Artificial Intelligence

Unit 2: Inferences in Propositional and First Order Logic

Syedur Rahman
Lecturer, CSE Department
North South University
syedur.rahman@wolfson.oxon.org

Artificial Intelligence: Lecture Notes

The lecture notes from the introductory lecture and this unit will be available shortly from the following URL:

- http://www.geocities.com/syedatnsu/

Acknowledgements

- These lecture notes contain material from the following sources
  - *Intelligent Systems* by S.Clark, 2005
  - *Logical Programming and Artificial Intelligence* by S. Kapetanakis, 2004
  - *Artificial Intelligence: A modern approach* by S. Russell and P. Norvig, International Edition, 2nd edition

Resolution

- Resolution is a method of inference of queries from knowledge bases in propositional logic that can be used when the statements are in Conj. Normal Form.

- The Resolution Rule states that the clauses $x \vee y$ and $z \vee \neg y$ can be resolved to $x \vee z$

- **Conjunctive Normal Form (CNF)**
  conjunction of disjunctions of literals
  E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$□
  But not $(A \wedge \neg B) \vee (\neg C \vee \neg D)$□

Remember the Connectives and Rules in Propositional Logic

high priority

$\neg$	not
$\wedge$	and
$\vee$	or
$\Rightarrow$	implies
$\Leftrightarrow$	if and only if

low priority

xor **exclusive or** $p \text{ xor } q \equiv (p \vee q) \wedge \neg(p \wedge q)$

$\downarrow$ **negated or** $p \downarrow q \equiv \neg(p \vee q)$

$\uparrow$ **negated and** $p \uparrow q \equiv \neg(p \wedge q)$

$\Rightarrow$ **implication** $p \Rightarrow q \equiv \neg p \vee q$

$\Leftrightarrow$ if and only if $p \Leftrightarrow q \equiv (q \Rightarrow p) \wedge (p \Rightarrow q)$

$$\left.\begin{array}{rcl} p \vee p & \equiv & p \\ p \wedge p & \equiv & p \end{array}\right\} \textit{idempotence}$$

$$\left.\begin{array}{rcl} (p \vee q) \vee r & \equiv & p \vee (q \vee r) \\ (p \wedge q) \wedge r & \equiv & p \wedge (q \wedge r) \end{array}\right\} \textit{associativity}$$

$$\left.\begin{array}{rcl} p \vee \neg p & \equiv & T \\ p \wedge \neg p & \equiv & F \end{array}\right\} \textit{excluded middle}$$

$$\left.\begin{array}{rcl} p \vee q & \equiv & q \vee p \\ p \wedge q & \equiv & q \wedge p \end{array}\right\} \textit{commutativity}$$

$$\left.\begin{array}{rcl} p \vee F & \equiv & p \\ p \wedge T & \equiv & p \end{array}\right\} \textit{identity}$$

$$\left.\begin{array}{rcl} p \vee (q \wedge r) & \equiv & (p \vee q) \wedge (p \vee r) \\ p \wedge (q \vee r) & \equiv & (p \wedge q) \vee (p \wedge r) \end{array}\right\} \textit{distributivity}$$

$$\left.\begin{array}{rcl} p \vee T & \equiv & T \\ p \wedge F & \equiv & F \end{array}\right\} \textit{strictness}$$

$$\left.\begin{array}{rcl} \neg(p \vee q) & \equiv & (\neg p) \wedge (\neg q) \\ \neg(p \wedge q) & \equiv & (\neg p) \vee (\neg q) \end{array}\right\} \textit{de Morgan}$$

$\neg\neg p \equiv p$ *double negation*

Resolution Algorithm

- To show $KB \models \alpha$ we show that $KB \wedge \neg\alpha$ is unsatisfiable (where α is a query)
  - Convert $KB \wedge \neg\alpha$ into CNF
  - Apply resolution rule to resulting clauses
  - Continue until there are no new clauses that can be added (in which case *KB* does not entail α)
  - Or until the empty clause is derived (in which case *KB* does entail α)

For Example

KB: p

q

$p \wedge q \Rightarrow r$

Query: r

Consider the following resolution

Knowledge Base:

$p \vee q$

$p \Rightarrow x$

$\neg q$

$x \Rightarrow y$

Query:

y

Consider the following resolution

Knowledge Base:

$$p$$

$$p \Rightarrow x$$

$$\neg q$$

$$z \Rightarrow y$$

$$x \Rightarrow y$$

Query:

$$z$$

Example Conversion to CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

$\equiv$ (Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$)

$(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

$\equiv$ (Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$)

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

$\equiv$ (Move $\neg$ inwards using de Morgan's rules and double-negation)

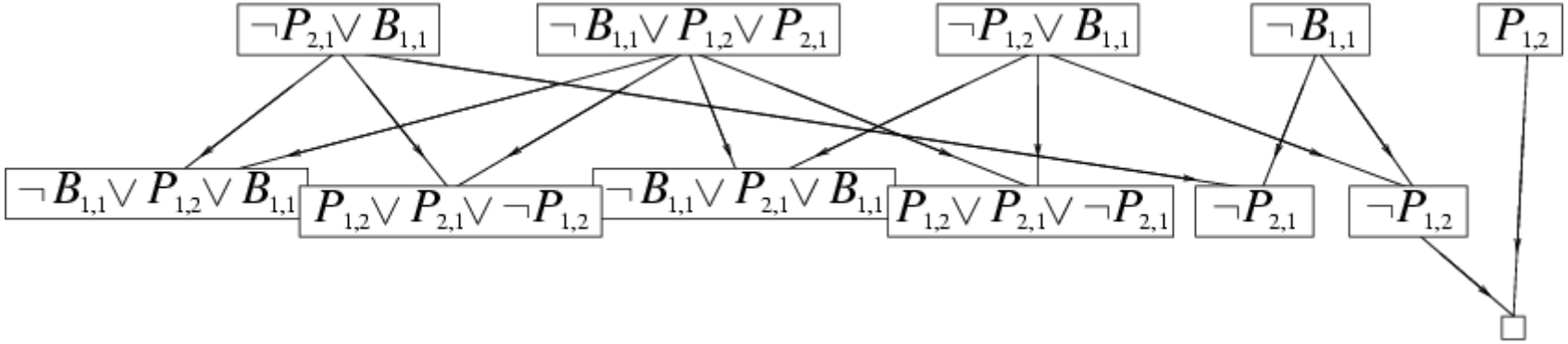
$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

