g(b)); p(b, f(g(b)), f(g(b))**} **Unified Successfully.**
  And Unifier = { Z/b, X/f(Y) , Y/g(b) }.

3. UNIFY(knows(Richard, x), knows(Richard, John))

- Here, **Ψ1** = **knows(Richard, x)**, and Ψ_2 = **knows(Richard, John)**
  S0 => **{ knows(Richard, x); knows(Richard, John)**}
  SUBST θ= {x/John}
  S1 => **{ knows(Richard, John); knows(Richard, John)}, Successfully Unified.**
  Unifier: {x/John}.

MGU EXAMPLES

4. Find the MGU of {p (X, X), and p (Z, f(Z))}

- Here, Ψ_1 = {p (X, X), and Ψ2 = p (Z, f(Z))
  S0 => {p (X, X), p (Z, f(Z))}
  SUBST θ= {X/Z}
  S1 => {p (Z, Z), p (Z, f(Z))}
  SUBST θ= {f(Z) /Z}, Unification Failed.
- **Hence, unification is not possible for these expressions.**

5. Find the MGU of UNIFY(prime (11), prime(y))

- Here, Ψ1 = {prime(11) , and Ψ2 = prime(y)}
  S0 => {prime(11) , prime(y)}
  SUBST θ= {y/11}
- S1 => {prime(11) , prime(11)} , **Successfully unified.**
  Unifier: {y/11}.

MGU EXAMPLES

6. Find the MGU of Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x)}

- Here, **Ψ_1** = **Q(a, g(x, a), f(y))**, and **Ψ2** = **Q(a, g(f(b), a), x)**
  S0 => {**Q(a, g(x, a), f(y)); Q(a, g(f(b), a), x)**}
  SUBST θ= {x/f(b)}
  S1 => **{Q(a, g(f(b), a), f(y)); Q(a, g(f(b), a), f(b))}**
- **SUBST θ= {y/b}**
  S1 => **{Q(a, g(f(b), a), f(b)); Q(a, g(f(b), a), f(b))}**, **Successfully Unified.**
- **Unifier: [a/a, x/f(b), y/b].**
- **CHALLENGES**

1. **P(A, B, B), P(x, y, z)**
2. **Q(y, G(A, B)), Q(G(x, x), y)**
3. **Older(Father(y), y), Older(Father(x), John)**
4. **Knows(Father(y), y), Knows(x, x)**

UNIFICATION ALGORITHM

```
mgu := {a=b}; stop := false;
 WHILE (not(stop) AND mgu contains s=t)
 Case1: t is a variable, s is not a variable:
     Replace s = t by t = s in mgu
Case2: s is a variable, t is the SAME variable:
Delete s=t from mgu
Case3: s is a variable, t is not a variable and contains s:
 stop := true
 Case4: s is a variable, t is not identical to nor contains s:
 Replace all occurrences of s in mgu by t
 Case5: s is of the form f(s1,…,sn), t of g(t1,…,tm):
 If f not equal to g or m not equal to n then stop := true
 Else replace s=t in mgu by s1 = t1,…,sn=tn
```

EXAMPLE

1. What is the m.g.u. of: p(f(y),w,g(z,y)) = p(x,x,g(z,A))

MGU:

x/f(A), w/f(A) , y/A

Result:

p(f(A),f(A),g(z,A))

2…What is the m.g.u. of: f(x,g(f(a),u)) = f(g(u,v),x)

MGU: **x/g(f(a),f(a)), u/f(a), v/f(a)**

Result: f(g(f(a),f(a)),g(f(a), f(a)))

Case 5: f(y) = x, w = x, g(z,y) = g(z,A)
Case 1: x = f(y), w = x, g(z,y) = g(z,A)
Case 4: x = f(y), w = f(y) , g(z,y) = g(z,A)
Case 5: x = f(y), w = f(y) , z = z, y = A
Case 2: x = f(y), w = f(y) , y = A
Case 4: x = f(A), w = f(A) , y = A

Init: f(x,g(f(a),u)) = f(g(u,v),x)
Case 5: x = g(u,v), g(f(a),u) = x
Case 4: x = g(u,v), g(f(a),u) = g(u,v)
Case 5: x = g(u,v), f(a) = u, u = v
Case 1: x = g(u,v), u = f(a), u = v
Case 4: x = g(f(a),v), u = f(a), f(a) = v
Case 1: x = g(f(a),v), u = f(a), v = f(a)
Case 4: x = g(f(a),f(a)), u = f(a), v = f(a)

EXAMPLE

- What is the m.g.u. of: p(A,x,f(g(y))) = p(z,f(z),f(A))

Init: p(A,x,f(g(y))) = p(z,f(z),f(A))

Case 5: A = z, x = f(z), f(g(y)) = f(A)

Case 1: z = A, x = f(z), f(g(y)) = f(A)

Case 4: z = A, x = f(A), f(g(y)) = f(A)

Case 5: z = A, x = f(A), g(y) = A

Case 5: stop :failure

EXAMPLE

- What is the m.g.u. of: q(x,x) = q(y,f(y))

Init: q(x,x) = q(y,f(y))

Case 5: x = y, x = f(y)

Case 4: x = y, y = f(y)

Case 3: stop := true

FIRST-ORDER DEDUCTION: INFERENCE IN FOL

- **Inference:** deriving sentences from other sentences. Want to be able to draw logically sound conclusions from a knowledge-base expressed in first-order logic. Several styles of inference proving:
  1. **Forward chaining**
  2. **Backward chaining**
  3. **Resolution proofs with refutation**
- **Entailment:** necessary truth of one sentence given another
- Properties of inference procedures:
  - **Soundness:** If A |- B then A |= B derivations produce only entailed sentences
  - **Completeness:** If A |= B then A |- B derivations can produce all entailed sentences
- Forward and backward chaining are **sound** and can be **reasonably efficient** but **are incomplete**.
- Resolution is sound and complete for FOPC but can be very inefficient.

INFERENCE RULES FOR QUANTIFIERS

Universal Elimination: $\forall$v a |- SUBST({v/g},a) for any sentence, a, variable, v, and ground term, g

- **$\forall$x Loves(x, FOPC) |- Loves(Ray, FOPC)**

Existential Elimination: $\exists$v a |- SUBST({v/k},a) for any sentence, a, variable, v, and constant symbol, k, that doesn't occur elsewhere in the KB (Skolem constant)

- **$\exists$x (Owns(Mary,x) $\wedge$ Cat(x)) |- Owns(Mary,MarysCat) $\wedge$ Cat(MarysCat)**

Existential Introduction: a |- $\exists$v SUBST({g/v},a) for any sentence, a, variable, v, that does not occur in a,and ground term, g, that does occur in a

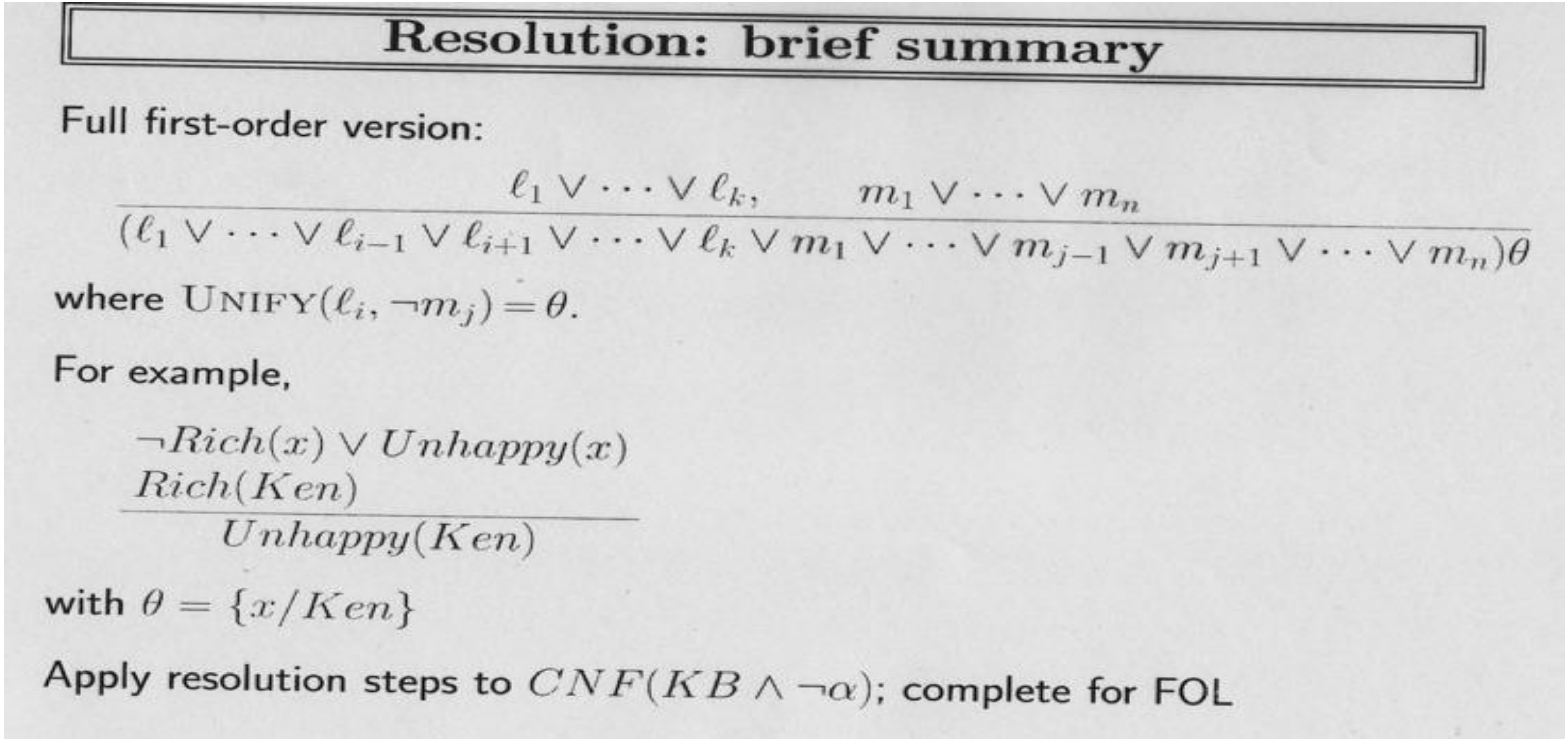
- **Loves(Ray, FOPC) |- $\exists$x Loves(x, FOPC)**

RESOLUTION INFERENCE RULE IN FOL

- **The resolution inference rule:** The resolution rule for first-order logic is simply a lifted version of the propositional rule. **Resolution can resolve two clauses if they contain complementary literals,** which are assumed to be standardized apart so that they share no variables.
- Where $\mathbf{l_i}$ and $\mathbf{m_j}$ are complementary literals.

$$\frac{l_1 \vee \ldots\ldots \vee l_k, \qquad m_1 \vee \ldots\ldots \vee m_n}{SUBST(\theta, l_1 \vee \ldots\ldots \vee l_{i-1} \vee l_{i+1} \vee \ldots \vee l_k \vee m_1 \vee \ldots\ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots\ldots \vee m_n)}$$

- This rule is also called the **binary resolution rule** because it only resolves exactly two literals.
- Resolution is simply a **generalization of modus ponens.** As with modus ponens, chains of resolution steps can be used to construct proofs.



Resolution: brief summary

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

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

The Resolution Rule

Resolution relies on the following rule:

$$\boxed{\frac{\neg\alpha \Rightarrow \beta,\ \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma} \qquad \text{Resolution rule}}$$

equivalently,

$$\boxed{\frac{\alpha \vee \beta,\ \neg\beta \vee \gamma}{\alpha \vee \gamma} \qquad \text{Resolution rule}}$$

Applying the resolution rule:

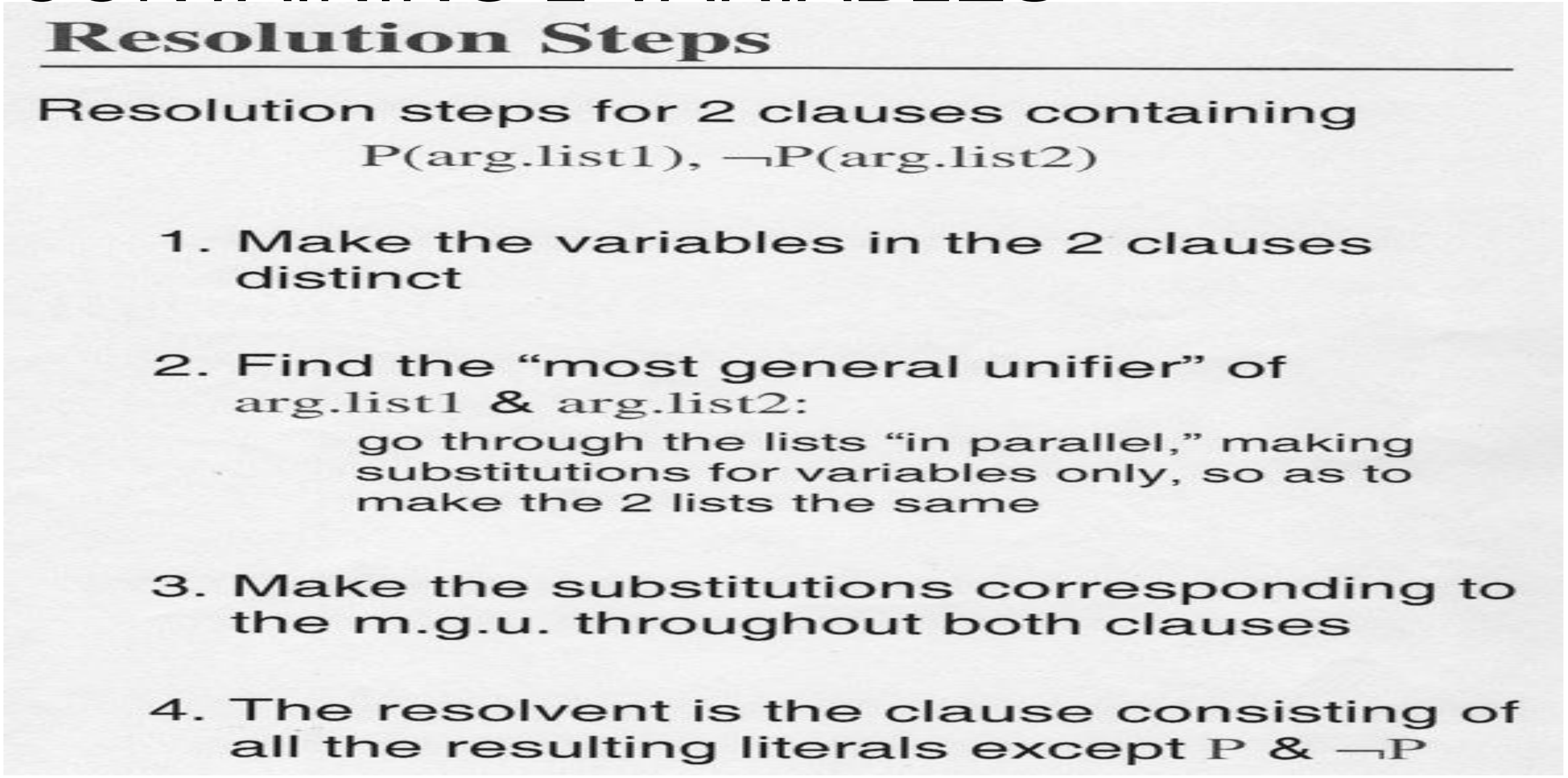
1. Find two sentences that contain the same literal, once in its positive form & once in its negative form:

