CS 561: Artificial Intelligence

First Order Logic

Pros and cons of propositional logic

☺Propositional logic is declarative and not procedural (domain-specific) like programming languages

☺Procedural

- ☺ lacks a general mechanism for deriving facts
- ☺each update to a data structure is done by a domain-specific procedures whose details are derived by the programmer from his or her own knowledge of the domain

☺Declarative

- ☺Knowledge and inference are separate
- ☺Inference is entirely domain-independent

☺ Propositional logic allows partial/disjunctive/negated information

- (unlike most data structures and databases)

Pros and cons of propositional logic

☺ Propositional logic is compositional:

- meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

☺ Meaning in propositional logic is context-independent

- (unlike *natural language, where meaning depends on context*)

Pros and cons of propositional logic

- Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "person-x" is mortal "
    - except by writing one sentence for each person

- propositional logic assumes the world contains facts
  - Either holds true or false

Example

- Consider the problem of representing the following information:
  - Every person is mortal.
  - Confucius is a person.
  - Confucius is mortal.
- How can these sentences be represented so that we can infer the third sentence from the first two?

Example cont.

- In PL, create propositional symbols to stand for all or part of each sentence.
  - P = “person”; Q = “mortal”; R = “Confucius”
- so the above 3 sentences are represented as:
  - P => Q; R => P; R => Q
- Although the third sentence is entailed by the first two, we needed an explicit symbol, R, to represent an individual, Confucius, who is a member of the classes “person” and “mortal.”
- To represent other individuals we must introduce separate symbols for each one, with means for representing the fact that all individuals who are “people” are also "mortal.”

Proofs in propositional calculus

If it is sunny today, then the sun shines on the screen. If the sun shines on the screen, the blinds are brought down. The blinds are not down.

Is it sunny today?

P: It is sunny today.

Q: The sun shines on the screen.

R: The blinds are down.

Premises: $P \rightarrow Q$, $Q \rightarrow R$, $\neg R$

Question: P

Prove using a truth table

Variables			Premises			Trial Conclusions	
P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$\neg R$	P	$\neg P$
T	T	T	T	T	F	T	F
T	T	F	T	F	T	T	F
T	F	T	F	T	F	T	F
T	F	F	F	T	T	T	F
F	T	T	T	T	F	F	T
F	T	F	T	F	T	F	T
F	F	T	T	T	F	F	T
F	F	F	T	T	T	F	T

Propositional calculus is cumbersome

If it is sunny today, then the sun shines on the screen. If the sun shines on the screen, the blinds are brought down. The blinds are not down.

Is it sunny today?

- - -

If it is sunny on a particular day, then the sun shines on the screen. If the sun shines on the screen on a particular day, the blinds are brought down. The blinds are not down today.

Is it sunny today?

Represent in predicate calculus

If it is sunny on a particular day, then the sun shines on the screen [on that day]. If the sun shines on the screen on a particular day, the blinds are down [on that day].
The blinds are not down today.

Is it sunny today?

Premises:

$\forall$D sunny (D)$\rightarrow$ screen-shines (D)

$\forall$D screen-shines(D) $\rightarrow$ blinds-down(D)

$\neg$ blinds-down (today)

Question: sunny(today)

First-order logic

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

FOPL

Declarative, Context-independent and Unambiguous compositional semantics of PL + representational ideas of NLP

Symbols and terms

1. *Truth symbols:* true and false (these are reserved symbols)
2. *Constant symbols:* expressions having the first character lowercase

   E.g., today, fisher
3. *Variable symbols*: symbol expressions beginning with an uppercase character

   E.g., X, Y, Z, Building
4. *Function symbols:* symbol expressions having the first character lowercase

Arity: number of elements in the domain

E.g., mother-of (bill); maximum-of (7,8)

Symbols and terms (cont'd)

function expression: consists of a function constant of arity n, followed by n terms, t_1 ,t_2 ,…, t_n, enclosed in parentheses and separated by commas.

E.g., mother-of(mother-of(joe))

maximum(maximum(7, 18), add_one(18))

term: either a constant, variable, or function expression.

E.g. color_of(house_of(neighbor(joe)))

house_of(X)

Predicates and atomic sentences

Predicate symbols are symbols beginning with a lowercase letter. Predicates are special functions with true/false as the range

Arity: number of arguments

An atomic sentence is a predicate constant of arity n, followed by n terms, t_1 ,t_2 ,. . . , t_n, enclosed in parentheses and separated by commas

The truth values, true and false, are also atomic sentences.

Examples

greater_than(2, 3)

Predicate symbol

term (constant)

mother_of(joe, susan)

mother_of(sister_of(joe), susan)

Predicate Calculus Sentences

Every atomic sentence is a sentence

1. If s is a sentence, then so is its negation, $\neg s$.

If s_1 and s_2 are sentences, then so is their

2. Conjunction, $s_1 \wedge s_2$
3. Disjunction, $s_1 \vee s_2$
4. Implication, $s_1 \rightarrow s_2$
5. Equivalence, $s_1 \equiv s_2$

Predicate calculus sentences (cont'd)

If X is a variable and s is a sentence, then so are

6. $\forall X\ s$
7. $\exists X\ s$

Remember that logic sentences evaluate to true or false, therefore only such objects are atomic sentences. Functions are **not** atomic sentences

Well-formed formula

A well-formed formula (wff) is a sentence containing no "free" variables. i.e. all variables are "bound" by universal or existential quantifiers

- $\forall x\ R(x,y)$ has x bound as a universally quantified variable, but y is free

Interpretation

Let the domain D be a nonempty set.

An *interpretation* over D is an assignment of the entities of D to each of the constant, variable, predicate, and function symbols of a predicate calculus expression:

1. Each constant is assigned an element of D.
2. Each variable is assigned to a nonempty subset of D (*allowable substitutions*).
3. Each function f of arity m is defined (D^m to D)
4. Each predicate of arity n is defined
(D^n to {T,F}).