$\equiv$ (Apply distributivity law ($\wedge$ over $\vee$) and flatten)

$(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Resolution example

- $KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ $\alpha = \neg P_{1,2}$



- Empty clause is equivalent to *false* and therefore the argument is valid

Horn Clauses

A **Horn clause** is a disjunction of literals of which *at most one is positive*

- $(\neg x \vee \neg y \vee z)$ is a Horn clause
- $(\neg z \vee a \vee b)$ is not a Horn clause

- Every Horn clause can be written as an implication whose premise is a conjunction of positive literals and whose conclusion is a single positive literal
  - $(\neg x \vee \neg y \vee z)$ can be written as $(x \wedge y) \Rightarrow z$
- The positive literal is called the **head** and the negative literals form the **body** of a horn clause.
- In a knowledge base, a definite clause with no negative literals is called a **fact** (E.g. x, y) as opposed to **rules** e.g. $(\neg x \vee \neg y \vee z)$, $(x \wedge y) \Rightarrow z$
- Real-world knowledge bases often contain clauses of this restricted kind

Horn Clauses

- Inference with Horn clauses can be done through **forward chaining** and **backward chaining** algorithms
  - Both algorithms have inference steps which are easy to follow for humans
- Both forward and backward chaining algorithms are **sound** and **complete**
- Deciding entailment with Horn clauses can be done in time that is *linear* in the size of the knowledge base
- Horn clauses form the basis for the logic programming language Prolog

Inference Rules

modus ponens

from a and $a \Rightarrow b$ derive b

modus tollens

from $a \Rightarrow b$ and $\neg b$ derive $\neg a$

and-introduction

from a and b derive $a \wedge b$

and-elimination

from $a \wedge b$ derive a

Forward chaining

Idea: fire any rule whose premises are satisfied in the *KB,* add its conclusion to the *KB*, until query is found.

Forward chaining

Idea: fire any rule whose premises are satisfied in the *KB,* add its conclusion to the *KB*, until query is found. Consider the following example:

Knowledge Base:

1. $E \Rightarrow D$
2. $E \Rightarrow B$
3. $(B \wedge D) \Rightarrow A$
4. E
5. C

Query: $A \wedge C$

Forward Chaining Derivation

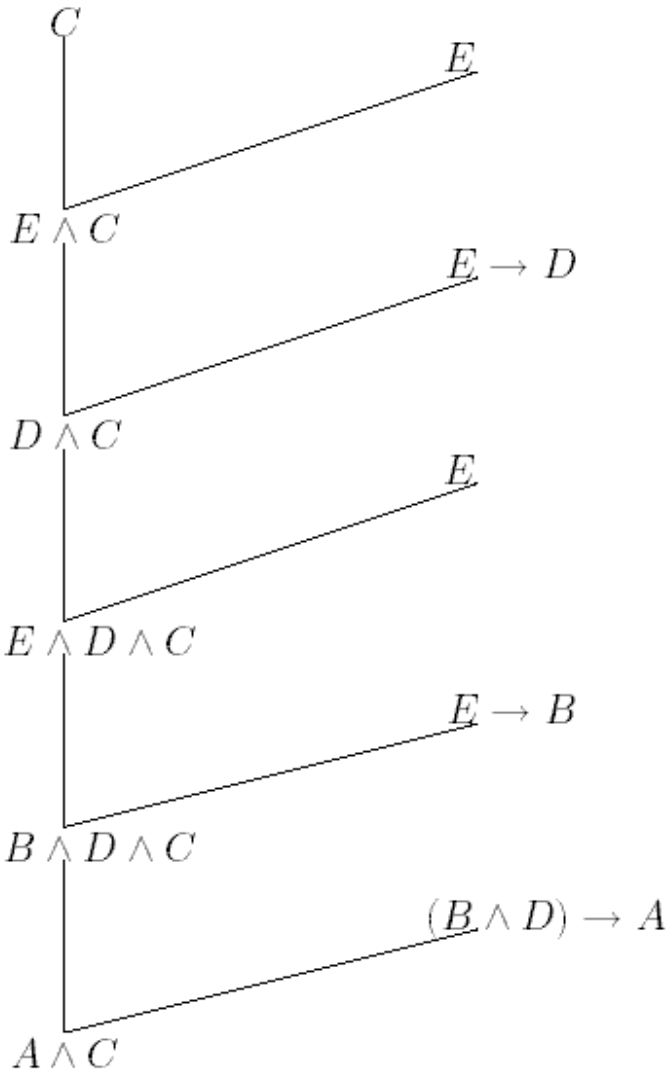
KB:

1. $E \Rightarrow D$
2. $E \Rightarrow B$
3. $(B \wedge D) \Rightarrow A$
4. E
5. C

Query:

$A \wedge C$

Forward Chaining Derivation



Forward chaining with AND-OR graph

- Idea: fire any rule whose premises are satisfied in the *KB*
  - add its conclusion to the *KB*, until query is found