CNF sentences $\longrightarrow$ $\boxed{\text{summer} \vee \text{winter},\ \neg\text{winter} \vee \text{cold}}$

2. Use the resolution rule to eliminate the literal from both sentences

$\longrightarrow$ $\boxed{\text{summer} \vee \text{cold}}$

RESOLUTION STEPS FOR 2 CLAUSES CONTAINING 2 VARIABLES



Resolution Steps

Resolution steps for 2 clauses containing
P(arg.list1), ¬P(arg.list2)

1. Make the variables in the 2 clauses distinct
2. Find the "most general unifier" of arg.list1 & arg.list2:
   go through the lists "in parallel," making substitutions for variables only, so as to make the 2 lists the same
3. Make the substitutions corresponding to the m.g.u. throughout both clauses
4. The resolvent is the clause consisting of all the resulting literals except P & ¬P

EXAMPLE FOR RESOLUTION INFERENCE RULE

EXAMPLE 1

- We can resolve two clauses **[Animal (g(x) V Loves (f(x), x)]** and **[¬ Loves(a, b) V ¬Kills(a, b)]**, **w**here two complimentary literals are: **Loves (f(x), x) and ¬ Loves (a, b)**
- These literals can be unified with unifier **θ= [a/f(x), and b/x]** , and it will generate a resolvent clause:
- **[Animal (g(x) V ¬ Kills(f(x), x)].**

EXAMPLE 2

1) ∀x,y(Parent(x,y) ∧ Male(x) ⇒ Father(x,y))

2) Parent(Tom,John)

3) Male(Tom)

θ={x/Tom, y/John)

4) Father(Tom,John)

RESOLUTION IN FOL

- Resolution means proving a conclusion S from given premises expressed in FOL.
- Resolution is a theorem proving technique that proceeds by building refutation proofs, i.e., proofs by contradictions. Unification is a key concept in proofs by resolutions.
- In order to utilize **generalized Modus Ponens**, all sentences in the KB must be in the form of **Horn sentences,** disjunctions with at most one non-negated literal: $\forall v1,v2,...vn\ p1 \wedge p2 \wedge ... \wedge pm \Rightarrow \theta$

RESOLUTION STEPS

1. Conversion of facts into first-order logic.
2. Convert all sentences to CNF(clausal form)
3. Negate conclusion S & convert result to CNF
4. Add negated conclusion S to the premise clauses
5. Draw resolution graph (unification), to better understand all the above steps, we will take an example in which we will apply resolution.Repeat until contradiction or no progress is made:

   a. Select 2 clauses (call them parent clauses)

   b. Resolve them together, performing all required unifications

   c. If resolvent is the empty clause, a contradiction has been found (i.e., S follows from the premises)

   d. If not, add resolvent to the premises

CONVERSION TO CLAUSAL FORM

1. Eliminate implications and biconditionals by rewriting them.
2. Move $\neg$ inward to only be a part of literals by using deMorgan's laws and quantifier rules.
3. **Skolemize:** Remove existential quantifiers by replacing each existentially quantified variable with a Skolem constant or Skolem function as appropriate. 1)**$\exists$ x P(x)-->P(E)**
 ii)$\forall$ xy $\exists$ x P(x,y)--> P(E(y),y) Eg: $\exists$ x (P(x)) $\rightarrow$ P(Jerry), $\exists$ x $\forall$ y (P(x,y)) $\rightarrow$ $\forall$ y P(Fred,y)
4. Standardize variables by renaming to avoid use of the same variable name by two different quantifiers.
   $\forall$**x** P(x) $\vee$ $\exists$x P(x) -> $\forall$**x1** P(x1) $\vee$ $\exists$**x2** P(x2)
5. Move quantifiers left while maintaining order. Renaming above guarantees this is a truth-preserving transformation.
   $\forall$**x1** P(x1) $\vee$ $\forall$ **x2** P(x2) -> $\forall$**x1** $\forall$ **x2** (P(x1) $\vee$ P(x2))
6. Distribute $\wedge$ over $\vee$ to convert to conjunctions of clauses.
7. Eliminate conjunctions. Convert clauses to implications if desired for readability ($\neg$a $\vee$ $\neg$b $\vee$ c $\vee$ d) -> a $\wedge$ b $\Rightarrow$ c $\vee$ d
8. Rename variables if necessary
9. Drop universal quantifiers.

SKOLEMIZATION

1. If an existential variable is not within the scope of any universally quantified variable, then replace every instance of the variable with the same unique constant that does not appear anywhere else.

   $\exists x\ (P(x) \wedge Q(x))$ -> $P(C1) \wedge Q(C1)$

2. If it is within the scope of n universally quantified variables, then replace it with a unique n-ary function over these universally quantified variables.

   $\forall x1 \exists x2\ (P(x1) \vee P(x2))$ -> $\forall x1(P(x1) \vee P(f1(x1)))$

   $\forall x(\text{Person}(x) \Rightarrow \exists y(\text{Heart}(y) \wedge \text{Has}(x,y)))$ ->

   $\forall x(\text{Person}(x) \Rightarrow \text{Heart}(\text{HeartOf}(x)) \wedge \text{Has}(x,\text{HeartOf}(x)))$

   - Afterwards, all variables can be assumed to be universally quantified, so remove all quantifiers

$$\forall x : \exists y : father(y,x) \text{ becomes } \forall x : father(S1(x),x)$$

$$\exists x : President(x) \text{ becomes } President(S2)$$

EXAMPLE 1 CLAUSAL CONVERSION

Sentence:

$$\begin{aligned}\forall x : brick(x) \rightarrow \quad & ((\exists y : on(x,y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \\ & \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x,y)))\end{aligned}$$

Substitute $\neg E_1 \vee E_2$ for $E_1 \rightarrow E_2$

$$\begin{aligned}\forall x : brick(x) \rightarrow \quad & ((\exists y : on(x,y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \\ & \wedge (\forall y : \neg brick(y) \rightarrow \neg equal(x,y))\end{aligned}$$

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad & ((\exists y : on(x,y) \wedge \neg pyramid(y)) \\ & \wedge (\neg \exists y : on(x,y) \wedge on(y,x)) \\ & \wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x,y))\end{aligned}$$

Move negations down to the atomic formulas

$$\begin{aligned}&\forall x : \neg brick(x) \vee \\ &((\exists y : on(x,y) \wedge \neg pyramid(y)) \\ &\wedge (\neg\exists y : \; on(x,y) \wedge on(y,x)) \\ &\wedge (\forall y : \neg(\neg brick(y)) \vee \neg equal(x,y)))\end{aligned}$$

Eliminate Existential Quantifiers:

Skolemization

$$\begin{aligned}\forall x : \neg brick(x) \vee \quad &(on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ &\wedge (\forall y : (\neg on(x,y) \vee \neg on(y,x))) \\ &\wedge (\forall y : (brick(y) \vee \neg equal(x,y)))\end{aligned}$$

Rename variables as necessary

no two variables of the same name.

$$\forall x : \neg brick(x) \vee \quad (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ \wedge (\forall y : (\neg on(x, y) \vee \neg on(y, x))) \\ \wedge (\forall z : (brick(z) \vee \neg equal(x, z)))$$

Move the universal quantifiers to the left

$$\forall x \forall y \forall z : \neg brick(x) \vee \quad (on(x, S1(x)) \wedge \neg pyramid(S1(x))) \\ \wedge (\neg on(x, y) \vee \neg on(y, x)) \\ \wedge (brick(z) \vee \neg equal(x, z))$$

Move disjunctions down to the literals

$$\begin{aligned}\forall x \forall y \forall z :\ & (\neg brick(x) \lor on(x, S1(x))) \\ & \land (\neg brick(x) \lor \neg pyramid(S1(x))) \\ & \land (\neg brick(x) \lor \neg on(x,y) \lor \neg on(y,x)) \\ & \land (\neg brick(x) \lor brick(z) \lor \neg equal(x,z))\end{aligned}$$

Eliminate the conjunctions

$$\begin{aligned}&\forall x : \neg brick(x) \lor on(x, S1(x)) \\ &\forall x : \neg brick(x) \lor \neg pyramid(S1(x)) \\ &\forall x \forall y : \neg brick(x) \lor \neg on(x,y) \lor \neg on(y,x) \\ &\forall x \forall z : \neg brick(x) \lor brick(z) \lor \neg equal(x,z)\end{aligned}$$

Rename all variables, as necessary, so no two have the same name

$$\forall x : \neg brick(x) \vee on(x, S1(x))$$
$$\forall w : \neg brick(w) \vee \neg pyramid(S1(w))$$
$$\forall u \forall y : \neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$
$$\forall v \forall z : \neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

Eliminate the universal quantifiers

$$\neg brick(x) \vee on(x, S1(x))$$
$$\neg brick(w) \vee \neg pyramid(S1(w))$$
$$\neg brick(u) \vee \neg on(u, y) \vee \neg on(y, u)$$
$$\neg brick(v) \vee brick(z) \vee \neg equal(v, z)$$

EXAMPLE-2 CLAUSAL CONVERSION

$$\begin{aligned}&\forall x((\mathrm{Prof}(x) \lor \mathrm{Student}(x)) \Longrightarrow (\exists y(\mathrm{Class}(y) \land \mathrm{Has}(x,y)) \land \\ &\qquad \exists y(\mathrm{Book}(y) \land \mathrm{Has}(x,y))))\end{aligned}$$

$$\begin{aligned}&\forall x(\neg(\mathrm{Prof}(x) \lor \mathrm{Student}(x)) \lor (\exists y(\mathrm{Class}(y) \land \mathrm{Has}(x,y)) \land \\ &\qquad \exists y(\mathrm{Book}(y) \land \mathrm{Has}(x,y))))\end{aligned}$$

$$\begin{aligned}&\forall x((\neg\mathrm{Prof}(x) \land \neg\mathrm{Student}(x)) \lor (\exists y(\mathrm{Class}(y) \land \mathrm{Has}(x,y)) \land \\ &\qquad \exists y(\mathrm{Book}(y) \land \mathrm{Has}(x,y))))\end{aligned}$$

$$\begin{aligned}&\forall x((\neg\mathrm{Prof}(x) \land \neg\mathrm{Student}(x)) \lor (\exists y(\mathrm{Class}(y) \land \mathrm{Has}(x,y)) \land \\ &\qquad \exists z(\mathrm{Book}(z) \land \mathrm{Has}(x,z))))\end{aligned}$$

$$\begin{aligned}&\forall x\exists y\exists z((\neg\mathrm{Prof}(x)\land\neg\mathrm{Student}(x))\lor ((\mathrm{Class}(y) \land \mathrm{Has}(x,y)) \land \\ &\qquad (\mathrm{Book}(z) \land \mathrm{Has}(x,z))))\end{aligned}$$

$$\begin{aligned}&(\neg\mathrm{Prof}(x)\land\neg\mathrm{Student}(x))\lor (\mathrm{Class}(f(x)) \land \mathrm{Has}(x,f(x)) \land \\ &\qquad \mathrm{Book}(g(x)) \land \mathrm{Has}(x,g(x))))\end{aligned}$$

$$\begin{aligned}&(\neg\mathrm{Prof}(x) \lor \mathrm{Class}(f(x))) \land \\ &(\neg\mathrm{Prof}(x) \lor \mathrm{Has}(x,f(x))) \land \\ &(\neg\mathrm{Prof}(x) \lor \mathrm{Book}(g(x))) \land \\ &(\neg\mathrm{Prof}(x) \lor \mathrm{Has}(x,g(x))) \land \\ &(\neg\mathrm{Student}(x) \lor \mathrm{Class}(f(x))) \land \\ &(\neg\mathrm{Student}(x) \lor \mathrm{Has}(x,f(x))) \land \\ &(\neg\mathrm{Student}(x) \lor \mathrm{Book}(g(x))) \land \\ &(\neg\mathrm{Student}(x) \lor \mathrm{Has}(x,g(x))))\end{aligned}$$

EXAMPLE-3 OF CONVERSION

- Convert: “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. Remove ⇒ from expression:

- $\forall X\ [\ \neg\forall Y\ \neg\text{Animal}(Y)\ \mathbf{\vee}\ \text{Loves}(X,Y)\ ]\ \mathbf{\vee}\ [\exists Y\ \text{Loves}(Y,X)\ ]$

2. Reduce scope of negation:

- $\forall X\ [\ \exists Y\ \text{Animal}(Y) \wedge \neg\text{Loves}(X,Y)\ ]\ \mathbf{\vee}\ [\exists Y\ \text{Loves}(Y,X)\ ]$

