




# Artificial Intelligence Programming

## *Inference*

Chris Brooks

Department of Computer Science  
University of San Francisco



# Logic-based agents

- The big picture: our agent needs to make decisions about what to do in a complex environment.
- In order to do this, it needs a model of the world.
- Logic is a language for representing this model.
- Our agent will maintain and update a *knowledge base* (KB) which stores this model.
- Our agent will use *inference* to derive new facts that are added to the KB.
- These new facts will help our agent make decisions.

# Logic Summary

- Recall that propositional logic concerns:
  - facts about the world that are true or false.
  - These facts can be used to construct sentences using logical connectives ( $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ )
  - Example:  $P_{1,1} \vee P_{2,2}$ .
- We can convert these sentences to CNF and use resolution as an inference mechanism.
  - Resolution is sound and complete.

# Logic Summary

- A big weakness of propositional logic is its closed-world assumption
  - No way to talk about objects that cannot be named.
- Also, no way to specify classes or relations
- In practice, this leads to an exponential number of sentences in the agent's KB.

# Logic Summary

- First order logic addresses these issues by providing variables and quantification over these variables.
  - FOL works with objects and relations between them, which form atomic sentences.
  - Atomic sentences can be joined with logical connectives. objects.
  - **Example:**  $Siblings(Bart, Lisa), \forall x likes(x, Marge) \Rightarrow likes(x, homer)$

# Quantifiers and variables

- $\exists$  is the symbol for existential quantification
  - It means that the sentence is true for at least one element in the domain.
  - $\exists x \text{ female}(x) \wedge \text{playsSaxophone}(x)$
  - What would happen if I said  
 $\exists x \text{ female}(x) \Rightarrow \text{playsSaxophone}(x)$ ?

# Quantifiers

- In general,  $\Rightarrow$  makes sense with  $\forall$  ( $\wedge$  is usually too strong).
- $\wedge$  makes sense with  $\exists$  ( $\Rightarrow$  is generally too weak.)
- Some examples:
  - One of the Simpsons works at a nuclear plant.
  - All of the Simpsons are cartoon characters.
  - There is a Simpson with blue hair and a green dress.
  - There is a Simpson who doesn't have hair.

# Nesting quantifiers

- Often, we'll want to express more complex quantifications. For example, “every person has a mother”
  - $\forall x \exists y \text{ mother}(x, y)$
  - Notice the scope - for each  $x$ , a different  $y$  is (potentially) chosen.
- What if we said  $\exists y \forall x \text{ mother}(x, y)$ ?
- this is not a problem when nesting quantifiers of the same type.
- $\forall x \forall y \text{ brotherOf}(x, y) \Rightarrow \text{siblingOf}(x, y)$  and  $\forall y \forall x \text{ brotherOf}(x, y) \Rightarrow \text{siblingOf}(x, y)$  are equivalent.
- We often write that as  $\forall x, y \text{ brotherOf}(x, y) \Rightarrow \text{siblingOf}(x, y)$

# Negation

- We can negate quantifiers
  - $\neg\forall x \text{yellow}(x)$  says that it is not true that everyone is yellow.
  - $\exists x\neg\text{yellow}(x)$  has the same meaning - there is someone who is not yellow.
  - $\neg\exists x \text{daughterOf}(Bart, x)$  says that there does not exist anyone who is Bart's daughter.
  - $\forall x \neg\text{daughterOf}(Bart, x)$  says that for all individuals they are not Bart's daughter.
- In fact, we can use DeMorgan's rules with quantifiers just like with  $\wedge$  and  $\vee$ .