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

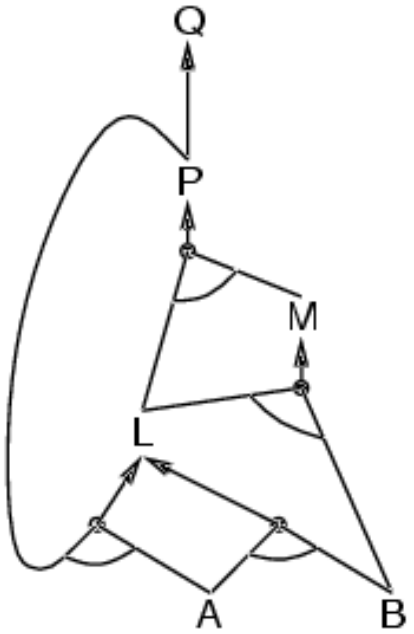
Query: Q

Figure: A simple knowledge base of horn clauses and corresponding AND-OR graph

Forward chaining with AND-OR graph

- Idea: fire any rule whose premises are satisfied in the *KB*
  - add its conclusion to the *KB*, until query is found

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Query: Q

Figure: A simple knowledge base of horn clauses and corresponding AND-OR graph

Data-Driven Reasoning

- Forward chaining is an example of **data-driven** reasoning
  - Reasoning starts with the known data
  - Can be used by an agent to derive conclusions from percepts without a specific query in mind
  - Humans use some data-driven reasoning (while keeping forward chaining under control)
- In contrast, backward chaining is **goal-directed** reasoning
  - Works backwards from the query

Backward chaining

Work backwards from the query *q*

- If *q* is known to be true, we're done
- Otherwise find implications in KB which conclude *q*
- If premises of one of these implications can be proved true (by backward chaining) *q* is true

Backward chaining

Work backwards from the query *q*

- If *q* is known to be true, we're done
- Otherwise find implications in KB which conclude *q*
- If premises of one of these implications can be proved true (by backward chaining) *q* is true

Consider the previous example

Knowledge Base:

1. $E \Rightarrow D$
2. $E \Rightarrow B$
3. $(B \wedge D) \Rightarrow A$
4. E
5. C

Query: $A \wedge C$

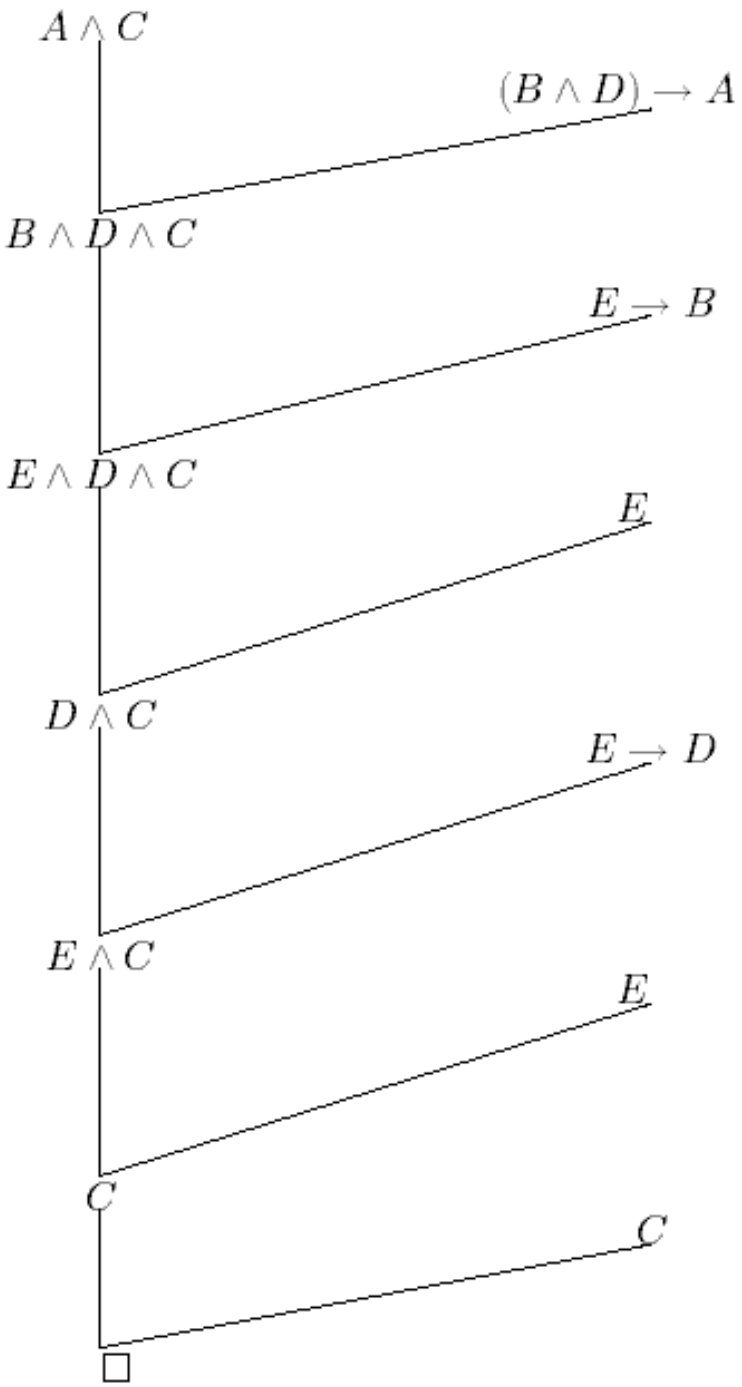
Backward chaining Derivation

KB:

1. $E \Rightarrow D$
2. $E \Rightarrow B$
3. $(B \wedge D) \Rightarrow A$
4. E
5. C