3. Rename variables:

- $\forall X\ [\ \exists Y\ \text{Animal}(Y) \wedge \neg\text{Loves}(X,Y)\ ]\ \mathbf{\vee}\ [\exists Z\ \text{Loves}(Z,X)\ ]$

4. Eliminate $\exists$ using Skolemization:

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

5. Drop universal quantifiers:

- [Animal(F(X)) ^ ¬Loves(X, F(X))] **V** [Loves(G(X), X)]

6. Convert to conjuction of disjuncts:

- [Animal(F(X)) **V** Loves(G(X), X)] ^ [¬Loves(X, F(X)) **V** Loves(G(X), X)]

7. Separate into clauses:

- [Animal(F(X)) **V**Loves(G(X), X)]
- [¬Loves(X, F(X)) **V** Loves(G(X), X)]

8. Rename variables (again) in different clauses:

- [Animal(F(X)) **V** Loves(G(X) ,X)]
- [¬Loves(W, F(W)) **V** Loves(G(W), W)]

CHALLENGE

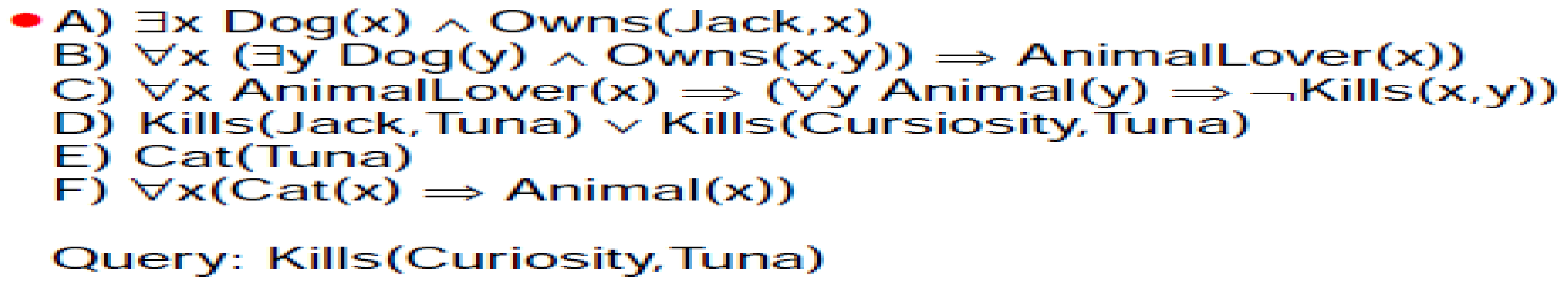
- **John likes all kind of food.**
- **Apple and vegetable are food**
- **Anything anyone eats and not killed is food.**
- **Anil eats peanuts and still alive**
- **Harry eats everything that Anil eats.**
  Prove by resolution that:
- **John likes peanuts.**

SAMPLE RESOLUTION PROOF

1. Jack owns a dog.
2. Every dog owner is an animal lover.
3. No animal lover kills an animal.
4. Either Jack or Curiosity killed Tuna the cat.

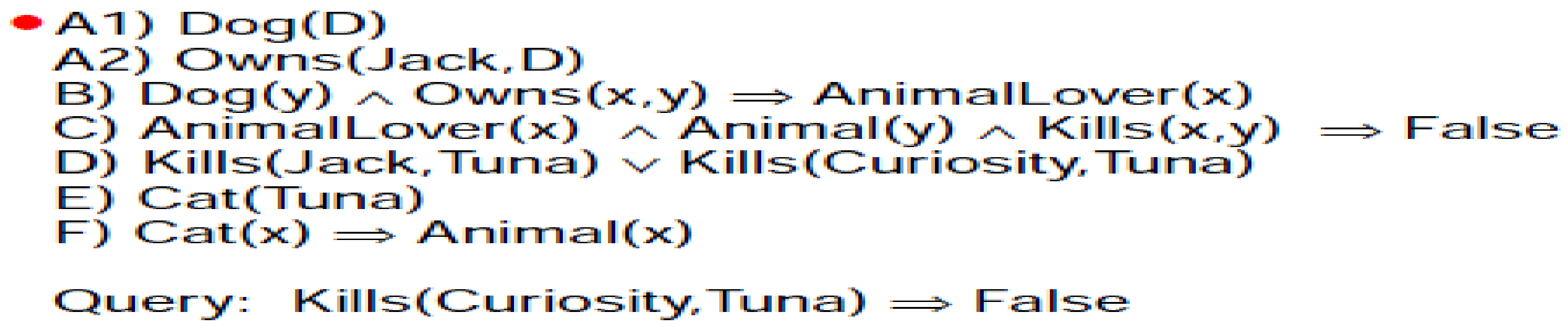
Did Curiosity kill the cat?

CLAUSAL FORM AND SKOLEMISATION



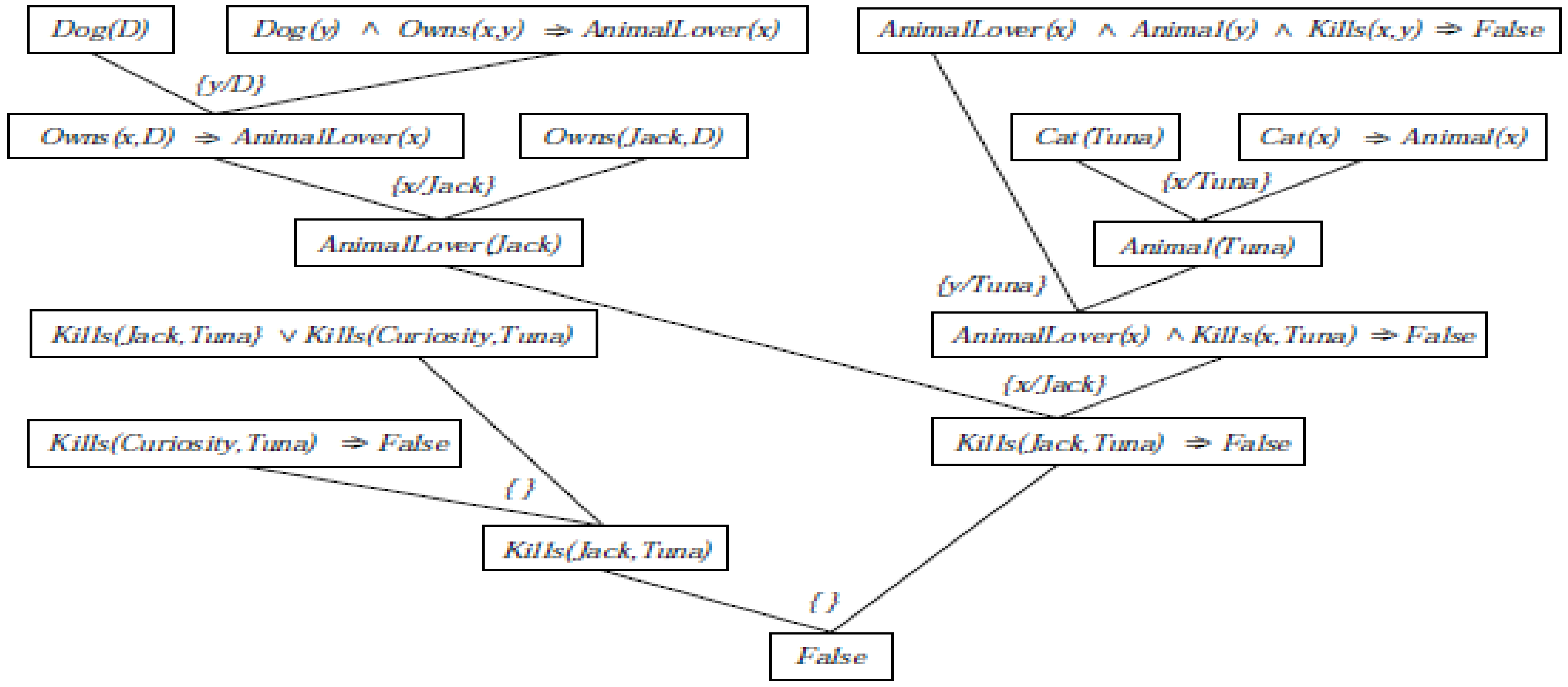
- A) $\exists x\ Dog(x) \wedge Owns(Jack,x)$
  B) $\forall x\ (\exists y\ Dog(y) \wedge Owns(x,y)) \Rightarrow AnimalLover(x))$
  C) $\forall x\ AnimalLover(x) \Rightarrow (\forall y\ Animal(y) \Rightarrow \neg Kills(x,y))$
  D) $Kills(Jack,Tuna) \vee Kills(Cursiosity,Tuna)$
  E) $Cat(Tuna)$
  F) $\forall x(Cat(x) \Rightarrow Animal(x))$

  Query: $Kills(Curiosity,Tuna)$



- A1) $Dog(D)$
  A2) $Owns(Jack,D)$
  B) $Dog(y) \wedge Owns(x,y) \Rightarrow AnimalLover(x)$
  C) $AnimalLover(x) \wedge Animal(y) \wedge Kills(x,y) \Rightarrow False$
  D) $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$
  E) $Cat(Tuna)$
  F) $Cat(x) \Rightarrow Animal(x)$

  Query: $Kills(Curiosity,Tuna) \Rightarrow False$


Dog(D)
Dog(y) ∧ Owns(x,y) ⇒ AnimalLover(x)
{y/D}
Owns(x,D) ⇒ AnimalLover(x)
Owns(Jack,D)
{x/Jack}
AnimalLover(Jack)
AnimalLover(x) ∧ Animal(y) ∧ Kills(x,y) ⇒ False
Cat(Tuna)
Cat(x) ⇒ Animal(x)
{x/Tuna}
Animal(Tuna)
{y/Tuna}
AnimalLover(x) ∧ Kills(x,Tuna) ⇒ False
{x/Jack}
Kills(Jack,Tuna) ∨ Kills(Curiosity,Tuna)
Kills(Curiosity,Tuna) ⇒ False
{ }
Kills(Jack,Tuna)
Kills(Jack,Tuna) ⇒ False
{ }
False

EXAMPLE 1:

1. If something is intelligent, it has common sense
2. Deep Blue does not have common sense

- Prove that Deep Blue is not intelligent
- Here MGU: x/D

1. $\forall x. I(x) \Rightarrow H(x)$
2. $\neg H(D)$

Conclusion: $\neg I(D)$
Denial: C3: $I(D)$

$\longrightarrow$

C1: $\neg I(x) \lor H(x)$
C2: $\neg H(D)$

CNF

A resolution proof of $\neg I(D)$:

$$\begin{tikzcd}
\boxed{\neg I(x) \lor H(x)} \arrow[dr, "\{x/D\}"'] & & \boxed{\neg H(D)} \arrow[dl] & \\
& \boxed{\neg I(D)} \arrow[dr] & & \boxed{I(D)} \arrow[dl] \\
& & \square &
\end{tikzcd}$$

Proof also written as:

C4: $\neg I(D)$ r[C1b, C2] $\xleftarrow{\text{2nd literal, 1st clause}}$

C5: $\square$ r[C3, C4]

Example 2:

Premises:

Mother(Lulu, Fifi)

Alive(Lulu)

$\forall x\ \forall y.\mathrm{Mother}(x,y) \Longrightarrow \mathrm{Parent}(x,y)$

$\forall x\ \forall y.(\mathrm{Parent}(x,y) \land \mathrm{Alive}(x)) \Longrightarrow \mathrm{Older}(x,y)$

Prove:

Older(Lulu, Fifi)

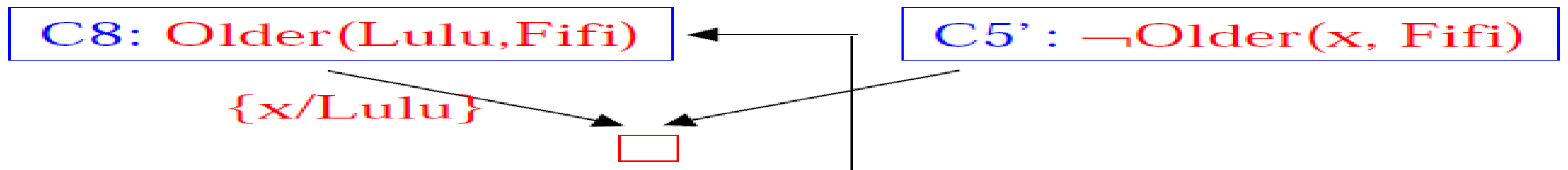
Denial:

$\neg$Older(Lulu, Fifi)

$$\begin{tikzcd}
\boxed{\mathrm{Mother(Lulu,Fifi)}} \arrow[d] & & \boxed{\neg\mathrm{Mother}(x,y) \lor \mathrm{Parent}(x,y)} \arrow[dll, "\{x/\mathrm{Lulu},y/\mathrm{Fifi}\}"] & \\
\boxed{\mathrm{Parent(Lulu,Fifi)}} \arrow[dr] & & \boxed{\neg\mathrm{Parent}(x,y) \lor \neg\mathrm{Alive}(x) \lor \mathrm{Older}(x,y)} \arrow[dl, "\{x/\mathrm{Lulu},y/\mathrm{Fifi}\}"] & \\
& \boxed{\neg\mathrm{Alive(Lulu)} \lor \mathrm{Older(Lulu,\ Fifi)}} \arrow[dr] & & \boxed{\mathrm{Alive(Lulu)}} \arrow[dl] \\
\boxed{\neg\mathrm{Older(Lulu,\ Fifi)}} \arrow[dr] & & \boxed{\mathrm{Older(Lulu,\ Fifi)}} \arrow[dl] & \\
& \square & &
\end{tikzcd}$$

Example 3:

- Suppose the desired conclusion had been "Something is older than Fifi"
  $\exists x.\mathrm{Older}(x, \mathrm{Fifi})$
- Denial:
  $\neg\exists x.\mathrm{Older}(x, \mathrm{Fifi})$
  also written as: $\forall x.\neg\mathrm{Older}(x, \mathrm{Fifi})$
  in clause form: $\neg\mathrm{Older}(x, \mathrm{Fifi})$
- Last proof step would have been