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*$(term_1,\ldots,term_n)$ is true iff the objects referred to by $term_1,\ldots,term_n$ are in the relation referred to by the *predicate*

Universal quantification

- ∀<*variables*> <*sentence*>

Everyone at IITP is smart:

∀x At(x, IITP) ⇒ Smart(x)

- Ground term: term with no variables

- ∀x *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model

- Roughly speaking, equivalent to the conjunction of instantiations of *P*

At(KingJohn, IITP) ⇒ Smart(KingJohn)
∧ At(Richard, IITP) ⇒ Smart(Richard)
∧ At(IITP, IITP) ⇒ Smart(IITP)
∧ …

A common mistake to avoid

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

- *Take care*: using $\wedge$ as the main connective with $\forall$:

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

  means "*Everyone is at IITP and everyone is smart*"

Existential quantification

- ∃<*variables*> <*sentence*>
- Someone at IITP is smart:

  $\exists x$ At(x, IITP) ∧ Smart(x)

- $\exists x$ *P* is true in a model *m* iff *P* is true with *x* being some possible object in the model
- Roughly speaking, equivalent to the disjunction of instantiations of *P*

  At(KingJohn, IITP) ∧ Smart(KingJohn)
  ∨ At(Richard, IITP) ∧ Smart(Richard)
  ∨ At(IITP,IITP) ∧ Smart(IITP)
  ∨ …

Another common mistake to avoid

- Typically, $\wedge$ is the main connective with $\exists$
- Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \, \text{At}(x, \text{IITP}) \Rightarrow \text{Smart}(x)$$

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

Properties of quantifiers

- $\forall x\ \forall y$ is the same as $\forall y\ \forall x$
- $\exists x\ \exists y$ is the same as $\exists y\ \exists x$
- $\exists x\ \forall y$ is not the same as $\forall y\ \exists x$
- $\exists x\ \forall y$ Loves(x, y)
  - "There is a person who loves everyone in the world"
- $\forall y\ \exists x$ Loves(x, y)
  - "Everyone in the world is loved by at least one person"
- Quantifier duality: each can be expressed using the other

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

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

Equality

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

- E.g., definition of *Sibling* in terms of *Parent*:

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

First-order predicate calculus

First-order predicate calculus allows quantified variables to refer to *objects in the domain of discourse and not to predicates or functions*.

John likes to eat everything

$\forall X \text{ food}(X) \rightarrow \text{likes (john, X)}$

John likes at least one dish Jane likes

$\exists F \text{ food}(F) \wedge \text{ likes (jane, F)} \wedge \text{likes (john, F)}$

John "does" everything Jane does

$\forall P \; P(\text{Jane}) \rightarrow P(\text{john})$ This is not first-order

Using FOL

The kinship domain:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Leftrightarrow Sibling(x,y)$

- One's mother is one's female parent

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

- "Sibling" is symmetric

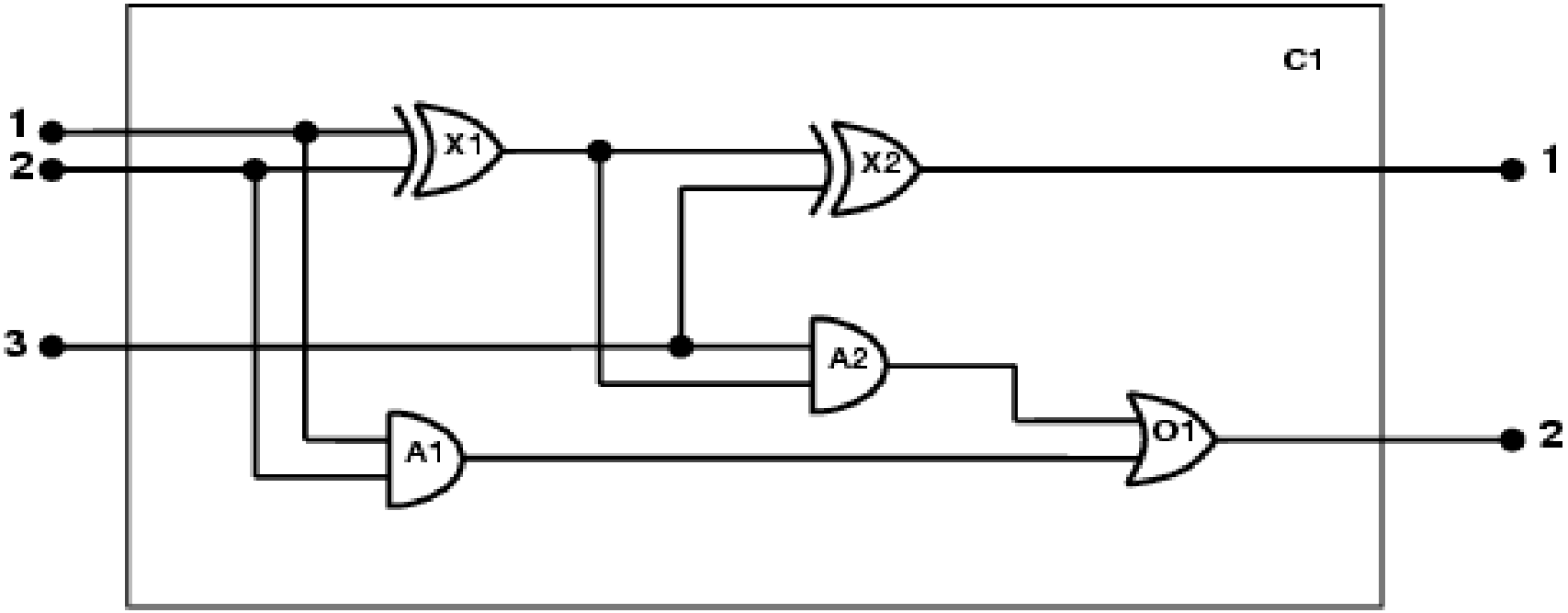
  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