Query:

$A \wedge C$



Backward chaining

Work backwards from the query *q*

- If *q* is known to be true, we're done
- Otherwise find implications in KB which conclude *q*
- If premises of one of these implications can be proved true (by backward chaining) *q* is true

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$

Query: Q

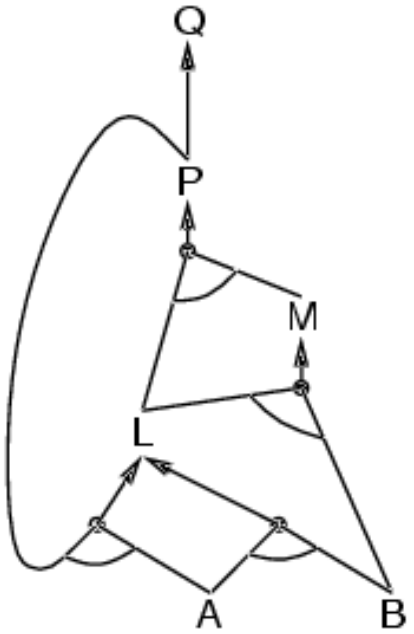
Figure: A simple knowledge base of horn clauses and corresponding AND-OR graph

Backward chaining

Work backwards from the query *q*

- If *q* is known to be true, we're done
- Otherwise find implications in KB which conclude *q*
- If premises of one of these implications can be proved true (by backward chaining) *q* is true

$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



Query: Q

Figure: A simple knowledge base of horn clauses and corresponding AND-OR graph

Forward vs. backward chaining

- FC is data-driven, automatic, unconscious processing
  - e.g. object recognition, routine decisions
- FC may do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving
  - e.g. Where are my keys? What shall I do now?
- Complexity of BC can be *much less* than linear in size of KB
  - Process considers only relevant facts
- Agent should use both FC and BC, limiting forward reasoning to generation of facts that are likely to be relevant to queries solved by backward chaining

First-order logic

- Propositional logic assumes the world contains facts
- First-order logic (much like natural language) assumes the world contains:
  - **Objects**: people, houses, numbers, colours, football games, wars, …
  - **Relations**: red, round, prime, brother of, bigger than, part of, comes between, …
  - **Functions**: father of, best friend, one more than, plus, …

Syntax of FOL: Basic elements

- Constant symbols: John, 2, Richard, Oxford,...
- Predicate symbols: Brother, >, Male, Female...
- Function symbols: PosSqrt, LeftLegOf, Length...
- Variables: x, y, a, b,...
- Connectives: $\neg$, $\Rightarrow$, $\wedge$, $\vee$, $\Leftrightarrow$
- Equality: =
- Quantifiers: $\forall$, $\exists$

Atomic sentences

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

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

Example

- *Sibling(John,Richard), Brother(John,Richard)*
- *Male(Richard), Male(John)*
- *> (Length(LeftLegOf(Richard)),Length(LeftLegOf(John)))*

Complex sentences

- Complex sentences are made from atomic sentences using connectives

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

- e.g.

  Sibling(John,Richard) $\Rightarrow$ *Sibling(Richard,John)*

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

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

Truth in first-order logic

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

Quantifiers in First Order Logic

$\forall$ **Universal quantifier**: A formula $\forall x,p(x)$ reads 'for all values of x in a particular universe of discourse or domain, $p(x)$ is true'.

$\forall x,$dishonest(x) reads 'everyone/thing is dishonest' but if the domain of x is specified as politicians then it reads "all politicians are dishonest'

$\forall x,(horse(x) \Rightarrow quadruped(x))$ reads 'if x is a horse then it is a quadruped' or in other words 'all horses are quadrupeds'

$\exists$ **Existential quantifier**: A formula $\exists x,p(x)$ reads 'there exists a value of x such that $p(x)$ is true'.

$\exists$x, $horse(x)$ reads 'there is a horse'

$\exists$x,($horse$(x) $\wedge$ $colour(x)$=$black$) reads 'there is a horse which is black'

Binding Variables: A variable x is bound, when a quantifier is used on the variable, otherwise it is free.

In $\forall x,p(x,y)$, x is bound but y is free.

Another Example

Remember the example:

age(s,x) ≡ *'s* is *x* years old'

adult(*s*) ≡ *'s* is at least 18 years old*'*

How do we define *adult*(*s*) using logic?

Another Example

Remember the example:

age(s,x) ≡ *'s* is *x* years old'

adult(*s*) ≡ *'s* is at least 18 years old*'*

How do we define *adult*(*s*) using logic?

adult(*s*) ≡ *s* has an age $x \wedge x \geq 18$

adult(*s*) ≡ $\exists x,(age(s,x) \wedge x \geq 18)$

Quantifiers and De Morgan

De Morgan's law extends over quantifiers:

$$\exists x, \neg p(x) \equiv \neg(\forall x, p(x))$$
$$\forall x, \neg p(x) \equiv \neg(\exists x, p(x))$$

Example:

$\neg(\exists x, unicorn(x)) \equiv \forall x, \neg unicorn(x)$ ≡ there is no unicorn

Restricting Predicates

Often the range of values for a predicate are restricted:

$\forall x,(p(x) \Rightarrow q(x))$ reads 'for all x of type p, $q(x)$ is true'.
$\exists x,(p(x) \wedge q(x))$ reads 'for some x of type p, $q(x)$ is true'.

Extended De Morgan works for restricted predicates too:

$$\neg(\forall x,(p(x) \Rightarrow q(x))) \equiv \exists x, \neg(p(x) \wedge q(x))$$
$$\neg(\exists x,(p(x) \wedge q(x))) \equiv \forall x, \neg(p(x) \Rightarrow q(x))$$

Commutability

Quantifiers may commutate only in case of similar ones.