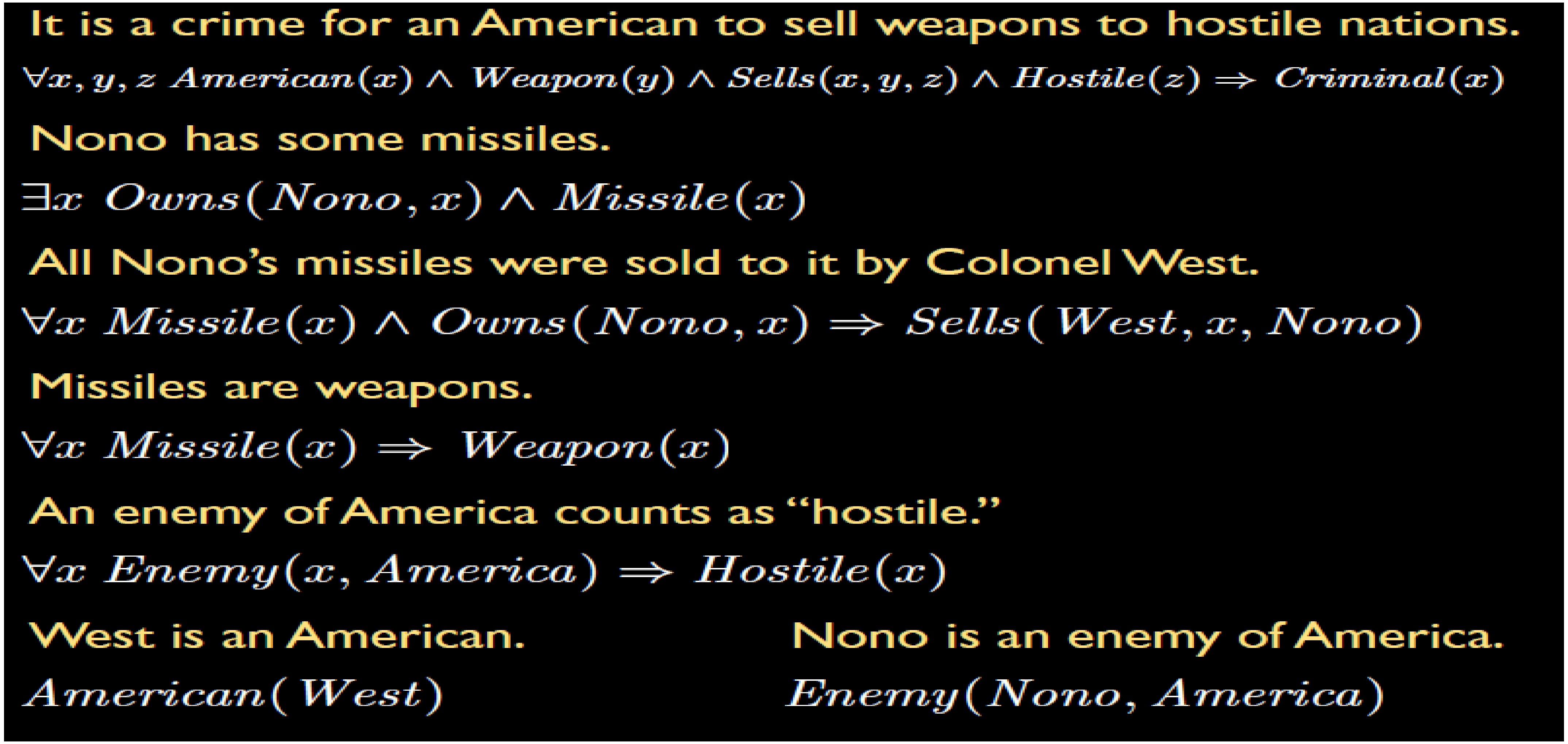
Don't make mistake of <u>first</u> forming clause from conclusion & <u>then</u> denying it:

- Conclusion:
  $\exists x.\mathrm{Older}(x, \mathrm{Fifi})$
  clause form: $\mathrm{Older}(C, \mathrm{Fifi})$
  denial: $\boxed{\neg\mathrm{Older}(C, \mathrm{Fifi})}$

Cannot unify Lulu,C!!

QUESTION

- 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 its missiles were sold to it by Colonel West, who is American. To prove: West is a criminal



It is a crime for an American to sell weapons to hostile nations.

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

Nono has some missiles.

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

All Nono's missiles were sold to it by Colonel West.

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

Missiles are weapons.

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

An enemy of America counts as "hostile."

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

West is an American.

$American(West)$

Nono is an enemy of America.

$Enemy(Nono, America)$

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

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

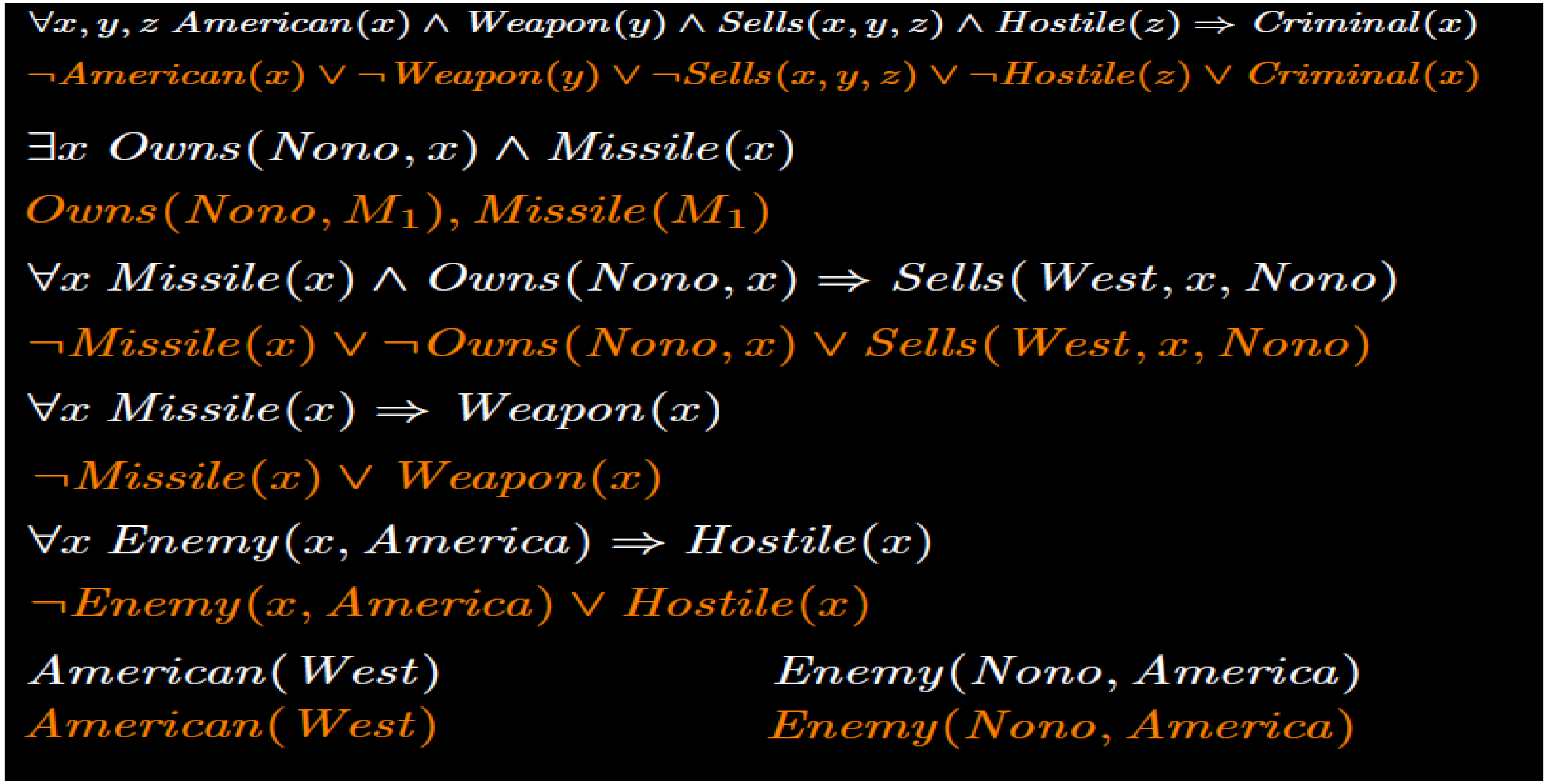
$$\forall x \; Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

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

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

$American(West)$ $\qquad$ $Enemy(Nono, America)$

CONVERT TO NORMAL FORM



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

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

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

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

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

$\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(West, x, Nono)$

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

$\neg Missile(x) \vee Weapon(x)$

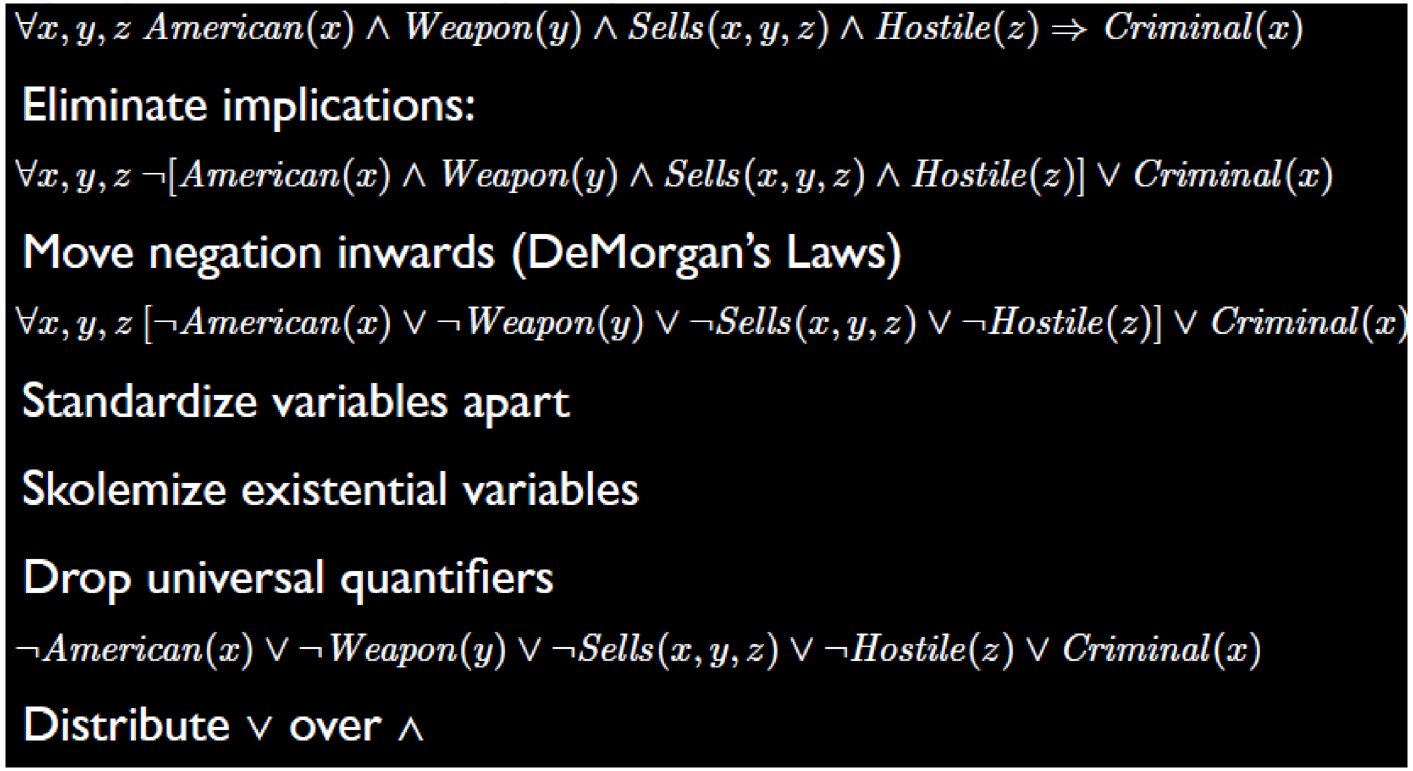
$\forall x\ Enemy(x, America) \Rightarrow Hostile(x)$

$\neg Enemy(x, America) \vee Hostile(x)$

$American(West)$ $Enemy(Nono, America)$

$American(West)$ $Enemy(Nono, America)$

EXAMPLE OF NORMAL FORM



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

Eliminate implications:

$\forall x, y, z\ \neg[American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z)] \vee Criminal(x)$

Move negation inwards (DeMorgan's Laws)

$\forall x, y, z\ [\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x,y,z) \vee \neg Hostile(z)] \vee Criminal(x)$

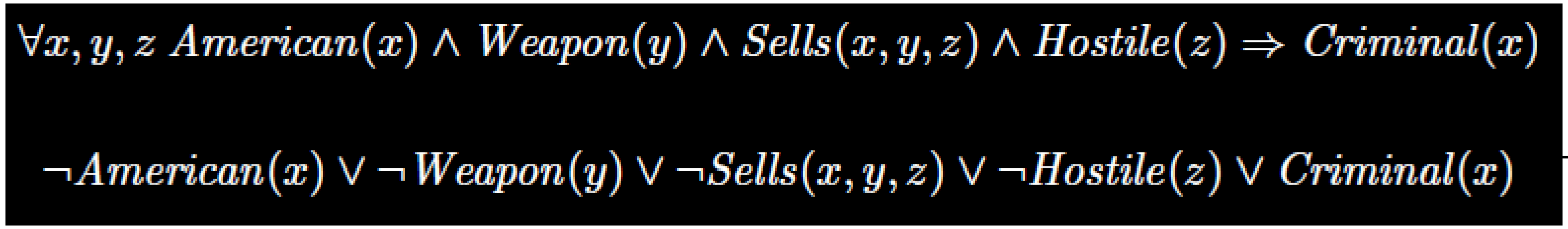
Standardize variables apart

Skolemize existential variables

Drop universal quantifiers

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

Distribute ∨ over ∧



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

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

Negate and convert to CNF

$\neg American(x) \lor \neg Weapon(y) \lor \neg Sells(x, y, z) \lor \neg Hostile(z) \lor Criminal(x)$

