Inference in First-Order Logic

Outline

- Reducing first-order inference to propositional inference
- Lifting and Unification
- Forward chaining
- Backward chaining
- Resolution

FOL to PL

- Make use of propositional inference?

- *Ground sentences* (sentences with no variables)

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

3

FOL to PL

- Make use of propositional inference?
- *Ground sentences* (sentences with no variables)

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

- View ground atomic sentences as propositional symbols!

- How to convert universal and existential quantifiers?

4

Universal Instantiation (UI) Rule

- Replace variable by **ground term** (a term with no variables)
- Every instantiation of a universally quantified sentence is entailed by it:

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

for any ground term *g*

Subst(θ,α) = α θ: the result of applying the **substitution** *(binding list)* θ to the sentence α

  - Given a sentence *S* and a substitution θ, *S*θ denotes the result of plugging θ into *S*; e.g.,

    S = Smarter(x,y), θ = {x/Hillary,y/Bill}

    *S*θ = Smarter(Hillary,Bill)

- E.g., ∀x *King*(*x*) ∧ *Greedy*(*x*) ⇒ *Evil*(*x*) yields:
King(*John*) ∧ *Greedy*(*John*) ⇒ *Evil*(*John*)
King(*Richard*) ∧ *Greedy*(*Richard*) ⇒ *Evil*(*Richard*)
King(*Father*(*John*)) ∧ *Greedy*(*Father*(*John*)) ⇒ *Evil*(*Father*(*John*))

Existential instantiation (EI) Rule

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

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

- E.g., ∃*x* *Crown*(*x*) ∧ *OnHead*(*x*,*John*) yields:

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

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

- Sentences involving nested quantifiers: *skolemization*

EI versus UI

- UI should be applied for *all possible* ground term substitutions
  - the new KB is logically equivalent to the old.

- EI can be applied once to replace the existential sentence
  - the new KB is not equivalent to the old but is satisfiable exactly when the old KB is satisfiable (*inferentially equivalent*).

7

FOL to PL

- First order inference can be done by converting the knowledge base to PL and using propositional inference

- **Propositionalization**
  - Apply UI (using all possible ground term substitutions) and EI
  - View ground atomic sentences as propositional symbols

8

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.

9

Reduction contd.

- *CLAIM*: A ground sentence is entailed by the new KB iff entailed by the original KB.
- *CLAIM*: Every FOL KB can be propositionalized so as to preserve entailment
- *IDEA*: propositionalize KB and query, apply resolution, return result
- → a complete decision procedure for entailment in FOL?

10

Reduction contd.

- *PROBLEM*: with function symbols, there are infinitely many ground terms,
  - e.g., *Father*(*Father*(*Father*(*John*)))
- *THEOREM*: Herbrand (1930). If a sentence α is entailed by an 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

11

Reduction contd.

- *IDEA*: For $n = 0$ to ∞ do
  - create a propositional KB by instantiating with depth-n terms
  - see if α is entailed by this KB
- *PROBLEM*: works if α is entailed, loops forever if α is not entailed
- THEOREM: Turing (1936), Church (1936) Entailment for FOL is semidecidable
  - algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every non-entailed sentence.

12

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!
- From now on, assume EI applied, all variables universally quantified

13

Lifting

- Instead of translating the knowledge base to PL, we can redefine the inference rules into FOL → **lifted** inference rules
- Generalized Modus Ponens

$$\frac{\text{King(John), Greedy(y),}\quad \text{King(x)} \wedge \text{Greedy(x)} \Rightarrow \text{Evil(x)}}{\text{Subst}(\theta, \text{Evil(x)})}$$

- 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

14

Unification

- **Unification**: the process of finding substitutions that make different logical expressions look identical
- Unification is a key component of all FOL inference algorithms
- Unify(α,β) = θ if $\alpha\theta = \beta\theta$, θ is a **unifier**

15

Unification

- Unify(α ,β) = θ if $\alpha\theta = \beta\theta$

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)	

16

Unification

- Unify(α ,β) = θ if $\alpha\theta$ = $\beta\theta$

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)	

Unification

- Unify(α ,β) = θ if $\alpha\theta$ = $\beta\theta$

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)	

Unification

- Unify(α ,β) = θ if $\alpha\theta$ = $\beta\theta$

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)	

19

Unification

- Unify(α ,β) = θ if $\alpha\theta$ = $\beta\theta$

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 by renaming variables, e.g., Knows(z_{17},OJ)
  {x/OJ, z_{17}/John}

20

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,
  α = {y/John, x/z } or
  α' = {y/John, x/John, z/John}
- The first unifier is more general than the second.
- For every unifiable pair, there is a single most general unifier (MGU) that is unique up to renaming of variables.
  MGU = { y/John, x/z }
- Unification is a key component of all FOL inference algorithms
- See textbook for an algorithm for computing MGU