Therefore the following are true:

$$\exists x, \exists y, p(x,y) \equiv \exists y, \exists x, p(x,y)$$

$$\forall x, \forall y, p(x,y) \equiv \forall y, \forall x, p(x,y)$$

And the following is NOT true:

$$\forall x, \exists y, p(x,y) \equiv \exists y, \forall x, p(x,y)$$

Witness and Counter-examples

For an existential formula, $\exists x,p(x)$ a witness is a value of x making $p(x)$ true, thereby proving $\exists x,p(x)$ true as a whole.

For a universal formula, $\forall x,p(x)$ a counter-example is a value of x making $p(x)$ false, thereby proving $\forall x,p(x)$ false as a whole.

Uniqueness

The Uniqueness Quantifier $\exists!$: A formula $\exists!x,p(x)$ reads 'there exists exactly one value of x such that $p(x)$ is true' given a certain domain of x.

$\exists!x,p(x)$ can be expressed as:

$$(\exists x\, p(x)) \wedge (\forall y\, \forall z\, (p(y) \wedge p(z) \Rightarrow y{=}z))$$

and more formally as:

$$\exists x\, p(x) \wedge \forall y\, (x{\neq}y) \Rightarrow \neg p(y)$$

More examples with quantifiers

Given the domain of real numbers

What do the following mean:

- $\forall x \exists y\ (x<y)$
- $\forall x \forall y \exists z\ (z = x + y)$
- $\forall x \forall y\ (x>0 \wedge y<0 \Rightarrow x-y>0)$

Write the following using quantifiers

- The product of two positive numbers is positive.
- Every pair of numbers has a product which is a number
- There is no largest number

More examples with quantifiers

Given *loves*(*x, y*) reads "*x* loves *y*" with the domain of people

What do the following mean?

- $\forall x \, \forall y \, \forall z \;\; loves(x, y) \wedge loves(x, z) \Rightarrow z=y$
- $\forall x \, \exists y \; loves(x, y) \wedge \forall z \; (z \neq y) \Rightarrow \neg loves(x, z)$

Using *loves*(*x, y*), define a predicate *unloved*(*x*) which reads "*x* is an unloved person", i.e. there is no one that loves *x.*

Expanding Quantifiers

Suppose there are just two makes m of car (Toyota and Ford) and three colours c of car paint (black, silver and red). Expand out the quantifiers in the following formulae to obtain equivalents using $\wedge$ and $\vee$:

(a) $\forall m, \exists c, \text{available}(m, c)$

(b) $\exists c, \forall m, \text{available}(m, c)$

Is (a) $\Rightarrow$ (b) true? Is (b) $\Rightarrow$ (a) true?

Expand the quantifiers ....

$$\begin{aligned}
&\forall m, \exists c, \text{available}(m, c) \\
&\quad \equiv \quad (\text{available}(Toyota, black) \vee \text{available}(Toyota, silver) \vee \text{available}(Toyota, red)) \wedge \\
&\qquad\quad (\text{available}(Ford, black) \vee \text{available}(Ford, silver) \vee \text{available}(Ford, red)) \\
&\exists c, \forall m, \text{available}(m, c) \\
&\quad \equiv \quad (\text{available}(Toyota, black) \wedge \text{available}(Ford, black)) \vee \\
&\qquad\quad (\text{available}(Toyota, silver) \wedge \text{available}(Ford, silver)) \vee \\
&\qquad\quad (\text{available}(Toyota, red) \wedge \text{available}(Ford, red))
\end{aligned}$$

(b) $\Rightarrow$ (a) is true but not the converse.

Introduction to Inference in FOL

- Substitutions, instantiation and unification
- Introduction to Logic Programming
- Resolution in First Order Logic
- Forward/Backward Chaining in First Order Logic

Substitutions

A substitution θ is a set of the form $\{x_1/y_1, x_2/y_2, x_3/y_3,\ldots, x_n/y_n\}$, where x's are variables and y's are variables or constants, such that when θ is applied to a sentence α, it returns a sentence with all x's replaced with corresponding y's.

E.g. $\theta = \{x/\text{Tom}, y/z\}$

$\alpha = \forall x \forall y\ \text{parent}(x,y) \Rightarrow \text{child}(y,x)$

$\text{Subst}(\theta, \alpha) = \forall z\ \text{parent}(\text{Tom}, z) \Rightarrow \text{child}(z, \text{Tom})$

Logic Programming

Declarative languages such as Prolog are used for logic programming, where rather than carrying out a set of instructions (as in procedural languages such as C, Java etc.), programs make inferences given rules and facts.

Note that Prolog variables are in uppercase and constants/predicates are in lowercase.

Knowledge base:

Mark is Tom's parent

Jill is Tom's Parent

x is y's parent iff y is x's child

Prolog Facts:

Query:

Is Tom Jill's child?

Is Jill Tom's child?

Does Jill have children, who are they?

Does Tom have children, who are they?

Does Tom have parents, who are they?

Prolog Queries:

Logic Programming

Declarative languages such as Prolog are used for logic programming, where rather than carrying out a set of instructions (as in procedural languages such as C, Java etc.), programs make inferences given rules and facts.

Note that Prolog variables are in uppercase and constants/predicates are in lowercase.