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

$\neg Missile(x) \lor \neg Owns(Nono, x) \lor Sells(West, x, Nono)$

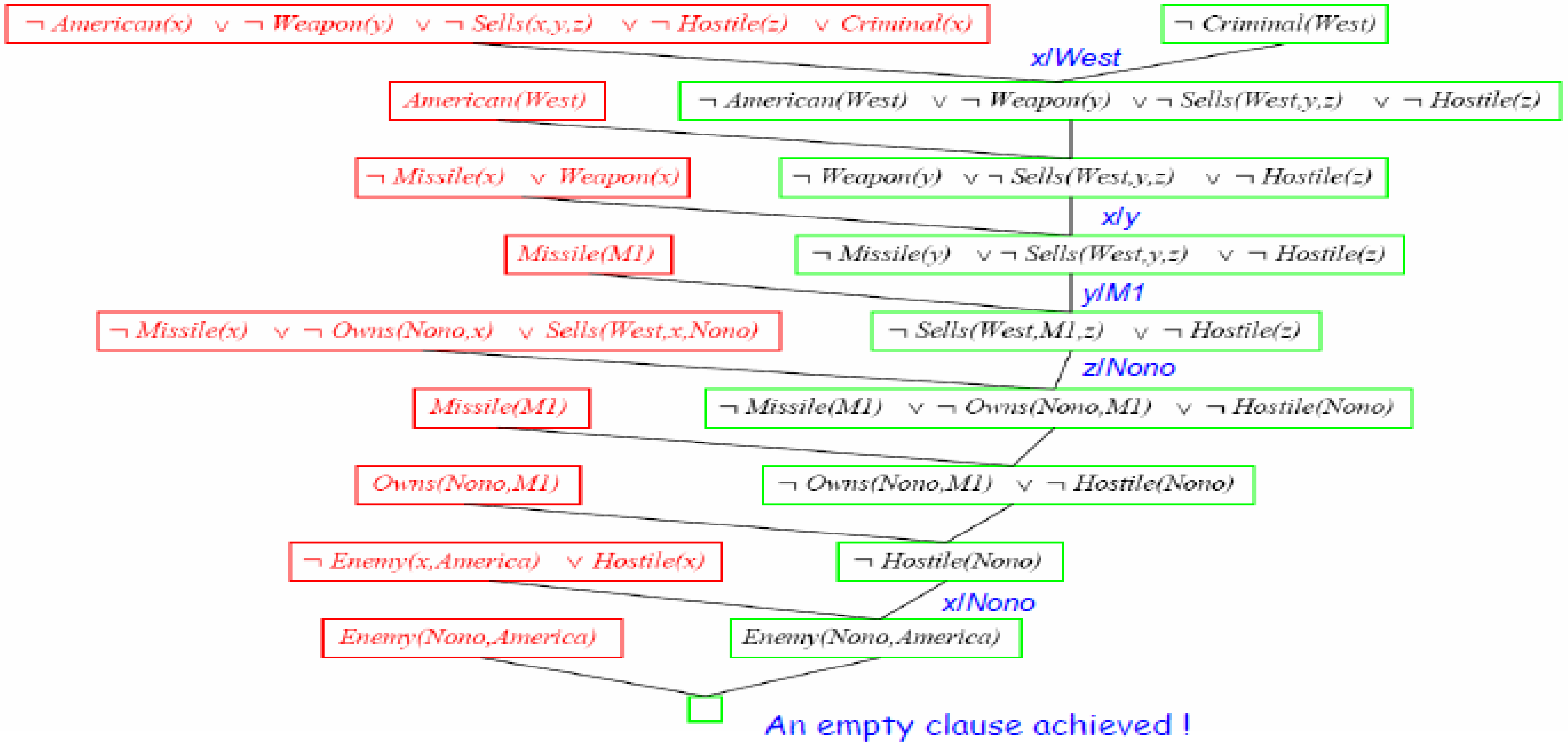
$\neg Missile(x) \lor Weapon(x)$

$\neg Enemy(x, America) \lor Hostile(x)$

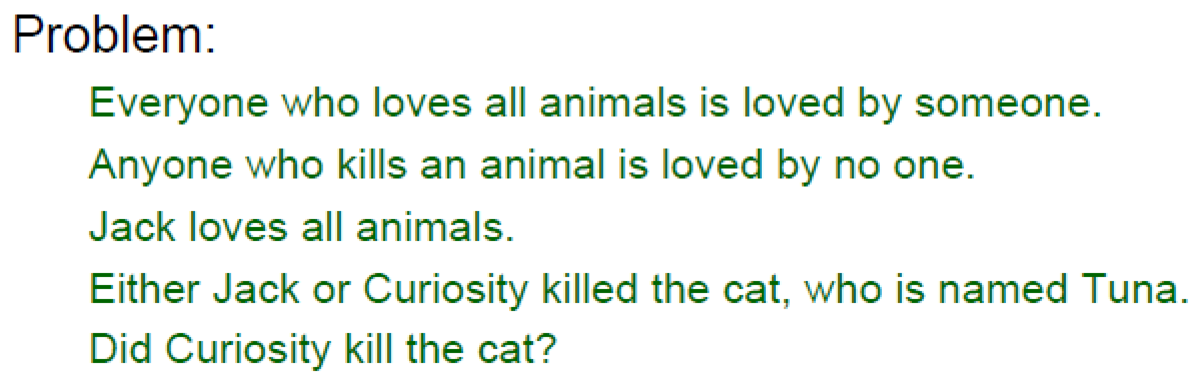
$American(West)$

$Enemy(Nono, America)$

$\neg Criminal(West)$


¬ American(x) ∨ ¬ Weapon(y) ∨ ¬ Sells(x,y,z) ∨ ¬ Hostile(z) ∨ Criminal(x)
¬ Criminal(West)
x/West
American(West)
¬ American(West) ∨ ¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
¬ Missile(x) ∨ Weapon(x)
¬ Weapon(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
x/y
Missile(M1)
¬ Missile(y) ∨ ¬ Sells(West,y,z) ∨ ¬ Hostile(z)
y/M1
¬ Missile(x) ∨ ¬ Owns(Nono,x) ∨ Sells(West,x,Nono)
¬ Sells(West,M1,z) ∨ ¬ Hostile(z)
z/Nono
Missile(M1)
¬ Missile(M1) ∨ ¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
Owns(Nono,M1)
¬ Owns(Nono,M1) ∨ ¬ Hostile(Nono)
¬ Enemy(x,America) ∨ Hostile(x)
¬ Hostile(Nono)
x/Nono
Enemy(Nono,America)
Enemy(Nono,America)
An empty clause achieved !

RESOLUTION EXAMPLE



Problem:

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?

SOLUTION

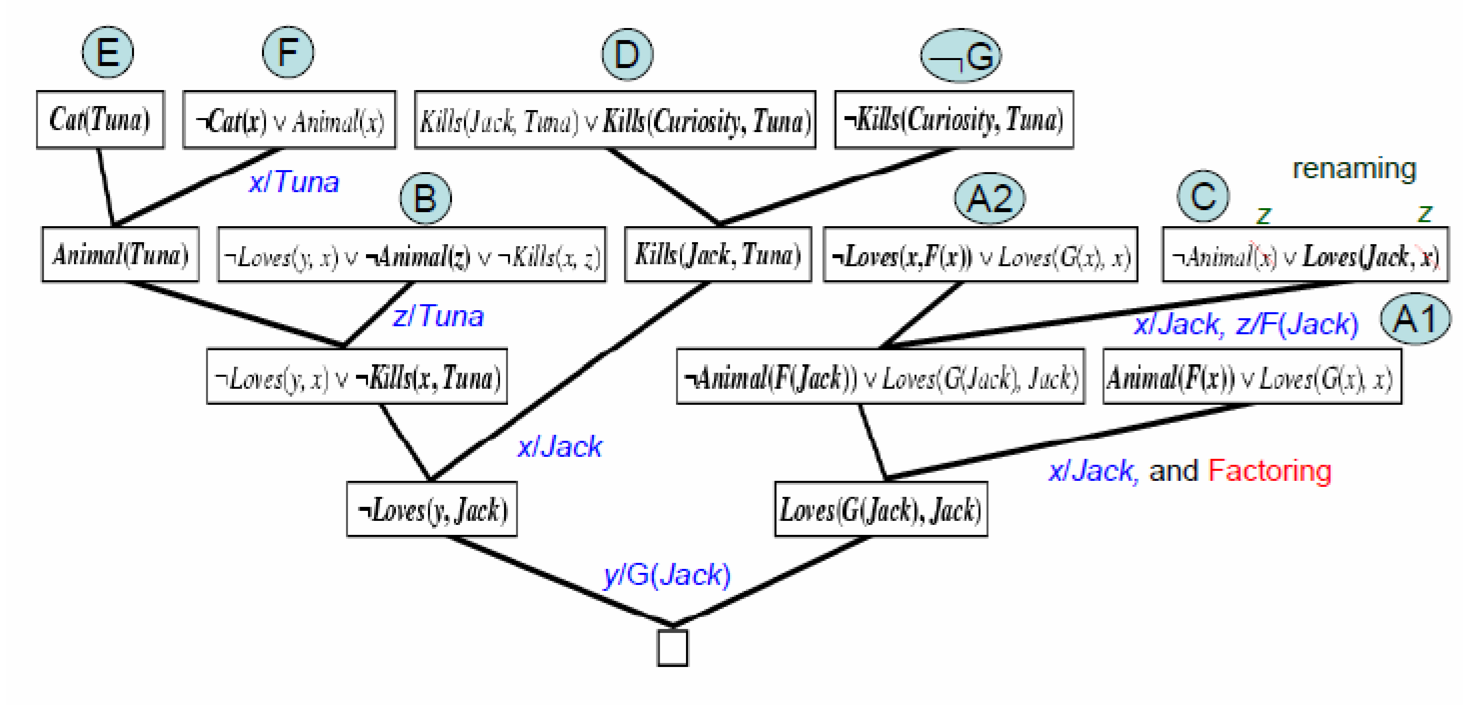
A. $\forall x\ [\forall y\ \ Animal(y) \Rightarrow loves(x, y)\ ] \Rightarrow\ \exists y\ loves(y, x)$
B. $\forall x\ [\forall y\ \ Animal(y) \Rightarrow Kills(x, y)\ ] \Rightarrow\ \forall z\ \neg loves(z, x)$
C. $\forall x\ Animal(x) \Rightarrow loves(Jack, x)$
D. $Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$
E. $Cat(Tuna)$
F. $\forall x\ Cat(x) \Rightarrow Animal(x)$ (background knowledge!)

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

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

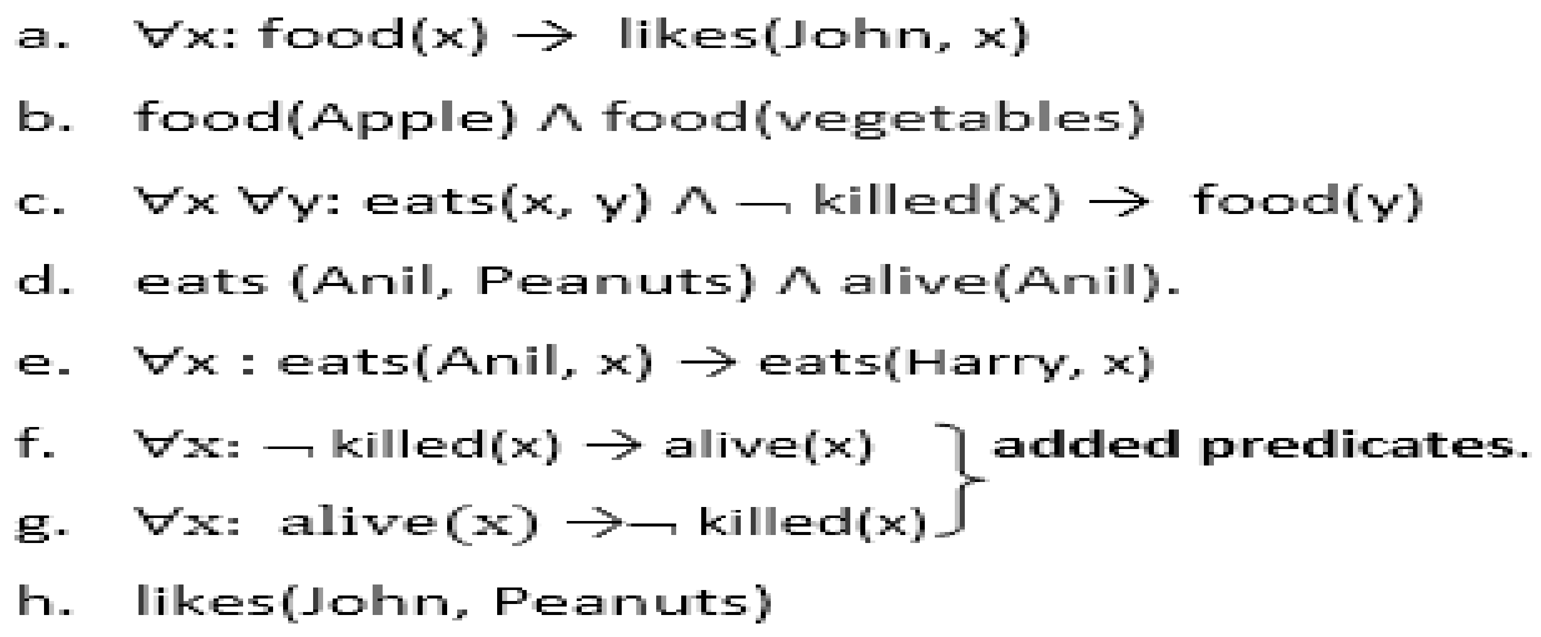
$\neg$G. $\neg Kills(Curiosity, Tuna)$

RESOLUTION GRAPH


E
F
D
¬G
Cat(Tuna)
¬Cat(x) ∨ Animal(x)
Kills(Jack, Tuna) ∨ Kills(Curiosity, Tuna)
¬Kills(Curiosity, Tuna)
x/Tuna
B
A2
C
renaming
z
z
Animal(Tuna)
¬Loves(y, x) ∨ ¬Animal(z) ∨ ¬Kills(x, z)
Kills(Jack, Tuna)
¬Loves(x,F(x)) ∨ Loves(G(x), x)
¬Animal(x) ∨ Loves(Jack, x)
z/Tuna
x/Jack, z/F(Jack)
A1
¬Loves(y, x) ∨ ¬Kills(x, Tuna)
¬Animal(F(Jack)) ∨ Loves(G(Jack), Jack)
Animal(F(x)) ∨ Loves(G(x), x)
x/Jack
x/Jack, and Factoring
¬Loves(y, Jack)
Loves(G(Jack), Jack)
y/G(Jack)
□