Knowledge Engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

The electronic circuits domain

One-bit full adder



The electronic circuits domain

1. Identify the task
   - Does the circuit actually add properly? (*circuit verification*)
2. Assemble the relevant knowledge

-Composed of wires and gates; Types of gates (AND, OR, XOR, NOT)

-Irrelevant: size, shape, color, cost of gates

3. Decide on a vocabulary

- Alternatives:

  Type(X_1) = XOR

  Type(X_1, XOR)

  XOR(X_1)

The Electronic Circuits Domain

4. Encode general knowledge of the domain
   - $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Signal(t_1) = Signal(t_2)
   - $\forall t$ Signal$(t) = 1 \vee$ Signal$(t) = 0$

     $1 \neq 0$
   - $\forall t_1,t_2$ Connected$(t_1, t_2) \Rightarrow$ Connected(t_2, t_1)
   - $\forall g$ Type$(g) =$ OR $\Rightarrow$ Signal(Out(1,g)) $= 1 \Leftrightarrow \exists n$ Signal(In(n,g)) $= 1$
   - $\forall g$ Type$(g) =$ AND $\Rightarrow$ Signal(Out(1,g)) $= 0 \Leftrightarrow \exists n$ Signal(In(n,g)) $= 0$
   - $\forall g$ Type$(g) =$ XOR $\Rightarrow$ Signal(Out(1,g)) $= 1 \Leftrightarrow$ Signal(In(1,g)) $\neq$ Signal(In(2,g))
   - $\forall g$ Type$(g) =$ NOT $\Rightarrow$ Signal(Out(1,g)) $\neq$ Signal(In(1,g))

The electronic circuits domain

5. Encode the specific problem instance

$\text{Type}(X_1) = \text{XOR}$ $\text{Type}(X_2) = \text{XOR}$

$\text{Type}(A_1) = \text{AND}$ $\text{Type}(A_2) = \text{AND}$

$\text{Type}(O_1) = \text{OR}$

$\text{Connected}(\text{Out}(1,X_1),\text{In}(1,X_2))$

$\text{Connected}(\text{Out}(1,X_1),\text{In}(2,A_2))$

$\text{Connected}(\text{Out}(1,A_2),\text{In}(1,O_1))$

$\text{Connected}(\text{Out}(1,A_1),\text{In}(2,O_1))$

$\text{Connected}(\text{Out}(1,X_2),\text{Out}(1,C_1))$

$\text{Connected}(\text{Out}(1,O_1),\text{Out}(2,C_1))$

$\text{Connected}(\text{In}(1,C_1),\text{In}(1,X_1))$

$\text{Connected}(\text{In}(1,C_1),\text{In}(1,A_1))$

$\text{Connected}(\text{In}(2,C_1),\text{In}(2,X_1))$

$\text{Connected}(\text{In}(2,C_1),\text{In}(2,A_1))$

$\text{Connected}(\text{In}(3,C_1),\text{In}(2,X_2))$

$\text{Connected}(\text{In}(3,C_1),\text{In}(1,A_2))$

The electronic circuits domain

6. Pose queries to the inference procedure

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

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

7. Debug the knowledge base

   May have omitted assertions like $1 \neq 0$

First-order inference

- **Inference rules**
  - Universal instantiation
  - Existential instantiation
  - Can be applied to sentences with quantifiers to obtain sentences without quantifiers

- **First-order inference**
  - Convert the knowledge base (KB) to propositional logic
  - Use propositional inference

Universal instantiation (UI)

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

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

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:

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

- Instantiating the universal sentence in all possible ways, we have:
  $\text{King}(\text{John}) \wedge \text{Greedy}(\text{John}) \Rightarrow \text{Evil}(\text{John})$
  $\text{King}(\text{Richard}) \wedge \text{Greedy}(\text{Richard}) \Rightarrow \text{Evil}(\text{Richard})$
  King(John)
  Greedy(John)
  Brother(Richard, John)

- The new KB is propositionalized: proposition symbols are

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

Reduction Contd...

- 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)))

Reduction contd.

Theorem: Herbrand (1930): If a sentence α is entailed by a FOL KB, it is entailed by a finite subset of the propositionalized KB

Idea: For $n = 0$ to ∞ do
create a propositional KB by instantiating with depth n terms
see if α is entailed by this KB

e.g. constant symbols: *Richard* and *John*
terms with depth 1: (Father(Richard)) and (Father (John))

Problem: *works if α is entailed, loops if α is not entailed*

Theorem: Turing (1936), Church (1936) Entailment for FOL is semi-decidable (algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.)

Problems with Propositionalization

- Propositionalization is inefficient
  - seems to generate lots of irrelevant sentences

E.g., for query: Evil (x)

KB:

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

King(John)

$\forall y$ Greedy(y)

Brother(Richard, John)

Perverse inference: King (Richard) $\wedge$ Greedy(Richard) $\Rightarrow$ Evil(Richard)

Irrelevant facts: King (Richard), Greedy(Richard), Evil(Richard)

But, Evil (John)-perfect for human being

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

First-order inference rule

- If there is some substitution θ that makes the premise of the implication identical to sentences already in the KB then we can assert the conclusion of implication after applying
  - {x/John} achieves the aim

- Let us assume that "*everyone is greedy* "rather than knowing *Greedy (John)*
  - ∀y Greedy(y)

We can still conclude that Evil(John):

-John is king

-John is greedy as everyone is greedy

- Apply substitution {x/John, y/John} to

  premises: King (x) and Greedy (y) === conclusion of implication can be

  KB: King (John) and Greedy (John) inferred

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta = p_i\,\theta$ for all i

p_1' is $King(John)$ $\quad$ p_1 is $King(x)$

p_2' is $Greedy(y)$ $\quad$ p_2 is $Greedy(x)$

θ is $\{x/John, y/John\}$ $\quad$ q is $Evil(x)$

$q\,\theta$ is $Evil(John)$

- GMP used with KB of definite clauses (exactly one positive literal)
- All variables assumed universally quantified