Knowledge base:	**Prolog Facts:**
Mark is Tom's parent	parent(mark, tom)
Jill is Tom's Parent	parent(jill, tom)
x is y's parent iff y is x's child	child(X,Y) :- parent(Y,X)
i.e. $\forall x \forall y$ parent(x,y)$\Leftrightarrow$child(y,x)	parent(X,Y) :- child(Y,X)
Query:	**Prolog Queries:**
Is Tom Jill's child?	child(tom, jill)
Is Jill Tom's child?	child(jill, tom)
Does Jill have children, who are they?	child(X, jill)
Does Tom have children, who are they?	child(X, tom)
Does Tom have parents, who are they?	parents(X,tom)

Logic Programming

Declarative languages such as Prolog are used for logic programming, where rather than carrying out a set of instructions (as in procedural languages such as C, Java etc.), programs make inferences given rules and facts.

Note that Prolog variables are in uppercase and constants/predicates are in lowercase.

Knowledge base:

Mark is Tom's parent

Jill is Tom's Parent

x is y's parent iff y is x's child

i.e. $\forall x \forall y$ parent(x,y)$\Leftrightarrow$child(y,x)

Prolog Facts:

parent(mark, tom)

parent(jill, tom)

child(X,Y) :- parent(Y,X)

parent(X,Y) :- child(Y,X)

Query:

Is Tom Jill's child?

Is Jill Tom's child?

Does Jill have children, who are they?

Does Tom have children, who are they?

Does Tom have parents, who are they?

Prolog Queries:	**Answers:**
child(tom, jill)	Yes
child(jill, tom)	No
child(X, jill)	Yes X=tom, No
child(X, tom)	No
parents(X,tom)	Yes X=mark, X=jill, No

Universal instantiation (UI)

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

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

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

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

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

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

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

  .

  .

  .

Existential instantiation (EI)

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

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

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

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

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

Reduction to propositional inference

Suppose the KB contains just the following:

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

Instantiating the universal sentence in all possible ways, we have:

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

The new KB is **propositionalized**: proposition symbols are

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

Reduction

- Every FOL KB can be propositionalized so as to preserve entailment
  - A ground sentence is entailed by new KB iff entailed by original KB
- Idea: propositionalize KB and query, apply resolution, return result
- Problem: with function symbols, there are infinitely many ground terms
  - e.g., *Father*(*Father*(*Father*(*John*)))

Problems with propositionalization

Propositionalization seems to generate lots of irrelevant sentences

e.g., from:□□

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

it seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant□

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

Unification

We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

Unify(α,β) = θ if αθ = βθ

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

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

Unification

We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

Unify(α,β) = θ if αθ = βθ

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

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

Unification

We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

Unify(α,β) = θ if αθ = βθ

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

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

Unification

We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

Unify(α,β) = θ if αθ = βθ

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

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

Unification

We can get the inference immediately if we can find a substitution θ such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

θ = {x/John,y/John} works

Unify(α,β) = θ if αθ = βθ

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

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

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}
- The first unifier is **more general** than the second
  - It places fewer restrictions on the values of the variables
- There is a single **most general unifier** (MGU) that is unique up to renaming of variables
  MGU = { y/John, x/z }

Resolution in First Order Logic

Conversion to Conjunctive Normal Form

Step 1: Eliminate implications

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

$\equiv \forall x \neg[\text{King}(x) \wedge \text{Greedy}(x)] \vee \text{Evil}(x)$

$\equiv \forall x \neg\text{King}(x) \vee \neg\text{Greedy}(x) \vee \text{Evil}(x)$

Step 2: Move $\neg$ inwards

$\neg\exists x,\ P(x)$ becomes $\forall x,\ \neg P(x)$

$\neg\forall x, P(x)$ becomes $\exists x,\ \neg P(x)$

Resolution in First Order Logic

Step 3: Standardise Variables

If you have a sentence $(\forall x\, P(x)) \wedge (\exists x\, Q(x))$ that use the same bound variable name twice, change the name of one of the variables. E.g. to $(\forall x\, P(x)) \wedge (\exists y\, Q(y))$

Step 4: Skolemization

This is the process of removing existential quantifiers and replacing the variables with new constants. E.g.

$\exists x\, P(x)$ becomes P(C), where C is a Skolem constant

However the meaning is completely changed if we use a constant C and turn $\forall y\, \exists x\, P(x, y)$ into $\forall y\, Q(C, y)$.

Therefore we must use a Skolem function F(y) meaning the x for the particular y. i.e. $\forall y\, Q(F(x), y)$.

Resolution in First Order Logic

Step 5: Drop Universal Quantifiers

$\forall$x King(x) $\wedge$ Greedy(x) $\Rightarrow$ Evil(x) becomes:

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

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

E.g. $(p \wedge q) \vee r$ becomes $(p \vee r) \wedge (q \vee r)$

An example

The law says 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 Colonel West, who is American.

Prove that West is a criminal.

The sentences

"... 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." The sentence $\exists x\ Owns(Nono,x) \wedge Missile(x)$ is transformed into two definite clauses by Existential Elimination, introducing a new constant M_1:

$$Owns(Nono, M_1)$$

$$Missile(M_1)$$

"All of its missiles were sold to it by Colonel West":

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

We will also need to know that missiles are weapons:

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

and we must know that an enemy of America counts as "hostile":