STEP-1: CONVERSION OF FACTS INTO FOL

- In the first step we will convert all the given statements into its first order logic.



a. $\forall x$: food(x) $\rightarrow$ likes(John, x)

b. food(Apple) $\land$ food(vegetables)

c. $\forall x \forall y$: eats(x, y) $\land \neg$ killed(x) $\rightarrow$ food(y)

d. eats (Anil, Peanuts) $\land$ alive(Anil).

e. $\forall x$: eats(Anil, x) $\rightarrow$ eats(Harry, x)

f. $\forall x$: $\neg$ killed(x) $\rightarrow$ alive(x)

g. $\forall x$: alive(x) $\rightarrow \neg$ killed(x)

added predicates.

h. likes(John, Peanuts)

STEP-2: CONVERSION OF FOL INTO CNF

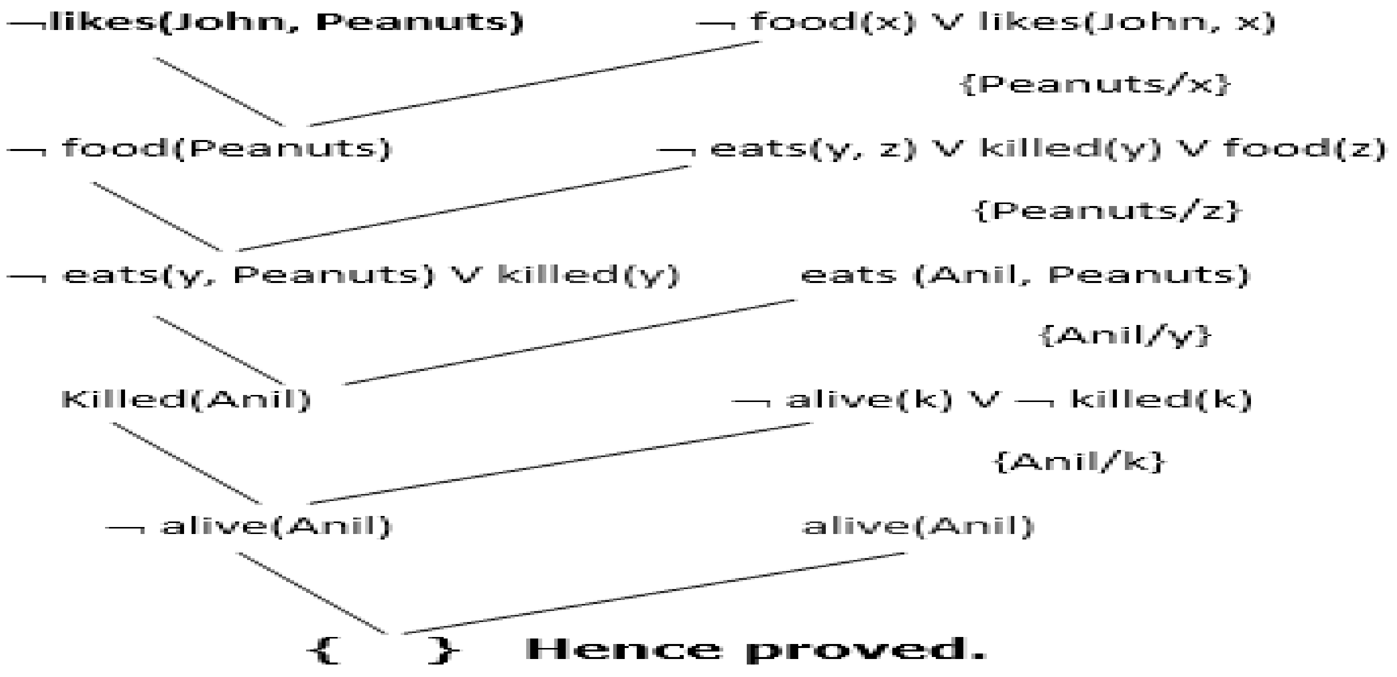
- In First order logic resolution, it is required to convert the FOL into CNF as CNF form makes easier for resolution proofs.
- **Eliminate all implication (→) and rewrite**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀x ∀y ¬ [eats(x, y) Λ ¬ killed(x)] V food(y)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀x ¬ eats(Anil, x) V eats(Harry, x)
  - ∀x¬ [¬ killed(x)] V alive(x)
  - ∀x ¬ alive(x) V ¬ killed(x)
  - likes(John, Peanuts).
- **Move negation (¬)inwards and rewrite**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀x ∀y ¬ eats(x, y) V killed(x) V food(y)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀x ¬ eats(Anil, x) V eats(Harry, x)
  - ∀x ¬killed(x)] V alive(x)
  - ∀x ¬ alive(x) V ¬ killed(x)
  - likes(John, Peanuts).

EXAMPLE

- **Rename variables or standardize variables**
  - ∀x ¬ food(x) V likes(John, x)
  - food(Apple) Λ food(vegetables)
  - ∀y ∀z ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts) Λ alive(Anil)
  - ∀w¬ eats(Anil, w) V eats(Harry, w)
  - ∀g ¬killed(g)] V alive(g)
  - ∀k ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).
- **Eliminate existential instantiation quantifier by elimination.**
  In this step, we will eliminate existential quantifier ∃, and this process is known as Skolemization. But in this example problem since there is no existential quantifier so all the statements will remain same in this step.

- **Drop Universal quantifiers.**
  In this step we will drop all universal quantifier since all the statements are not implicitly quantified so we don't need it.
  - ¬ food(x) V likes(John, x)
  - food(Apple)
  - food(vegetables)
  - ¬ eats(y, z) V killed(y) V food(z)
  - eats (Anil, Peanuts)
  - alive(Anil)
  - ¬ eats(Anil, w) V eats(Harry, w)
  - killed(g) V alive(g)
  - ¬ alive(k) V ¬ killed(k)
  - likes(John, Peanuts).

- **Step-3: Negate the statement to be proved**
- In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts)
- **Step-4: Draw Resolution graph:**
- Now in this step, we will solve the problem by resolution tree using substitution. For the above problem, it will be given as follows:



- .

CHALLENGE

- Consider the following story:
- “Anyone passing his or her artificial intelligence exam and winning the lottery is happy. But anyone who studies or is lucky can pass all his exams. Pete did not study but is lucky. Anyone who is lucky wins the lottery. Is Pete happy?”.

- Step 1: Change sentences to first order logic:

1. "Anyone passing his or her artificial intelligence exam and winning the lottery is happy"

- (∀ X) (PASS(X,ai) ^ WIN(X,lottery)⇒HAPPY(X))

2. "Anyone who studies or is lucky can pass all his exams"

- (∀ X ∀ Y) (STUDIES(X) V LUCKY(X) ⇒PASS(X,Y))

3. "Pete did not study but is lucky"

- ¬ STUDY(pete) ^ LUCKY(pete)

4. "Anyone who is lucky wins the lottery"

- (∀ X) (LUCKY(X) ⇒WINS(X,lottery))

- Step 2: Convert all sentences into clause form:

1. (∀ X) (PASS(X,ai) ^ WIN(X,lottery) ⇒HAPPY(X)) gives:

- ¬ PASS(X,ai) V ¬ WIN(X,lottery) VHAPPY(X)

2. (∀ X ∀ Y) (¬ STUDIES(X) V LUCKY(X) ⇒PASS(X,Y)) gives:

- ¬ STUDIES(Y) V PASS(Y,Z)
- ¬ LUCKY(W) V PASS(W,V)

3. ¬ STUDY(pete) ^ LUCKY(pete) gives:

- ¬ STUDIES(pete)
- LUCKY(pete)

4. (∀ X) (LUCKY(X)⇒WINS(X,lottery)) gives:

- ¬ LUCKY(U) VWINS(U,lottery)

- Step 3: Add negation (in clause form) of what we want to know:
- ¬ HAPPY(pete)
- Step 4: Use resolution (and our negated goal) to build a resolution refutation graph to prove a contradiction.
- When building the graph to prove the contradiction, it should be the case thatthe negative of the goal gets used somewhere in the proof.
- Contradiction occurs when resolve clauses like A, ¬ A
- result = {} (NULL set) since A cannot be true and false at the same time!

$$\begin{tikzcd}
\neg \text{PASS(X,ai)} \vee \neg \text{WIN(X,lotter)} \vee \text{HAPPY(X)} & \text{WIN(U,lottery)} \vee \neg \text{LUCKY(U)} \\
\neg \text{PASS(U,ai)} \vee \text{HAPPY(U)} \vee \neg \text{LUCKY(U)} \arrow[u, "\{x/u\}"] \arrow[ur] & \neg \text{HAPPY(pete)} \\
\text{LUCKY(pete)} & \neg \text{PASS(pete,ai)} \vee \neg \text{LUCKY(pete)} \arrow[ul] \arrow[u, "\{U/John\}"'] \\
\neg \text{PASS(pete,ai)} \arrow[u] \arrow[ur] & \neg \text{LUCKY(V)} \vee \text{PASS(V,W)} \\
\text{LUCKY(pete)} \arrow[u, "\{V/pete,\ w/ai\}"] \arrow[ur] & !\ \neg \text{LUCKY(pete)} \\
\{\} \arrow[u] \arrow[ur] &
\end{tikzcd}$$

RESOLUTION METHODS

- **Resolution Method:** Unification, Backward Chaining as well as Forward Chaining, all are based on the resolution method. The resolution method is a decision-making rule where a machine decides what to do and what not to do. It can be understood as:
- Firstly, we convert the given statement into FOPL.
- Secondly, we infer some related sentences from it.
- Then, realizing our goal, we need to prove it.
- Here, we make use of the resolution method to decide the best possible FOPL from various.

FORWARD AND BACKWARD CHAINING

- **Forward Chaining:** In forward chaining, we start with the atomic sentences in the knowledge base and apply Modus Ponen in forward direction. Also adding new sentences until any inference is not made. Use modus ponens to always deriving all consequences from new information. Inferences cascade to draw deeper and deeper conclusions. To avoid looping and duplicated effort, must prevent addition of a sentence to the KB which is the same as one already present. Must determine all ways in which a rule (Horn clause) can match existing facts to draw new conclusions.
- **Backward Chaining:** This algorithm is opposite of forward chaining algorithm. It works in backward direction. It starts from the goal, chain through the rules to search the known facts and infer sentences from the given facts in backward direction. Start from query or atomic sentence to be proven and look for ways to prove it. Query can contain variables which are assumed to be existentially quantified

FORWARD CHAINING EXAMPLE

Assume in KB

1) Parent(x,y) $\wedge$ Male(x) $\Rightarrow$ Father(x,y)

2) Father(x,y) $\wedge$ Father(x,z) $\Rightarrow$ Sibling(y,z)

Add to KB

3) Parent(Tom,John)

Rule 1) tried but can't "fire"

Add to KB

4) Male(Tom)

Rule 1) now satisfied and triggered and adds:

5) Father(Tom, John)

Rule 2) now triggered and adds:

6) Sibling(John, John) {x/Tom, y/John, z/John}

Add to KB
7) Parent(Tom,Fred)
Rule 1) triggered again and adds:
8) Father(Tom,Fred)
Rule 2) triggered again and adds:
9) Sibling(Fred,Fred) {x/Tom, y/Fred, z/Fred}
Rule 2) triggered again and adds:
10) Sibling(John, Fred) {x/Tom, y/John, z/Fred}
Rule 2) triggered again and adds:
11) Sibling(Fred, John) {x/Tom, y/Fred, z/John}

EXAMPLE

- The law is that it is a crime for an American to sell weapon to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold by Colonel West, who is American. Prove that West is a criminal.
- Criminal(West) true or false ?

- It is a crime for an American to sell weapon to hostile nations

  $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$ ①

- The country Nono has some missiles

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

  $\Rightarrow Owns(Nono, M_1)$ ② , $Missile(M_1)$ ③

  existential elimination/instantiation
  AND elimination

- All its (Nono's) missiles are sold to it by West

  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$ ④

- Missiles are weapons

  $Missile(x) \Rightarrow Weapon(x)$ ⑤

- An enemy of America counts as "hostile"

  $Enemy(x, America) \Rightarrow Hostile(x)$ ⑥

A datalog KB:
composed of a set of FOL definite clauses with no function symbols

background knowledge!