Unification

- **Unification**-Process of finding substitutions that make different logical expressions identical
- 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

- Unification-Process of finding substitutions that make different logical expressions identical
- 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

- Unification-Process of finding substitutions that make different logical expressions identical
- 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

- Unification-Process of finding substitutions that make different logical expressions identical
- 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

- Unification-Process of finding substitutions that make different logical expressions identical
- 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

- Complication
  - Returns substitutions that make two arguments same
  - But, could be more than one such unifier

- To unify *Knows(John, x)* and *Knows(y, z)*,
  $\theta = \{y/John, x/z\}$ or $\theta = \{y/John, x/John, z/John\}$

- The first unifier is more general than the second
  - Second one can be obtained from the first by additional substitution $\{z/John\}$
  - 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
  $MGU = \{ y/John, x/z \}$

First order definite clauses

- Resembles with propositional horn clauses
  - Disjunctions of literals of which at *most one is positive*
- Definite clause
  - Atomic sentence or
  - Implication whose
    - antecedent is a conjunction of positive literals
    - Consequent is a single positive literal
- E. g. of FO definite clause

  $\text{King}(x) \wedge \text{Greedy}(x) \Rightarrow \text{Evil}(x)$
  $\text{King}(\text{John})$
  $\text{Greedy}(y)$
- Unlike propositional definite clause, FO can include variables
  - Variables: universally quantified
  - General norm is to remove universal quantifiers for writing definite clause

First order definite clauses

- Definite clauses: suitable for use in GMP

Every knowledge base can't be converted into a set of definite clauses because of single positive literal restriction

Example knowledge base

- The law says that it is a crime for an American to sell weapons to hostile nations. The country C, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

- Prove that Col. West is a criminal

Example knowledge base contd.

…

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

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

C … has some missiles, i.e., $\exists x\ Owns(C, x) \land Missile(x)$:

$Owns(C,M_1)$ and $Missile(M_1)$: **definite clauses**

… all of its missiles were sold to it by Colonel West

$Missile(x) \land Owns(C, x) \Rightarrow Sells(West, x, C)$

Missiles are weapons:

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

An enemy of America counts as "hostile":

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

West, who is American …

$American(West)$

The country C, an enemy of America …

$Enemy(C, America)$

Data-log KB: as does not contain any function symbol

Forward chaining algorithm (simple)

- **Overall procedure**
  - Start from the known facts
  - Trigger all rules whose premises are satisfied
  - Add the conclusions of the rules to the known facts
  - Process continues until the query is answered or no new facts are added
  - Fact is not new if it is just a renaming of known fact
- **Renaming**
  - One sentence is a renaming of another if they are identical except for the names of the variables
  - E.g.

  Likes (x, IceCream) and Likes (y, IceCream) are identical

Forward chaining algorithm

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

repeat until *new* is empty
- $new \leftarrow \{\}$
- **for each** sentence r **in** KB **do**
  - $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
  - **for each** θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$ for some $p'_1, \ldots, p'_n$ in KB
    - $q' \leftarrow$ SUBST(θ, q)
    - **if** q' is not a renaming of a sentence already in KB or new **then do**
      - add q' to new
      - $\phi \leftarrow$ UNIFY(q', α)
      - **if** ϕ is not *fail* **then return** ϕ
- add new to KB

return *false*

Example knowledge base contd.

…

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

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

C … has some missiles, i.e., $\exists x\ Owns(C, x) \land Missile(x)$:

$Owns(C,M_1)$ and $Missile(M_1)$: **definite clauses**

… all of its missiles were sold to it by Colonel West

$Missile(x) \land Owns(C, x) \Rightarrow Sells(West, x, C)$

Missiles are weapons:

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

An enemy of America counts as "hostile":