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

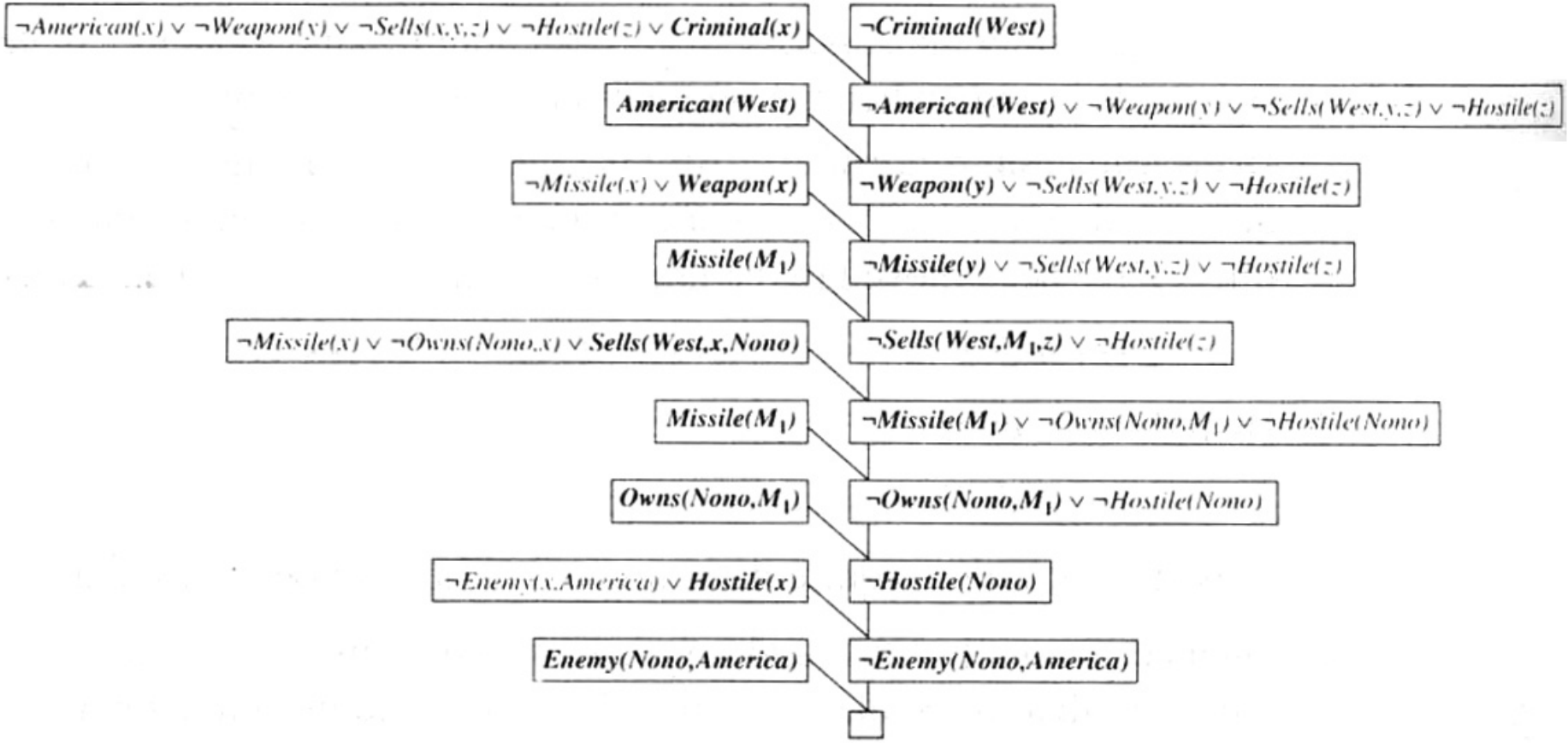
"West, who is American ...":

$$American(West)\ .$$

"The country Nono, an enemy of America ...":

$$Enemy(Nono, America)\ .$$

The Resolution



Forward Chaining In First Order Logic

"... it is a crime for an American to sell weapons to hostile nations":

$$American(x) \land Weapon(y) \land Sells(x, y, z) \land Hostile(z) \Rightarrow Criminal(x) . \quad (9.3)$$

"Nono ... has some missiles." The sentence $\exists x\ Owns(Nono, x) \land Missile(x)$ is transformed into two definite clauses by Existential Elimination, introducing a new constant M_1:

$$Owns(Nono, M_1) \quad (9.4)$$

$$Missile(M_1) \quad (9.5)$$

"All of its missiles were sold to it by Colonel West":

$$Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono) . \quad (9.6)$$

We will also need to know that missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x) \quad (9.7)$$

and we must know that an enemy of America counts as "hostile":

$$Enemy(x, America) \Rightarrow Hostile(x) . \quad (9.8)$$

"West, who is American ...":

$$American(West) . \quad (9.9)$$

"The country Nono, an enemy of America ...":