# More examples

- A husband is a male spouse
  - $\forall x, y \text{ husband}(x, y) \Leftrightarrow \text{spouse}(x, y) \wedge \text{male}(x)$
- Two siblings have a parent in common
  - $\forall x, y \text{ sibling}(x, y) \Leftrightarrow$   
 $\neg(x = y) \wedge \exists p \text{ Parent}(x, p) \wedge \text{Parent}(y, p)$
- Everyone who goes to Moe's likes either Homer or Barney (but not both)
  - $\forall x \text{ goesTo}(\text{moes}, x) \Rightarrow$   
 $(\text{Likes}(x, \text{Homer}) \Leftrightarrow \neg \text{Likes}(x, \text{Barney}))$

# More examples

- Everyone knows someone who is angry at Homer.
  - $\forall x \exists y \text{ knows}(x, y) \wedge \text{angryAt}(y, \text{homer})$
- Everyone who works at the power plant is scared of Mr. Burns
  - $\forall x \text{ worksAt}(\text{PowerPlant}, x) \Rightarrow \text{scaredOf}(x, \text{burns})$

# Audience Participation

- Everyone likes Lisa.
- Someone who works at the power plant doesn't like Homer. (both ways)
- Bart, Lisa, and Maggie are Marge's only children.
- People who go to Moe's are depressed.
- There is someone in Springfield who is taller than everyone else.
- When a person is fired from the power plant, they go to Moe's
- Everyone loves Krusty except Sideshow Bob
- Only Bart skateboards to school
- Someone with large feet robbed the Quickie-mart.

# Representing useful knowledge in FO

- We can use FOL to represent class/subclass information, causality, existence, and disjoint sets.
- Example: Let's suppose we are interested in building an agent that can help recommend music.
- We want it to be able to reason about musical artists, songs, albums, and genres.
- It would be tedious to enter every bit of information about every artist; instead, we'll enter some rules and let our agent derive entailed knowledge.

# Music example

- $\forall x \text{genre}(x, \text{Punk}) \rightarrow \text{genre}(x, \text{Rock})$  - subclass: all Punk songs are Rock songs.
- $\text{member}(\text{JohnLennon}, \text{Beatles}) \wedge$   
 $\text{member}(\text{PaulMcCartney}, \text{Beatles}) \wedge$   
 $\text{member}(\text{GeorgeHarrison}, \text{Beatles}) \wedge$   
 $\text{member}(\text{RingoStarr}, \text{Beatles}) \wedge \forall x \text{member}(x, \text{Beatles}) \rightarrow$   
 $x \in \{\text{John}, \text{Paul}, \text{George}, \text{Ringo}\}$  - exclusive membership: John, Paul, George, and Ringo are the Beatles.
- $\text{performedBy}(\text{Beatles}, \text{WhiteAlbum})$  The WhiteAlbum is a Beatles album
- $\forall x, y, z \text{member}(x, y) \wedge \text{performedBy}(y, z) \rightarrow \text{playedOn}(x, z)$   
if someone is a member of a group, and that group performed an album, then that person played on that album.

# Music example

- $genre(HolidaysInTheSun, Punk)$  - “Holidays in the Sun” is a Punk song.
- $\forall x genre(x, Rock) \rightarrow likes(Bob, x)$  Bob likes all rock songs.
  - We should be able to infer that Bob will like “Holidays in the Sun”
- $\forall w, x, y, z likes(x, y) \wedge member(z, y) \wedge performedBy(z, w) \rightarrow likes(x, w)$  - If someone likes albums by a band Y, and Z is a member of band Y, then that person will like albums by person Z.

# Inference in Propositional logic

- We talked about two basic mechanisms for performing inference in propositional logic:
  - Forward chaining: Begin with the facts in your KB. Apply inference rules until no more conclusions can be drawn.
  - Backward chaining: Begin with a fact (or its negation) that you wish to prove. Find facts that will justify this fact. Work backwards until you find facts in your KB that support (contradict) the fact you wish to prove.

# Inference in FOL

- Can we do the same sorts of inference with First-order logic that we do with propositional logic?
- Yes, with some extra details.
- Need to keep track of variable bindings (substitution)

# Inference in FOL

- As with propositional logic, we'll need to convert our knowledge base into a canonical form.
- In the simplest approach, we can convert our FOL sentences to propositional sentences. We do this by removing quantification.
  - we leave in predicates for readability, but remove all variables.