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

West, who is American …

$American(West)$

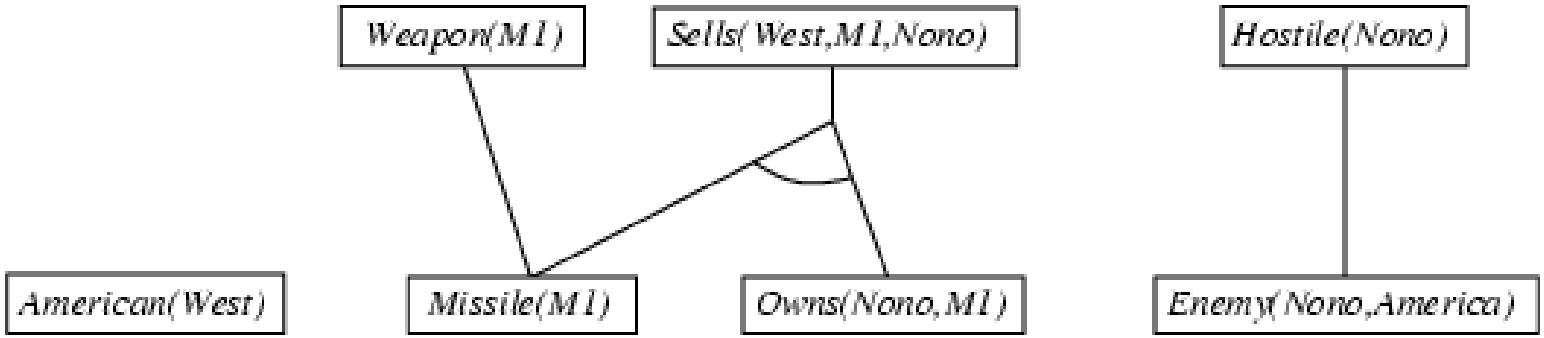
The country C, an enemy of America …

$Enemy(C, America)$

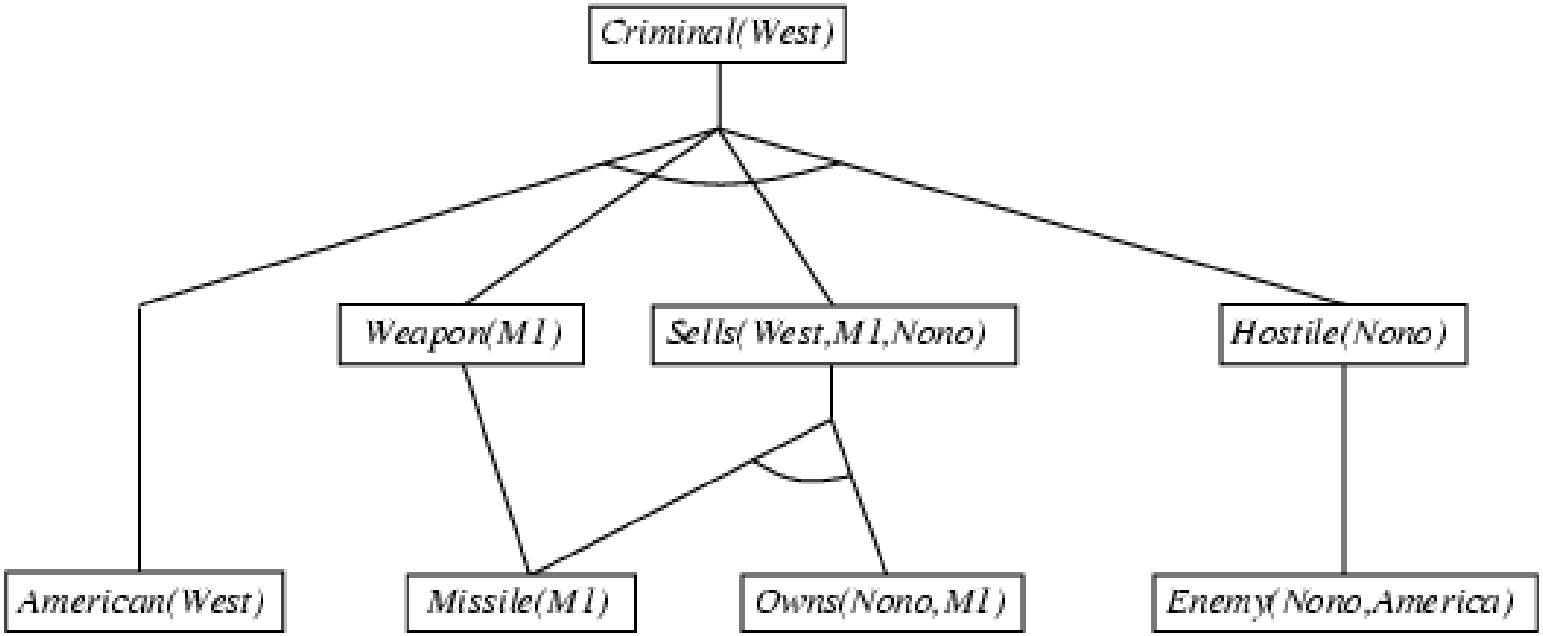
Data-log KB: as does not contain any function symbol

Forward chaining proof (*for a particular value of C, i.e. Nono*)

Forward chaining proof



Forward chaining proof



First order forward chaining

- Fixed-point KB
  - No new inferences are possible

- Fixed point of FO forward chaining vs. Fixed point propositional forward chaining
  - Almost equivalent
  - Difference: FO fixed point can include universally quantified atomic sentences

Properties of forward chaining

- Sound
  - Every inference is just an application of GMP, which is sound
- Complete for definite KB
  - Answers every query whose answers are entailed by any KB of definite clauses
- Datalog KB = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations=number of possible facts that can be added
  - k: maximum arity (# arguments) of any predicate
  - p: # predicates
  - n: # constant symbols
  - Distinct ground facts <= pn^k (algo reaches fixed point after this # iterations)

Efficiency of Forward chaining

- Irrelevant facts
  - Forward chaining makes allowable inferences based on known facts even if they are irrelevant to the goal at hand
- Avoiding irrelevant facts
  - Backward chaining
  - Restrict forward chaining to a selected subset of rules
  - Rewrite the rule set using the information about the goals
    - Allow only relevant variable bindings (*magic set*) during forward inference
  - E.g. goal: Criminal (West)

    new rule: $Magic(x) \wedge American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

    Magic (West) added to the KB

    KB may contain millions of Americans, only Colonel West will be considered during forward inference process: *Hybrid approach*

Backward chaining

- Work backward from the goal, chain through rules to find known facts that support the proof
- Overall procedure
  - Given: set of goals, original query
  - Returns the set of all substitutions satisfying the query
  - List of goals put as a stack (current branch of the proof succeeds if all the goals are satisfied)
  - Take the first goal from the list
  - Find every clause in KB whose *head* unifies with the goal
  - *Body* (premise) added to the goal stack
  - When goal unifies with known fact (clause with a head but no body)
    - No new goals are added
    - Goal is solved

Backward chaining algorithm

function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution $\{\}$
  local variables: ans, a set of substitutions, initially empty

  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST($goals$))
  for each r **in** KB where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    $ans \leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta, \theta'$)) $\cup$ ans
  return ans