21

Unification

P(x,F(y),B)	P(z,F(w),B)
P(x,F(y),B)	Q(z,F(w),B)
P(x,B)	P(F(x),B)
P(y,B)	P(F(x),B)
P(G(x),B)	P(F(x),B)
P(x,A)	P(x,B)

22

Inference in FOL

Based on lifted inference rules

- Generalized Modus Ponens
  - Forward chaining
  - Backward chaining
- Resolution

23

Generalized Modus Ponens (GMP)

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \quad \text{where } p_i'\theta = p_i\theta \text{ for all } i$$

p_1' is *King*(*John*) p_1 is *King*(*x*)
p_2' is *Greedy*(*y*) p_2 is *Greedy*(*x*)
θ is {x/John,y/John} q is *Evil*(*x*)
$q\theta$ is *Evil*(*John*)

- p, p', q atomic sentences
- All variables assumed universally quantified.

24

Definite Clauses

- Definite clauses: disjunction of literals with exactly one positive literal or

(conjunction of positive literals) ⇒ a positive literal

  King(x) ∧ Greedy(x) ⇒ Evil(x)
  King(John)
  Greedy(y)
  Brother(Richard,John)

- All variables assumed universally quantified.
- GMP is complete for KB of definite clauses
  - Forward chaining
  - Backward chaining

25

Example knowledge base

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

- Prove that Col. West is a criminal

26

Example knowledge base contd.

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

27

Example knowledge base contd.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles

28

Example knowledge base contd.

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

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

Nono … has some missiles,

i.e., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

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

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

29

Example knowledge base contd.

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

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

Nono … has some missiles,

i.e., $\exists x\ Owns(Nono,x) \wedge Missile(x)$:

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

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

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

Missiles are weapons:

30

Example knowledge base contd.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles,

i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

The country Nono, an enemy of America …

31

Example knowledge base contd.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles,

i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

The country Nono, an enemy of America …

Enemy(Nono,America)

An enemy of America counts as "hostile":

32

Example knowledge base contd.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles,

i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

The country Nono, an enemy of America …

Enemy(Nono,America)

An enemy of America counts as "hostile":

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

33

Example knowledge base contd.

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

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

Nono … has some missiles,

i.e., ∃x Owns(Nono,x) ∧ Missile(x):

Owns(Nono,M_1) and Missile(M_1)

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

Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)

Missiles are weapons:

Missile(x) ⇒ Weapon(x)

The country Nono, an enemy of America …

Enemy(Nono,America)

An enemy of America counts as "hostile":

Enemy(x,America) ⇒ Hostile(x)

West, who is American …

American(West)

34

Forward chaining algorithm

Ideas

- Starting from the known facts (atomic sentences)
- Trigger all the rules whose premises are satisfied
  - If premises unify with some facts under some substitution
- Add their conclusions to the known facts
- Repeat until the query is answered or no new facts are added

Linear in PL; linear in FOL? Guaranteed to stop in FOL?

35

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*

Properties of forward chaining

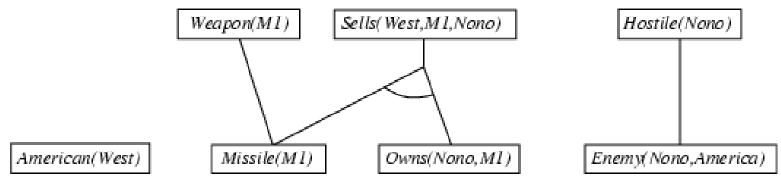
- Sound and complete for first-order definite clauses.
  - Answers every query whose answers are entailed by KB
- FC terminates for first-order definite clauses with *no functions* (e.g. crime KB) in finite number of iterations (why?)
- May not terminate in general definite clauses with functions if α is not entailed
  - This is unavoidable: entailment with definite clauses is also semidecidable