# Removing quantifiers

- Universal Instantiation: we can always substitute a ground term for a variable.
- This will typically be a term that occurs in some other sentence of interest.
- Choose a substitution for  $x$  that helps us with our proof.
  - $\forall x \text{LivesIn}(x, \text{Springfield}) \Rightarrow \text{knows}(x, \text{Homer})$
  - Since this is true for all  $x$ , we can substitute  $\{x = \text{Bart}\}$
  - $\text{LivesIn}(\text{Bart}, \text{Springfield}) \Rightarrow \text{knows}(\text{Bart}, \text{Homer})$

# Removing quantifiers

- Existential Instantiation: we can give a name to the object that satisfies a sentence.
  - $\exists x \text{LivesIn}(x, \text{Springfield}) \wedge \text{knows}(x, \text{Homer})$
  - We know this must hold for at least one object. Let's call that object  $K$ .
  - $\text{LivesIn}(K, \text{Springfield}) \wedge \text{knows}(K, \text{Homer})$
  - $K$  is called a *Skolem constant*.
  - $K$  must be unused - gives us a way of referring to an existential object.
- Once we've removed quantifiers, we can use propositional inference rules.

# Propositionalization

- We can replace every existential sentence with a Skolemized version.
- For universally quantified sentences, substitute in *every* possible substitution.
- This will (in theory) allow us to use propositional inference rules.
- Problem: very inefficient!
- This was the state of the art until about 1960.
- Unification of variables is much more efficient.

# Unification

- The key to unification is that we only want to make substitutions for those sentences that help us prove things.
- For example, if we know:
  - $\forall x \text{LivesIn}(x, \text{Springfield}) \wedge \text{WorksAt}(x, \text{PowerPlant}) \Rightarrow \text{knows}(x, \text{Homer})$
  - $\forall y \text{LivesIn}(y, \text{Springfield})$
  - $\text{WorksAt}(\text{MrSmithers}, \text{PowerPlant})$
- We should be able to conclude  $\text{knows}(\text{MrSmithers}, \text{Homer})$  directly.
- Substitution:  $\{x/\text{MrSmithers}, y/\text{MrSmithers}\}$

# Generalized Modus Ponens

- This reasoning is a generalized form of Modus Ponens.
- Basic idea: Let's say we have:
  - An implication of the form  $P_1 \wedge P_2 \wedge \dots \wedge P_i \Rightarrow Q$
  - Sentences  $P'_1, P'_2, \dots, P'_i$
  - a set of substitutions such that
$$P_1 = P'_1, P_2 = P'_2, \dots, P_n = P'_n$$
- We can then apply the substitution and apply Modus Ponens to conclude  $Q$ .
- this technique of using substitutions to pair up sentences for inference is called *unification*.

# Unification

- Our inference process now becomes one of finding substitutions that will allow us to derive new sentences.
- The Unify algorithm: takes two sentences, returns a set of substitutions that unifies the sentences.
  - $WorksAt(x, PowerPlant), WorksAt(Homer, PowerPlant)$  produces  $\{x/Homer\}$ .
  - $WorksAt(x, PowerPlant), WorksAt(Homer, y)$  produces  $\{x/Homer, y/PowerPlant\}$
  - $WorksAt(x, PowerPlant), WorksAt(FatherOf(Bart), y)$  produces  $\{x/FatherOf(Bart), y/PowerPlant\}$
  - $WorksAt(x, PowerPlant), WorksAt(Homer, x)$  fails - x can't bind to both Homer and PowerPlant.

# Unification

- This last sentence is a problem only because we happened to use  $x$  in both sentences.
- We can replace  $x$  with a unique variable (say  $x_{21}$ ) in one sentence.
  - This is called standardizing apart.

# Unification

- What if there is more than one substitution that can make two sentences look the same?
  - $Sibling(Bart, x), Sibling(y, z)$
  - can produce  $\{Bart/y, x/z\}$  or  $\{x/Bart, y/Bart, z/Bart\}$
- the first unification is more general than the second - it makes fewer commitments.
- We want to find the *most general unifier* when performing inference.
  - (This is the heuristic in our search.)

# Unification Algorithm

- To unify two sentences, proceed recursively.
- If either sentence is a single variable, find a unification that binds the variable to a constant.
- Else, call unify in the first term, followed by the rest of each sentence.
  - $Sibling(x, Bart) \wedge PlaysSax(x)$  and  $Sibling(Lisa, y)$
  - We can unify  $Sibling(x, Bart)$  and  $Sibling(Lisa, y)$  with  $\{x/Lisa, y/Bart\}$
- The process of finding a complete set of substitutions is a search process.
  - State is the list of substitutions
  - Successor function is the list of potential unifications and new substitution lists.