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

Example knowledge base Contd

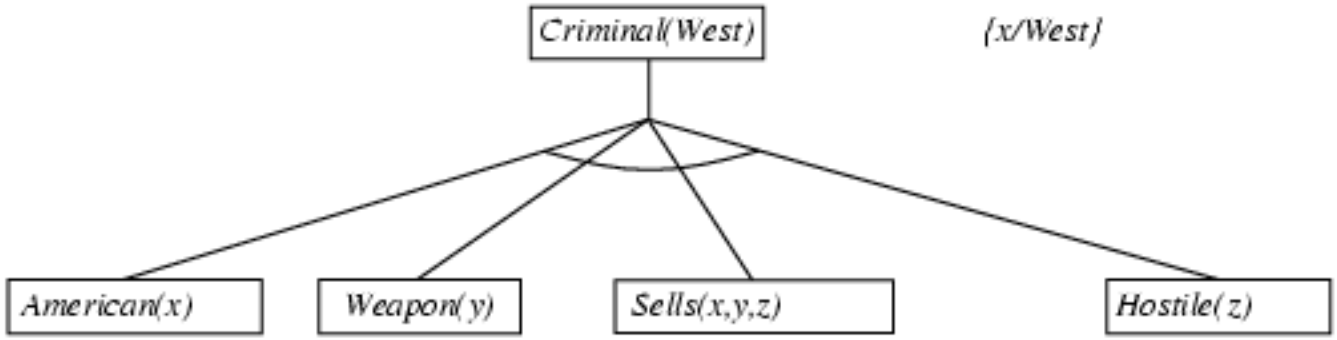
…
it is a crime for an American to sell weapons to hostile nations:
$American(x) \land Weapon(y) \land Sells(x,y,z) \land Hostile(z) \Rightarrow Criminal(x)$
Nono … has some missiles, i.e., $\exists x$ Owns(Nono, x) $\land$ Missile(x):
$Owns(Nono,M_1)$ and $Missile(M_1)$: ***definite clauses***

… all of its missiles were sold to it by Colonel West
$Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
Missiles are weapons:
$Missile(x) \Rightarrow Weapon(x)$
An enemy of America counts as "hostile“:
$Enemy(x, America) \Rightarrow Hostile(x)$
West, who is American …
$American(West)$
The country Nono, an enemy of America …
$Enemy(Nono, America)$

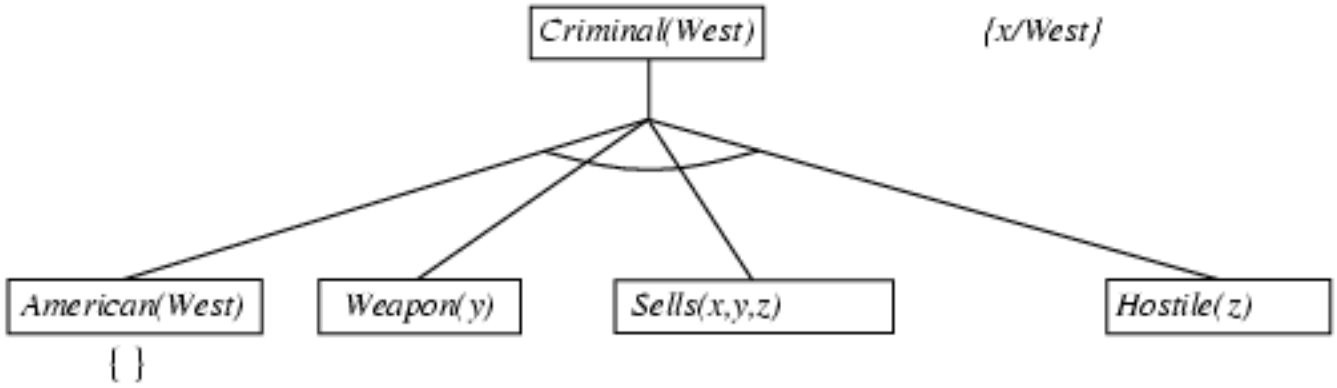
Data-log KB: as does not contain any function symbol