37

Forward chaining example

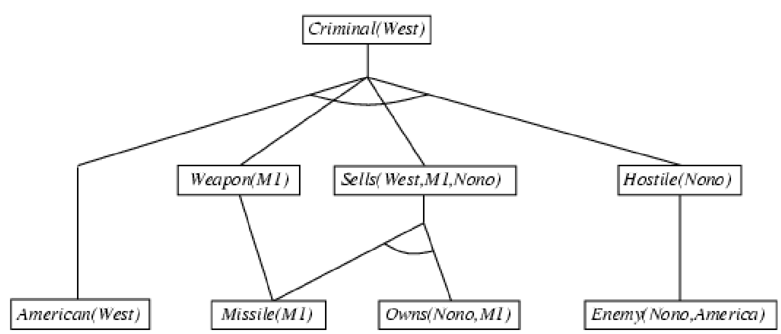
38

Forward chaining example



39

Forward chaining example

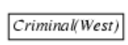


40

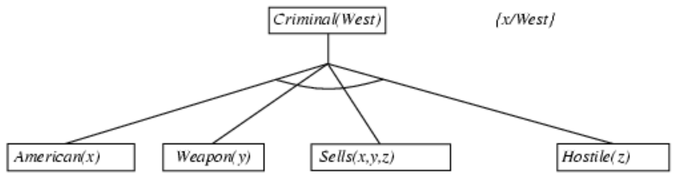
Backward chaining

- Idea: work backwards from the query *q*:
  - to prove *q* by BC,
    - check if *q* is known already, or
    - prove by BC all premises of some rule concluding *q*
- Can be implemented as a recursive depth-first AND/OR search
- Sophisticated techniques developed for efficient implementation

Backward chaining example

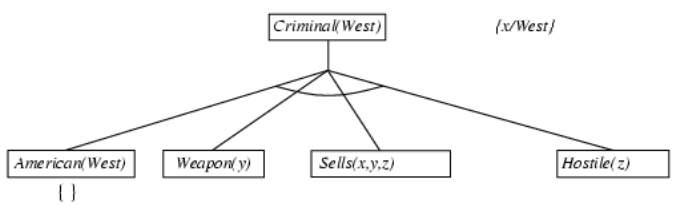


Backward chaining example



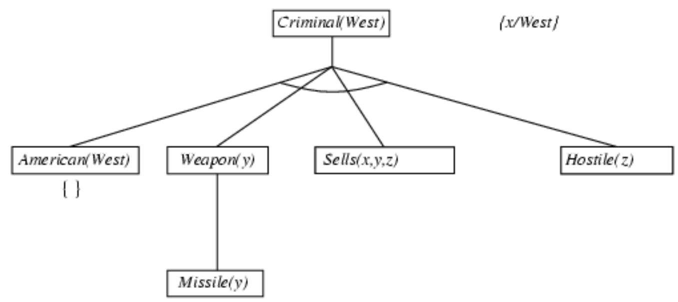
43

Backward chaining example



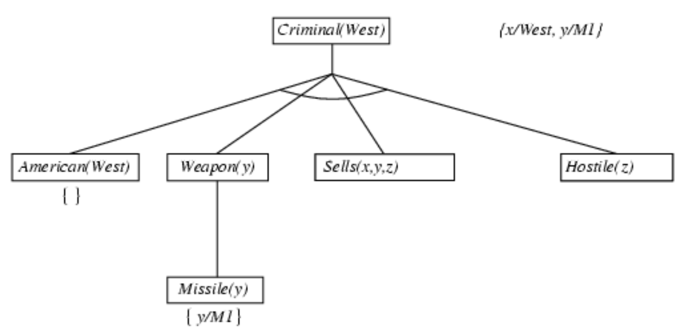
44

Backward chaining example



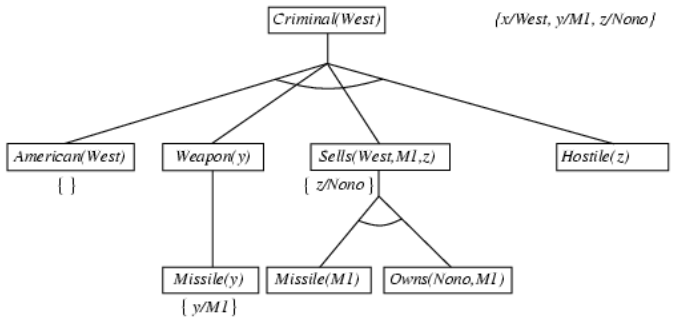
45

Backward chaining example



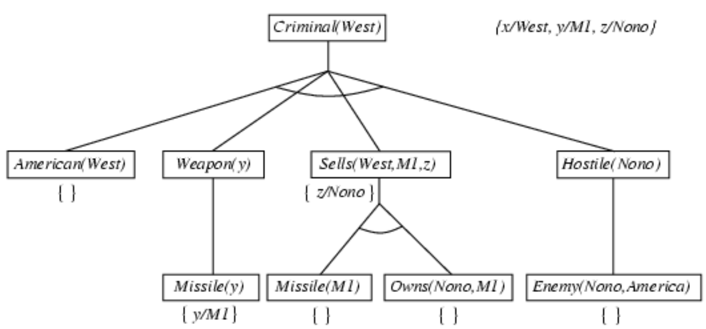
46

Backward chaining example



47

Backward chaining example



48

Logic programming: Prolog

- Backward chaining widely used for logic programming
- Prolog is the most widely used logic programming languages
  - Many expert systems have been written in Prolog
- *BASIS:* backward chaining with definite clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
- Prolog programs are sets of definite clauses in a notation different from standard FOL
  = head :- literal_1, … literal_n.