# Forward Chaining

- Basic idea: Begin with facts and rules (implications)
- Continually apply Modus Ponens until no new facts can be derived.
- Requires *definite clauses*
  - Implications with positive clauses in the antecedent
  - Positive facts

# Forward Chaining Algorithm



```
while (1) :  
    for rule in rules :  
        if (can_unify(rule, facts)) :  
            fire_rule(rule)  
            assert consequent facts  
if (no rules fired) :  
    return
```

# Example

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

## • Prove that West is a criminal.

- It is a crime for an American to sell weapons to hostile nations.
- 1.  $American(x) \wedge Weapon(y) \wedge Hostile(z) \wedge Sells(x, y, z) \Rightarrow Criminal(x)$
- Nono has missiles.
- 2.  $\exists x Owns(Nono, x) \wedge Missile(x)$
- Use Existential Elimination to substitute  $M_1$  for  $x$
- 3.  $Owns(Nono, M_1)$ , 4.  $Missile(M_1)$

# Example

- Prove that West is a criminal.
  - All Nono's missiles were sold to it by Colonel West.
  - 5.  $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
  - Missiles are weapons.
  - 6.  $Missile(x) \Rightarrow Weapon(x)$
  - An enemy of America is a hostile nation.
  - 7.  $Enemy(x, America) \Rightarrow Hostile(x)$
  - West is American
  - 8.  $American(West)$
  - Nono is an enemy of America
  - 9.  $Enemy(Nono, America)$
- We want to forward chain until we can show all the antecedents of 1.

# Example

- Algorithm: Repeatedly fire all rules whose antecedents are satisfied.
- Rule 6 matches with Fact 4.  $\{x/M_1\}$ . Add  $Weapon(M_1)$ .
- Rule 5 matches with 3 and 4.  $\{x/M_1\}$ . Add  $Sells(West, M_1, Nonono)$
- Rule 7 matches with 9.  $\{x/Nonono\}$ . Add  $Hostile(Nonono)$
- Iterate again.
- Now we can match rule 1 with  $\{x/West, y/M_1, z/Nonono\}$  and conclude  $Criminal(West)$ .

# Analyzing basic forward chaining

- Forward chaining is sound, since it uses Modus Ponens.
- Forward chaining is complete for definite clauses.
- Works much like BFS
- This basic algorithm is not very efficient, though.
  - Finding all possible unifiers is expensive
  - Every rule is rechecked on every iteration.
  - Facts that do not lead to the goal are generated.

# Backward Chaining

- Basic idea: work backward from the goal to the facts that must be asserted for the goal to hold.
- Uses Modus Ponens (in reverse) to focus search on finding clauses that can lead to a goal.
- Also uses definite clauses.
- Search proceeds in a depth-first manner.

# Backward Chaining Algorithm



```
goals = [goal_to_prove]
substitution_list = []

while (not (empty(goals))) :
    goal = goals.dequeue
    unify(goals, substitution_list)
    foreach sentence in KB
        if (unify(consequent(sentence, q)))
            goals.push(antecedents(sentence))
```

# Backward Chaining Example

- To Prove: 1. *Criminal(West)*
- To prove 1, prove:  
2. *American(West)*, 3. *Weapon(y)* 4. *Hostile(z)* 5. *Sells(West, y, z)*
- 2 unifies with 8. To prove:  
3. *Weapon(y)* 4. *Hostile(z)* 5. *Sells(West, y, z)*
- 6 unifies with 3  $\{x/M_1\}$ . To prove:  
.6. *Missile(M<sub>1</sub>)*, 4. *Hostile(z)* 5. *Sells(West, M<sub>1</sub>, z)*
- We can unify 6 with 2  $\{x/M_1\}$  and add *Owns(Nono, M<sub>1</sub>)* to the KB. To prove: 4. *Hostile(z)* 5. *Sells(West, M<sub>1</sub>, z)*
- To prove *Hostile(z)*, prove *Enemy(z, America)*. To prove:  
7. *Enemy(z, America)*, 5. *Sells(West, M<sub>1</sub>, z)*, 6. *Missile(M<sub>1</sub>)*