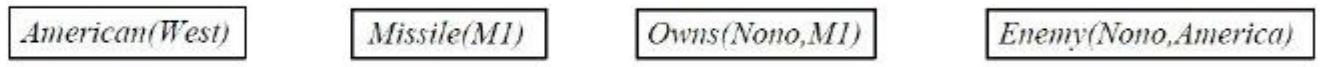
- West, who is American

  $American(West)$ ⑦

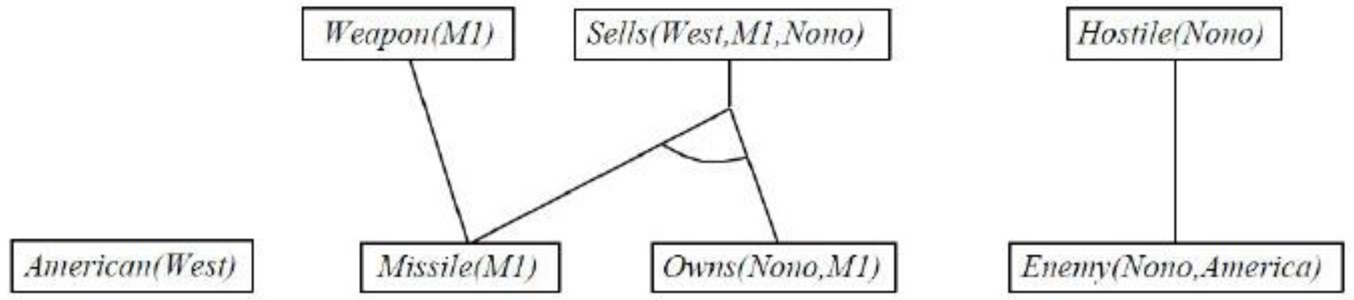
- The country Nono, an enemy of America

  $Enemy(Nono, America)$ ⑧

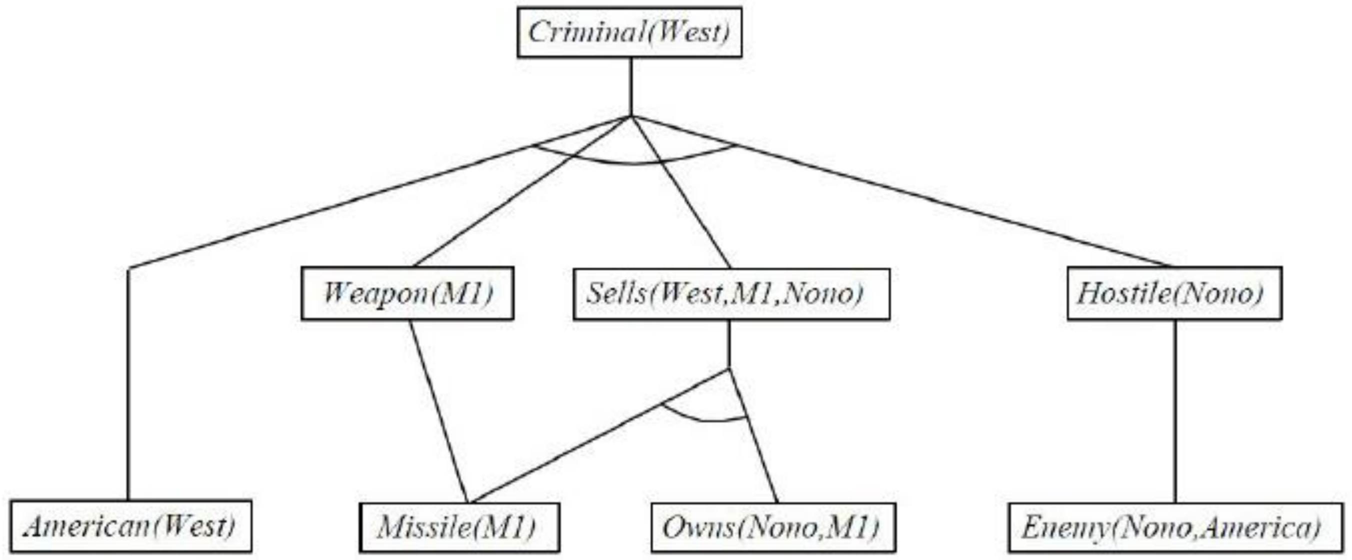
FORWARD CHAINING



FORWARD CHAINING

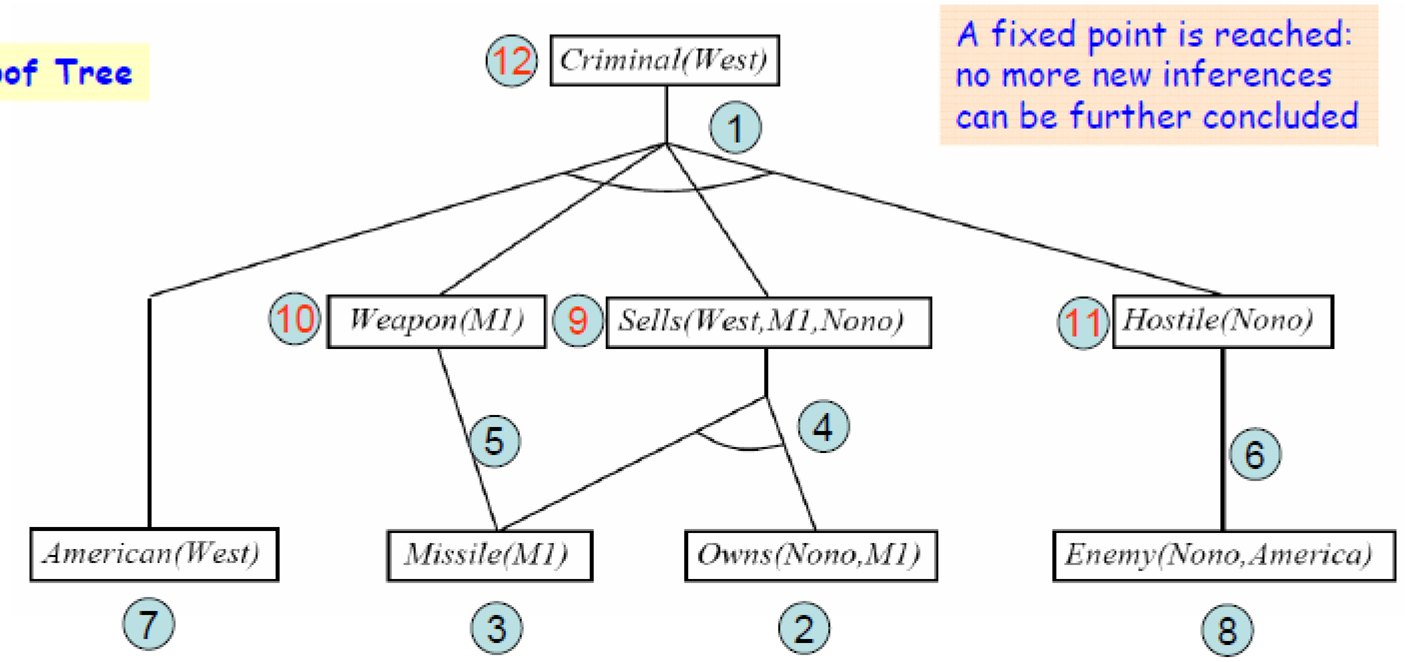


FORWARD CHAINING



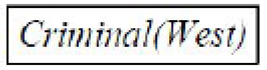
FC

Proof Tree

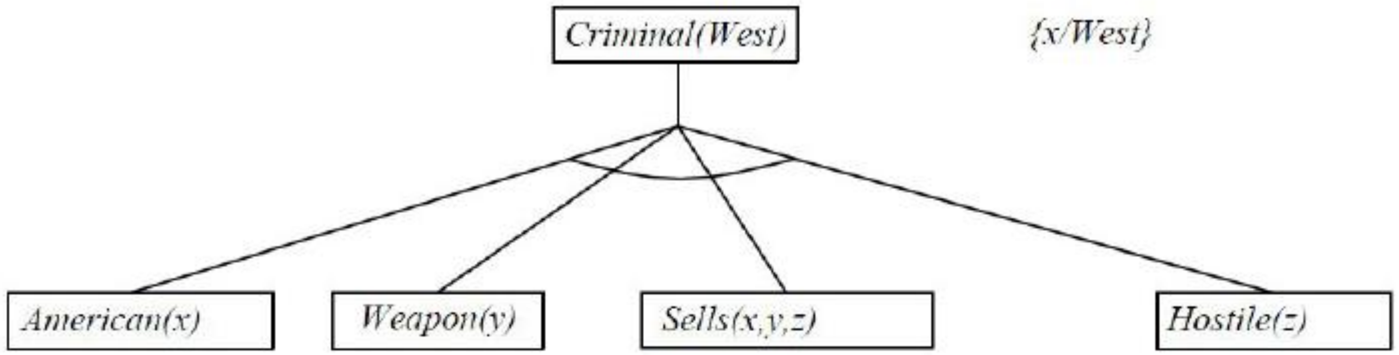


A fixed point is reached:
no more new inferences
can be further concluded

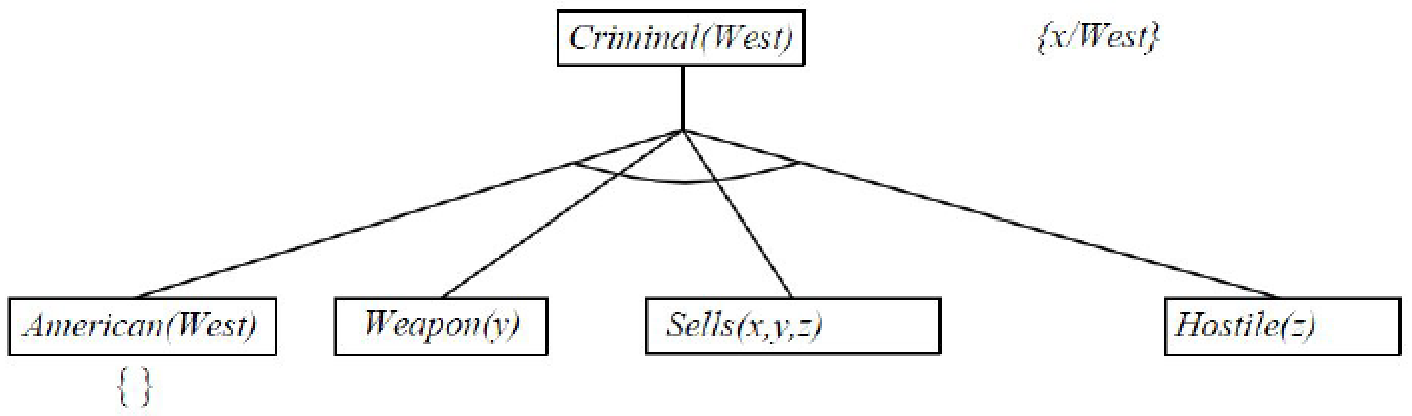
BACKWARD CHAINING



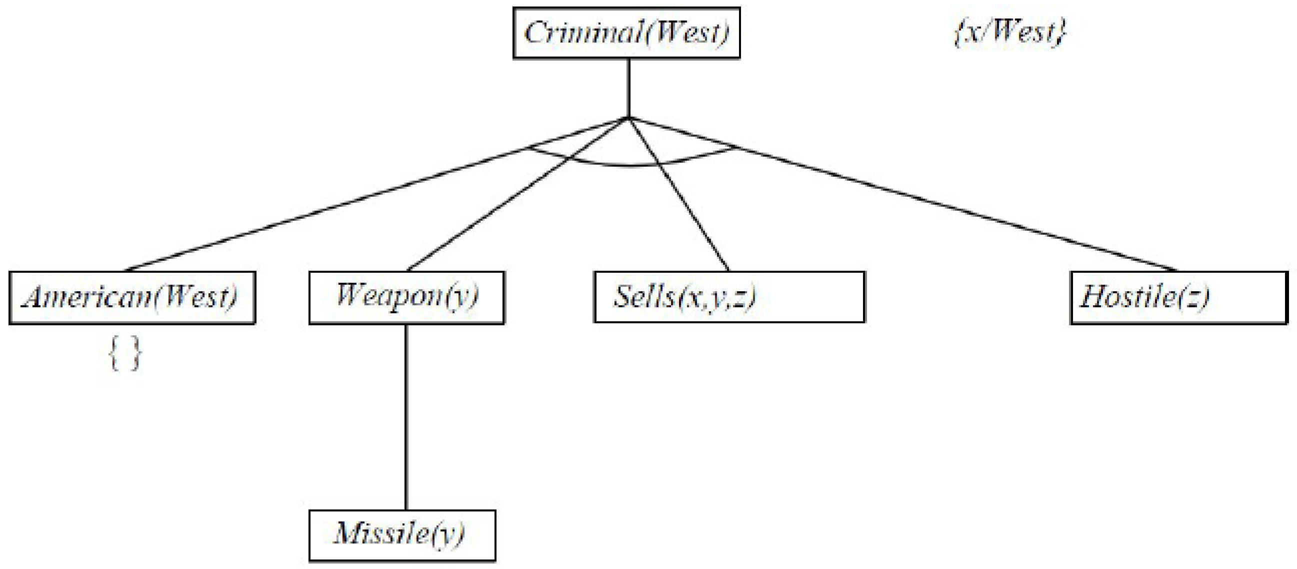
BACKWARD CHAINING



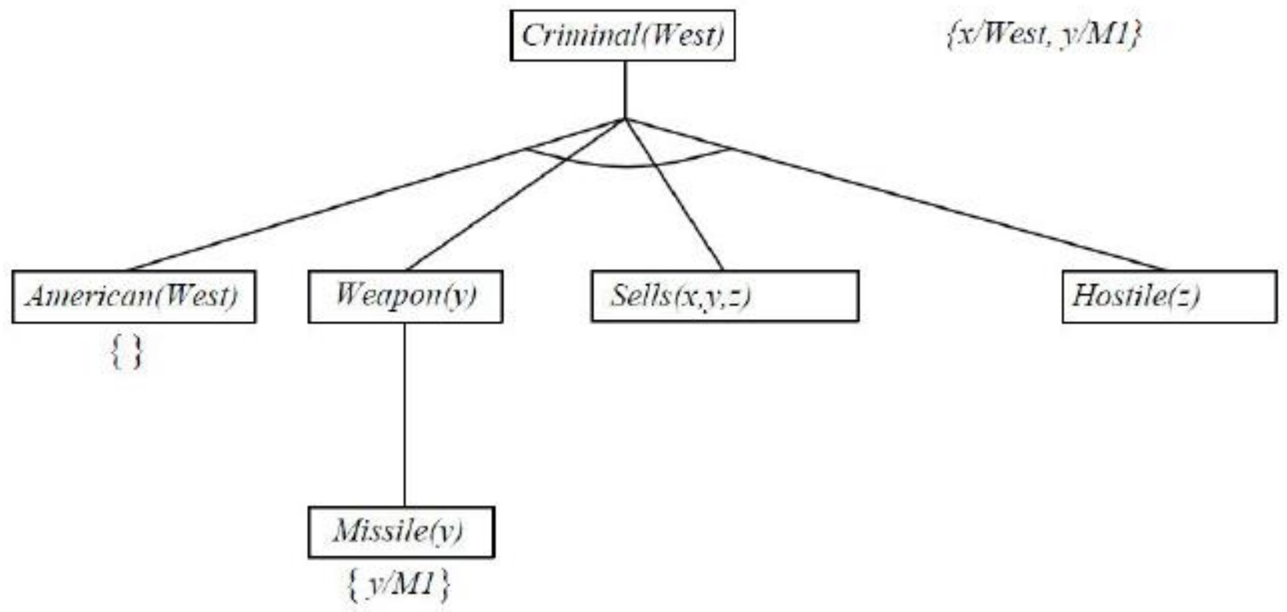
BACKWARD CHAINING



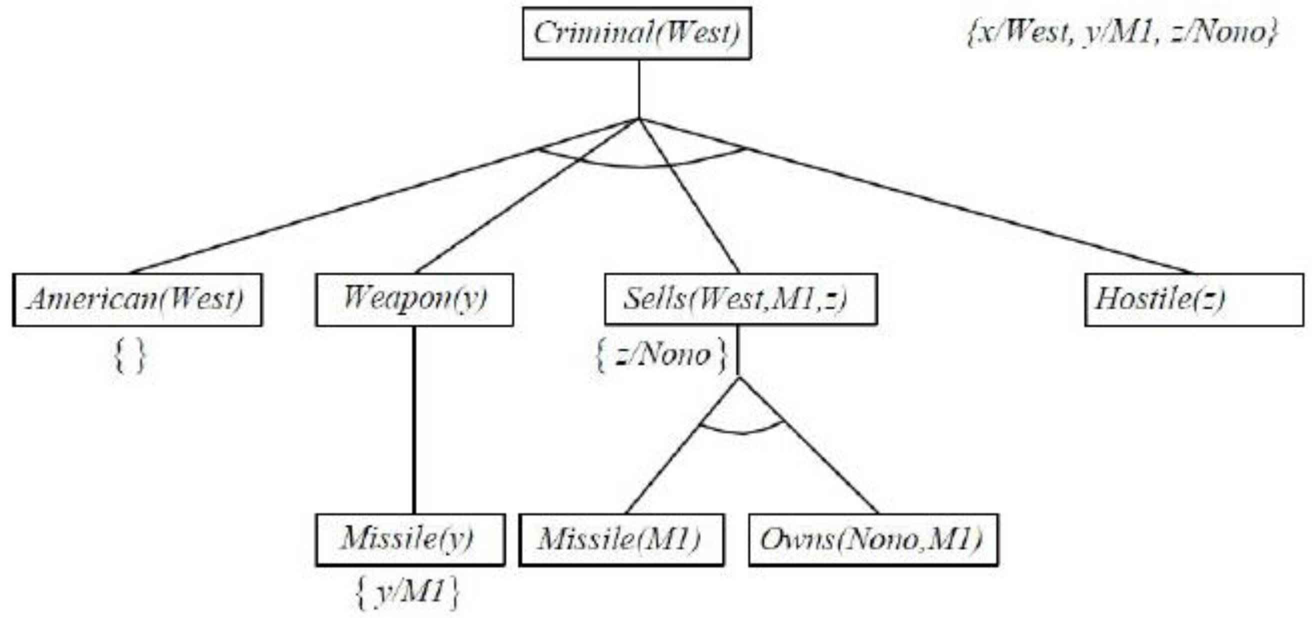
BACKWARD CHAINING



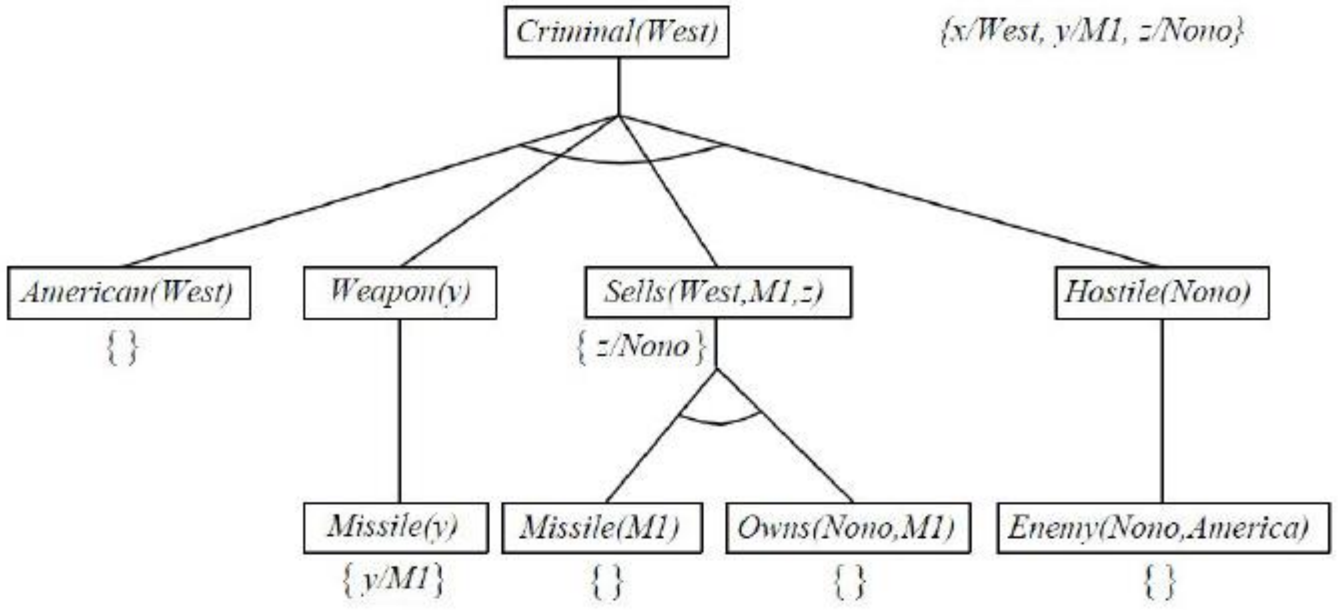
BACKWARD CHAINING



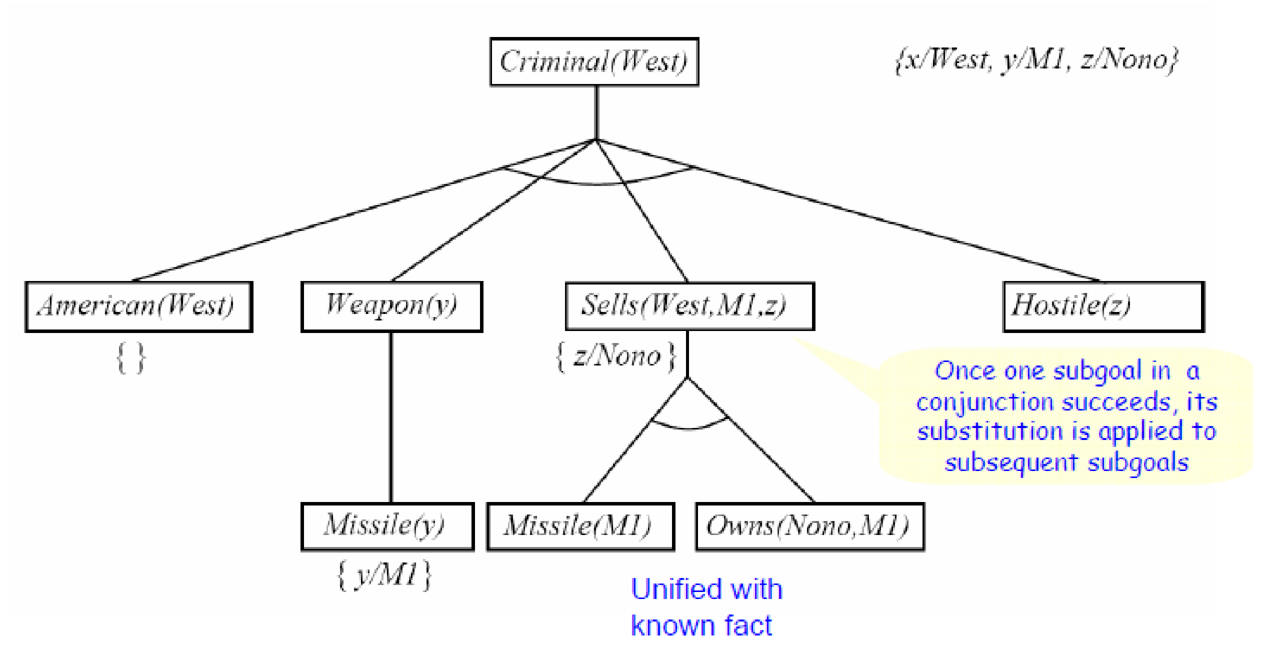
BACKWARD CHAINING



BACKWARD CHAINING

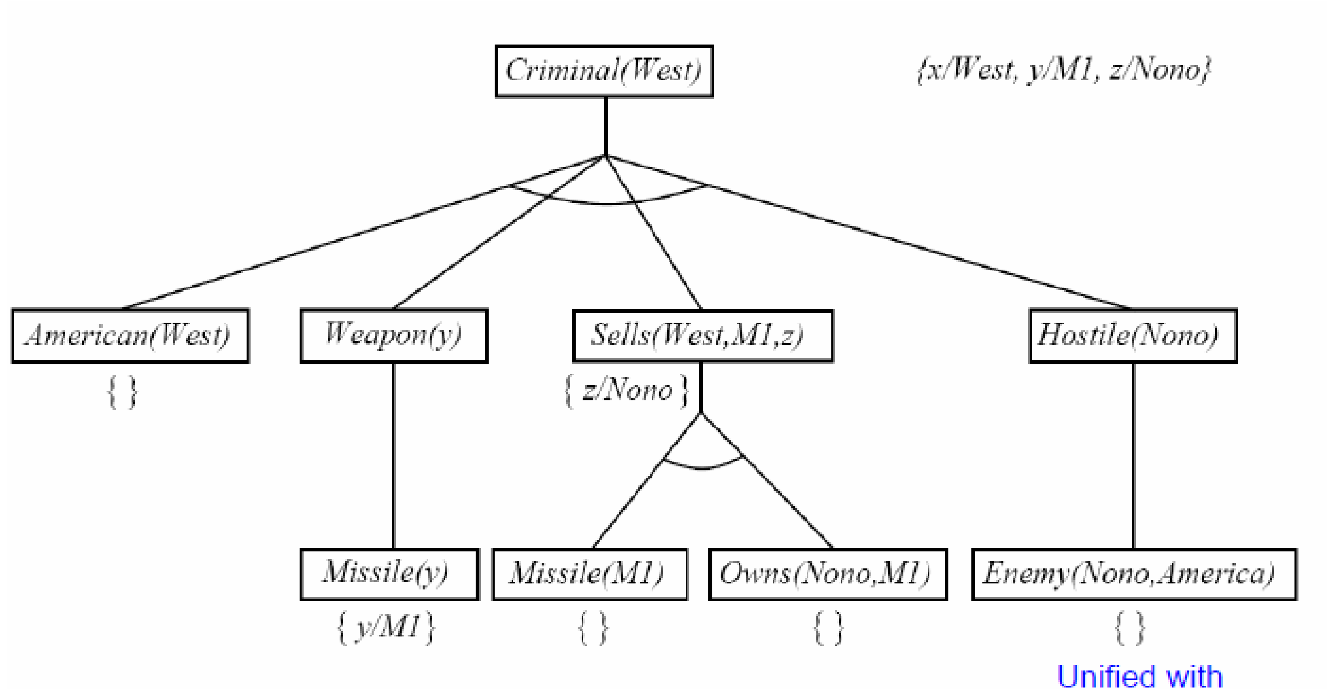


BACKWARD CHAINING



SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

BACKWARD CHAINING



Unified with known fact

BACKWARD CHAINING EXAMPLE

- KB:

1) Parent(x,y) Ù Male(x) Þ Father(x,y)

2) Father(x,y) Ù Father(x,z) Þ
Sibling(y,z)

3) Parent(Tom,John)

4) Male(Tom)

7) Parent(Tom,Fred)

Query: Parent(Tom,x)

SOLUTION

Answers: ({x/John}, {x/Fred})

Query: Father(Tom,s)

Subgoal: Parent(Tom,s) Ù Male(Tom)

{s/John}

Subgoal: Male(Tom)

Answer: {s/John}