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z),
                    hostile(Z).
```

- Has features beyond standard FOL

Logic programming

- Logic programming
  - Identify problem
  - Assemble information
  - <coffee break>
  - Encode info in KB
  - Encode problem instances as facts
  - Ask queries
  - Find false facts.
- Procedural programming
  - Identify problem
  - Assemble information
  - Figure out solution
  - Program solution
  - Encode problem instance as data
  - Apply program to data
  - Debug procedural errors

Resolution Rule

- **Binary resolution** rule:

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

  where $\mathtt{Unify}(\ell_i, \neg m_j) = \theta$.
- The two clauses are assumed to be standardized apart so that they share no variables.
- *Complete* if combining with ***factoring***: reduce two unifiable literals to one and apply the unifier to the entire clause
- For example,

$$\frac{\neg Rich(x) \vee Unhappy(x) \qquad Rich(Ken)}{Unhappy(Ken)}$$

  with $\theta = \{x/Ken\}$

51

Conversion to CNF

Every sentence in FOL can be converted into an inferentially equivalent CNF sentence

- Everyone who loves all animals is loved by someone:
  $\forall x\, [\forall y\, Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\, Loves(y,x)]$
- Eliminate biconditionals and implications
  $\forall x\, [\neg\forall y\, \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\, Loves(y,x)]$
- Move $\neg$ inwards: $\neg\forall x\, p \equiv \exists x\, \neg p$, $\neg\exists x\, p \equiv \forall x\, \neg p$
  $\forall x\, [\exists y\, \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\, Loves(y,x)]$
  $\forall x\, [\exists y\, \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\, Loves(y,x)]$
  $\forall x\, [\exists y\, Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\, Loves(y,x)]$

52

Conversion to CNF contd.

- Standardize variables: each quantifier should use a different one:
  $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$
- *Skolemize*: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
  $\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$
- Drop universal quantifiers:
  $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$
- Distribute $\vee$ over $\wedge$:
  $[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

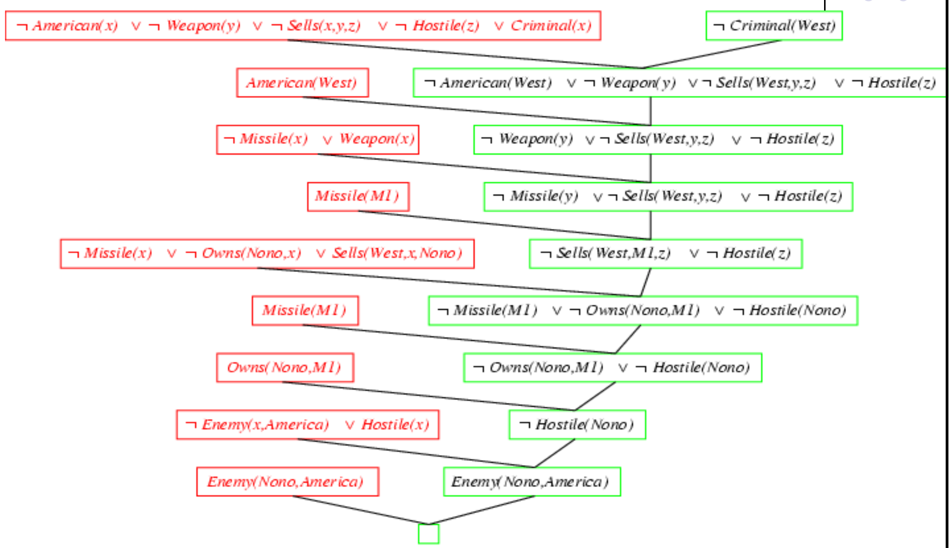
53

Resolution Algorithm

- The resolution algorithm is identical to the PL case: proof by contradiction, i.e., show $KB \wedge \neg \alpha$ unsatisfiable
- Convert $KB \wedge \neg\alpha$ to CNF
- Apply the resolution rule to $CNF(KB \wedge \neg\alpha)$, until the empty clause is generated (Hence KB entail α) or no more possible application of the rule or ?
- Refutation complete for FOL:
  - if KB entails α, refutation will prove it, otherwise, the refutation procedure may not terminate

54

Resolution proof: definite clauses



Resolution Algorithm

- Resolution provides a complete proof system for FOL
- Several strategies exist that improve the efficiency of the proof
- Efficient resolution-based theorem provers have been used to prove mathematical theorems and to verify and synthesize software and hardware designs

56