Backward chaining example

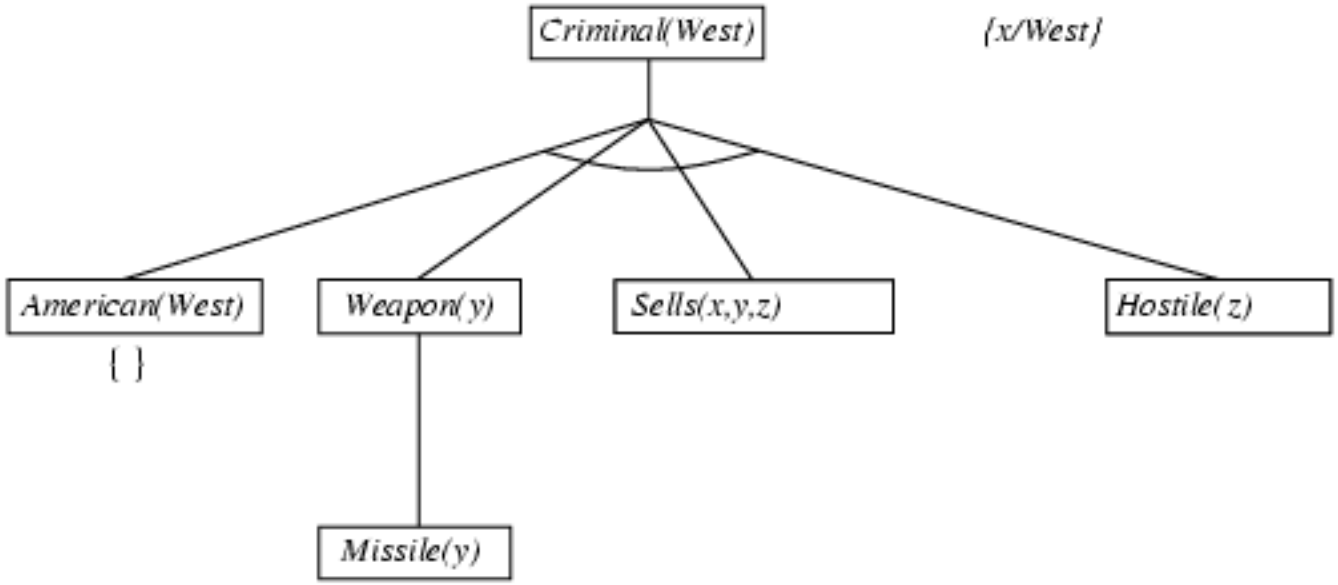
Backward chaining example



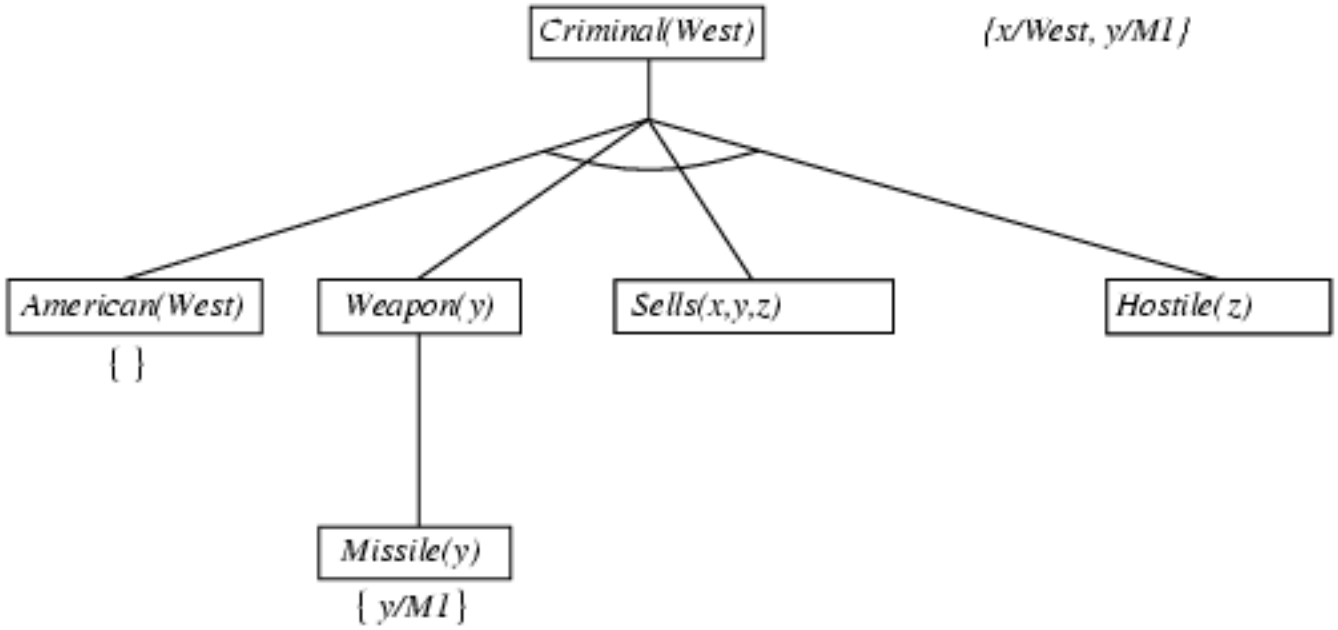
Backward chaining example



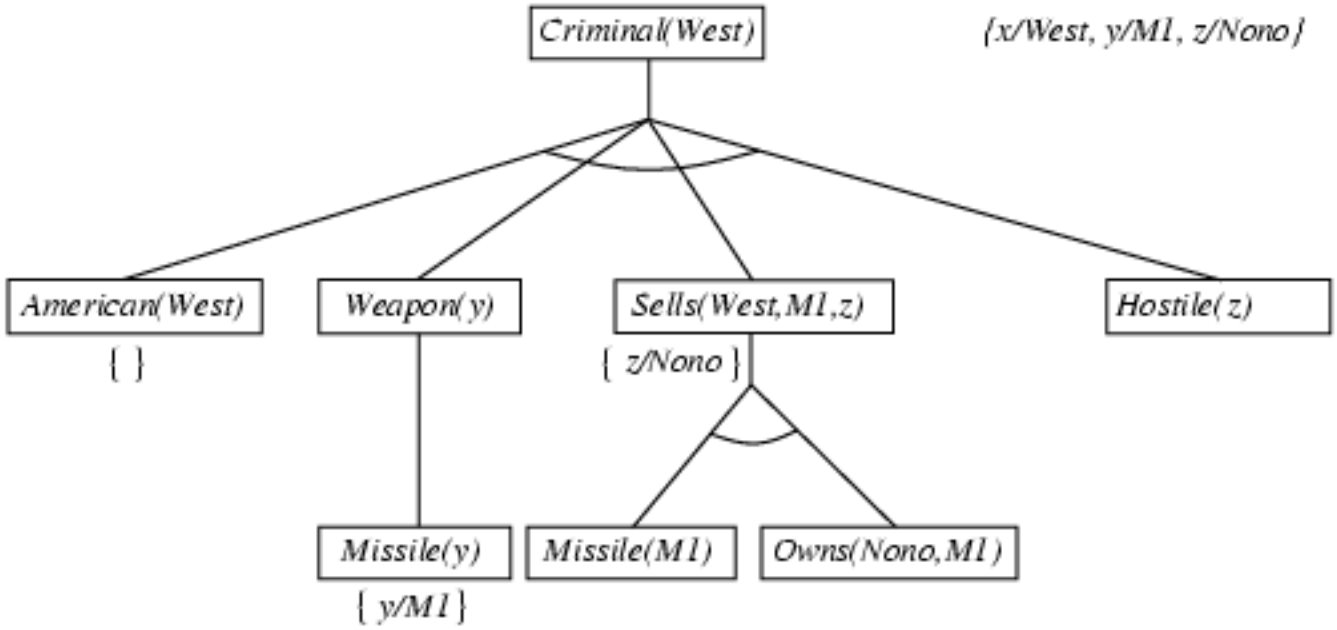
Backward chaining example



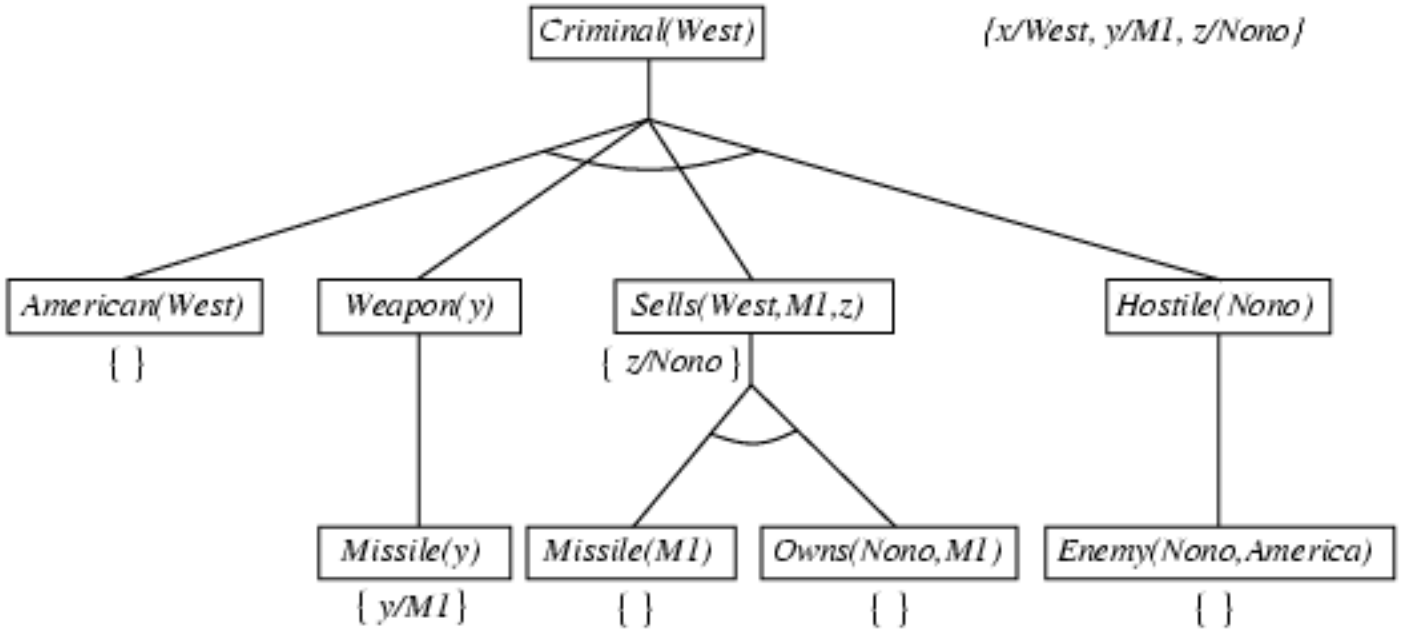
Backward chaining example



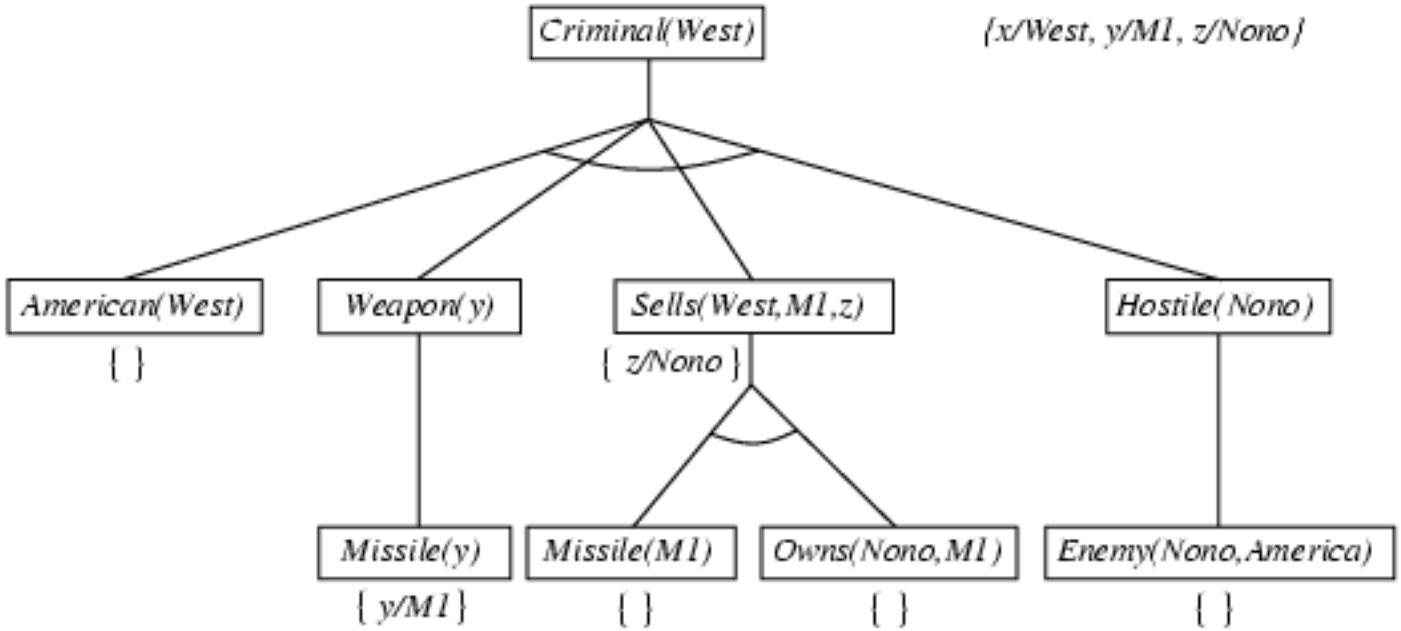
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Equivalent to DFS
- Depth-first recursive proof search: space is linear in size of proof (neglecting the space required for accumulating the solutions)
- Incomplete due to infinite loops

  $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated sub-goals (both success and failure)

  $\Rightarrow$ fix using caching of previous results (extra space)

- Widely used for logic programming

Thank you for your attention