# Backward Chaining Example

- We can unify 7 with  $Enemy(Nono, America)\{x/Nono\}$ .  
To prove: 5.  $Sells(West, M_1, Nono)$ , 6.  $Missile(M_1)$
- To prove  $Sells(West, M_1, Nono)$ , prove  $Missile(M_1)$  and  $Owns(Nono, M_1)$  To prove:  
8.  $Owns(Nono, M_1)$ , 6.  $Missile(M_1)$
- 8 resolves with the fact we added earlier. To prove:  
6.  $Missile(M_1)$
- $Missile(M_1)$  resolves with 5. The list of goals is empty, so we are done.

# Analyzing Backward Chaining

- Backward chaining uses depth-first search.
- This means that it suffers from repeated states.
- Also, it is not complete.
- Can be very effective for query-based systems
- Most backward chaining systems (esp. Prolog) give the programmer control over the search process, including backtracking.

# Resolution

- Recall Resolution in Propositional Logic:
- $(A \vee C) \wedge (\neg A \vee B) \Rightarrow (B \vee C)$
- Resolution in FOL works similarly.
- Requires that sentences be in CNF.

# Conversion to CNF

- The recipe for converting FOL sentences to CNF is similar to propositional logic.
  1. Eliminate Implications
  2. Move  $\neg$  inwards
  3. Standardize apart
  4. Skolemize Existential sentences
  5. Drop universal quantifiers
  6. Distribute  $\wedge$  over  $\vee$

# CNF conversion example

- Sentence: Everyone who loves all animals is loved by someone.
- Translation:  $\forall x(\forall y \text{Animal}(y) \Rightarrow \text{loves}(x, y)) \Rightarrow (\exists y \text{Loves}(y, x))$
- Eliminate implication
- $\forall x(\neg \forall y \neg \text{Animal}(y) \vee \text{loves}(x, y)) \vee (\exists y \text{Loves}(y, x))$
- Move negation inwards
  - $\forall x(\exists y \neg(\neg \text{Animal}(y) \vee \text{Loves}(x, y))) \vee (\exists y \text{Loves}(y, x))$
  - $\forall x(\exists y \neg \neg \text{Animal}(y) \wedge \neg \text{Loves}(x, y)) \vee (\exists y \text{Loves}(y, x))$
  - $\forall x(\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x, y)) \vee (\exists y \text{Loves}(y, x))$
- Standardize apart
- $\forall x(\exists y \text{Animal}(y) \wedge \neg \text{Loves}(x, y)) \vee (\exists z \text{Loves}(z, x))$

# CNF conversion example

- Skolemize. In this case we need a *Skolem function*, rather than a constant.
- $\forall x(\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, y)) \vee (\text{Loves}((G(x), x))$
- Drop universals
- $(\text{Animal}(F(x)) \wedge \neg \text{Loves}(x, F(x))) \vee (\text{Loves}((G(x), x))$
- Distribute  $\wedge$  over  $\vee$
- $(\text{Animal}(F(x)) \vee \text{Loves}(G(x), x)) \wedge (\neg \text{Loves}(x, F(x))) \vee (\text{Loves}((G(x), x))$

# Resolution Theorem Proving

- Resolution proofs work by inserting the negated form of the sentence to prove into the knowledge base, and then attempting to derive a contradiction.
- A *set of support* is used to help guide search
  - These are facts that are likely to be helpful in the proof.
  - This provides a heuristic.