$$Enemy(Nono, America) . \quad (9.10)$$

Forward Chaining In First Order Logic

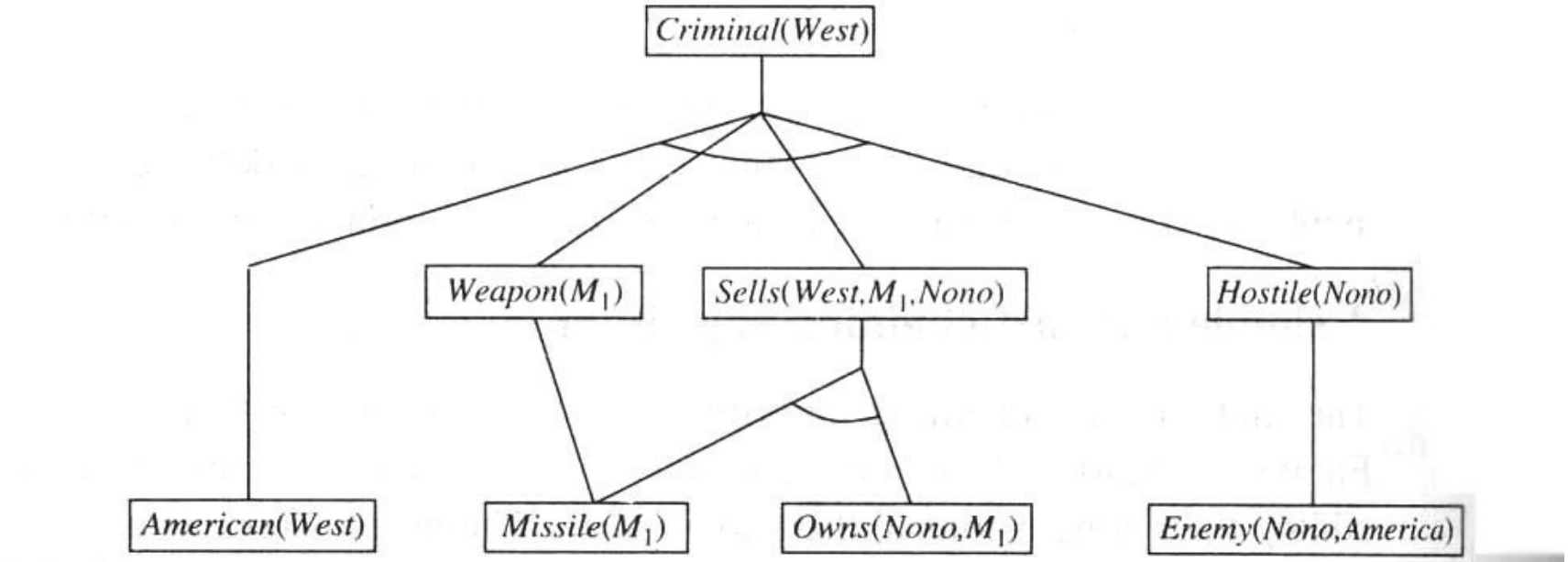


Figure 9.4 The proof tree generated by forward chaining on the crime example. The initial facts appear at the bottom level, facts inferred on the first iteration in the middle level, and facts inferred on the second iteration at the top level.

- On the first iteration, rule (9.3) has unsatisfied premises.
  Rule (9.6) is satisfied with $\{x/M_1\}$, and $Sells(West, M_1, Nono)$ is added.
  Rule (9.7) is satisfied with $\{x/M_1\}$, and $Weapon(M_1)$ is added.
  Rule (9.8) is satisfied with $\{x/Nono\}$, and $Hostile(Nono)$ is added.

Backward Chaining In First Order Logic

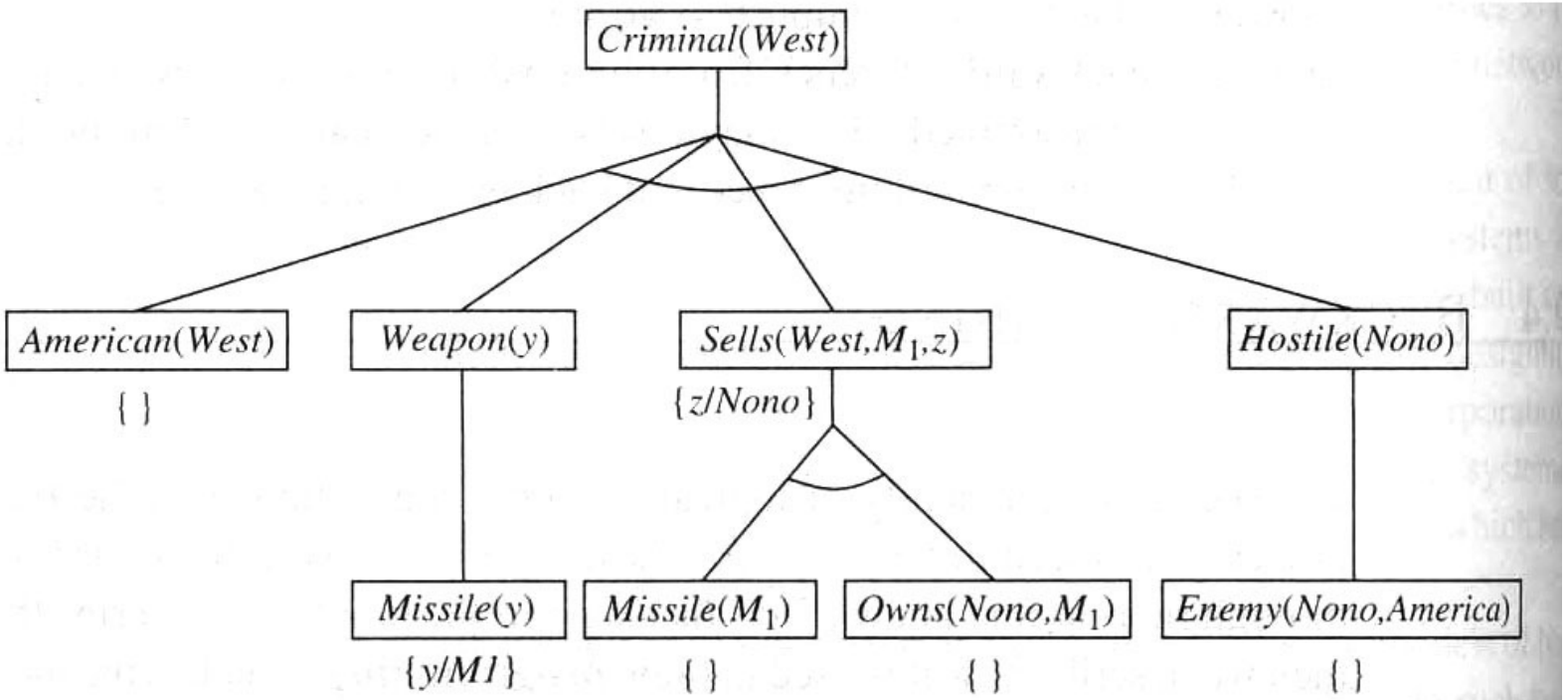


Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally $Hostile(z)$, z is already bound to $Nono$.