{s/Fred}

Subgoal: Male(Tom)

Answer: {s/Fred}

Answers: ({s/John}, {s/Fred})
Query: Father(f,s)
Subgoal: Parent(f,s) Ù Male(f)
{f/Tom, s/John}
Subgoal: Male(Tom)
Answer: {f/Tom, s/John}
{f/Tom, s/Fred}

Subgoal: Male(Tom)
Answer: {f/Tom, s/Fred}
Answers: ({f/Tom,s/John}, {f/Tom,s/Fred})
Query: Sibling(a,b)
Subgoal: Father(f,a) Ù Father(f,b)
{f/Tom, a/John}
Subgoal: Father(Tom,b)
{b/John}
Answer: {f/Tom, a/John, b/John}
{b/Fred}
Answer: {f/Tom, a/John, b/Fred}
{f/Tom, a/Fred}
Subgoal: Father(Tom,b)
{b/John}
Answer: {f/Tom, a/Fred, b/John}
{b/Fred}
Answer: {f/Tom, a/Fred, b/Fred}
Answers: ({f/Tom, a/John, b/John},{f/Tom, a/John, b/Fred}
{f/Tom, a/Fred, b/John}, {f/Tom, a/Fred, b/Fred})

PROBLEMS WITH FORWARD CHAINING

- Inference can explode forward and may never terminate.
- Inference is not directed towards any particular conclusion or goal. May draw lots of irrelevant conclusions.

FORWARD VS BACKWARD CHAINING

No	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	Forward chaining is known as data-driven inference technique as we reach to the goal using the available data.	Backward chaining is known as goal-driven technique as we start from the goal and divide into sub-goal to extract the facts.
4.	Forward chaining reasoning applies a breadth-first search strategy.	Backward chaining reasoning applies a depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

anooja@somaiya.edu

THANK YOU