# Resolution Algorithm

```
sos = [useful facts]
usable = all facts in KB

do
  fact = sos.pop
  foreach fact in usable
    resolve fact with usable
  simplify clauses, remove duplicates and tautologies
  if a clause has no literals :
    return refutation found
until
  sos = []
```

# Resolution Example

- 1.  $\neg American(x) \vee \neg Weapon(y) \vee \neg Sells(x, y, z) \vee \neg Hostile(z) \vee Criminal(x)$
- 2.  $\neg Missile(x) \vee \neg Owns(Nono, x) \vee Sells(west, x, Nono)$
- 3.  $\neg Enemy(x, America) \vee Hostile(x)$
- 4.  $\neg Missile(x) \vee Weapon(x)$
- 5.  $Owns(Nono, M_1)$
- 6.  $Missile(M_1)$
- 7.  $American(West)$
- 8.  $Enemy(Nono, America)$
- 9.  $\neg Criminal(West)$  (added)

# Resolution Example

- Resolve 1 and 9. Add  
10.  $\neg American(West) \vee \neg Weapon(y) \vee \neg Sells(West, y, z) \vee \neg Hostile(z)$
- Resolve 10 and 7. Add 11.  $\neg Weapon(y) \vee \neg Sells(West, y, z) \vee \neg Hostile(z)$
- Resolve 11 and 4. Add 12.  $\neg Missile(y) \vee \neg Sells(West, y, z) \vee \neg Hostile(z)$
- Resolve 12 and 6. Add 13.  $Sells(West, M_1, z) \vee \neg Hostile(z)$
- Resolve 13 and 2. Add 14.  $\neg Missile(M_1) \vee \neg Owns(Nono, M_1) \vee Hostile(Nono)$
- Resolve 14 and 6. Add 15.  $\neg Owns(Nono, M_1) \vee \neg Hostile(Nono)$
- Resolve 15 and 5. Add 16.  $\neg Hostile(Nono)$
- Resolve 16 and 3. Add 17.  $\neg Enemy(Nono, America)$ .
- Resolve 17 and 8. Contradiction!

# Analyzing Resolution Theorem Proving

- Resolution is refutation complete - if a sentence is unsatisfiable, resolution will discover a contradiction.
- Cannot always derive all consequences from a set of facts.
- Can produce nonconstructive proofs for existential goals.
  - Prove  $\exists x \text{likes}(x, \text{Homer})$  will be proven, but without an answer for who  $x$  is.
- Can use full FOL, rather than just definite clauses.

# Efficient Forward Chaining

- Recall that basic forward chaining tries to match every rule with asserted facts on every iteration.
  - This is known as “rules finding facts.”
- In practice, this is not very efficient.
  - The knowledge base does not change drastically between iterations.
  - A few facts are asserted or retracted at each step.
- Also, many rules have similar left-hand sides - can matches be combined?

# Rete

- Rete remembers past partial matches of rules and retains this information across iterations.
- Only new facts are tested against the left-hand side of a rule.
  - “Facts finding rules.”
- Rete compiles a set of rules into a network, where nodes in the network represent tests or conditions on a rule’s left-hand side.
- As facts match these nodes, activation propagates through the network.

# Rete Example

- Let's say we have the following rules:

```
(defrule drink-beer
  (thirsty homer)
=>
  (assert (drink-beer homer)))
```

```
(defrule go-to-moes
  (thirsty homer)
  (at homer ~moes)
=>
  (assert (at homer moes)))
```

```
(defrule happy-homer
  (drink-beer homer)
  (at homer moes)
  (friday)
=>
  (assert (happy homer)))
```

- What would the network look like?

# Rete Example

- Single-input nodes receive a fact, test it, and propagate.
- Two-input nodes group and unify facts that match each parent.
- We can also share nodes among rules, which improves efficiency.
- (watch compilations) in Jess shows the network structure generated by a rule.
  - $+1+1+1+1+1+1+2+2+t$  indicates 6 new 1-input nodes and 2 new 2-input nodes, plus one new terminal node.
  - $=1=1=1=1+2+t$  indicates four shared one input nodes, plus a new two-input node and a terminal node.

# Practical Expert Systems

- Rete inspired a whole generation of expert systems.
  - One of the most successful AI technologies.
  - Nice fit with the way human experts describe problems.
- Made it possible to construct large-scale rule-based systems.
  - Locating oil deposits, controlling factories, diagnosing illness, troubleshooting networks, etc.
- Most expert systems also include a notion of *uncertainty*
  - We'll examine this in the context of Bayesian networks.
- We will see this idea of networks of information again.