Artificial Intelligence Programming

Inference in FOL

Cindi Thompson

Department of Computer Science

University of San Francisco

Models in first-order logic

- Recall that a model is the set of "possible worlds" for a collection of sentences.
- In propositional logic, this meant truth assignments to facts.
- In FOL, models have objects in them.
- The *domain* of a model is the set of objects in that world.
- For example, the Simpsons model might have the domain
  - {Marge, Homer, Lisa, Bart, Maggie}
- We can then specify relations and functions between these objects
  - Married-to(Marge, Homer), Baby(Maggie), Father(Bart) = Homer

Models in FOL, Cont'd

Remember, models also contain enough information to determine the truth value of a given sentence.

- Thus we need an *interpretation* that specifies which objects, relations, and functions are referred to by the constant, predicate, and function symbols.
- 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 $predicate$

Quantifiers and variables

- Quantifiers let us make a statement about some or all objects in the domain.
  - "All of the Simpsons are yellow."
  - $\forall x Simpson(x) \Rightarrow Yellow(x)$
  - "At least one of the Simpsons is a baby."
  - $\exists x Simpson(x) \wedge Baby(x)$

Inference in Propositional logic

We talked about some 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.
- Resolution (by contradiction): convert to CNF, add the negation of the fact you wish to prove to the KB. Apply resolution until you reach a contradiction (empty sentence).

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.
  - $\forall x LivesIn(x, Springfield) \Rightarrow knows(x, Homer)$
  - Since this is true for all x, we can substitute $\{x = Bart\}$
  - $LivesIn(Bart, Springfield) \Rightarrow knows(Bart, Homer)$

Removing quantifiers

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

More on Skolemization

Existential instantiation: can give a name to the object that satisfies a sentence

- Everybody loves someone
- $\forall x, \exists y Loves(x, y)$
- Replace the existential variable with a skolem constant, does it mean what we want?
- $\forall x Loves(x, A)$
- What does this mean?

Skolemization

If an existential quantifier is within the scope of a universal quantifier, we need to replace the existential variable with a function of the universal variable

- $\forall x \exists y Loves(x, y)$
- $\forall x Loves(x, F(x))$
- What does this mean? Is it what we want?

Removing quantifiers

What about a Universal Quantifier within an existential quantifier?

- There is someone whom everyone loves
- $\exists x, \forall y Loves(y, x)$

Removing quantifiers

What about a Universal Quantifier within an existential quantifier?

- There is someone whom everyone loves
- $\exists x, \forall y Loves(y, x)$
- $\forall y Loves(y, K)$
- Seems to be what we want?

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:
  - $\forall x LivesIn(x, Springfield) \wedge WorksAt(x, PowerPlant) \Rightarrow knows(x, Homer)$
  - $\forall y LivesIn(y, Springfield)$
  - $WorksAt(MrSmithers, PowerPlant)$
- We should be able to conclude $knows(MrSmithers, Homer)$ directly.
- Substitution: $\{x/MrSmithers, y/MrSmithers\}$

Generalized Modus Ponens

- This reasoning is a generalized form of Modus Ponens.
- Basic idea: Let's say we have:
  - An implication of the form $P_1 \wedge P_2 \wedge ... \wedge P_i \Rightarrow Q$
  - Sentences $P'_1, P'_2, ..., P'_i$
  - A set of substitutions such that $P_1 = P'_1, P_2 = P'_2, ..., P'_n = P_n$
- 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 :
        #for some set of facts in KB
        if (can_unify(antecedent(rule), facts))
        and consequents not in rules:
            fire_rule(rule) #do substitutions
            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 $Sells(West, M_1, Nono)$
- Rule 7 matches with 9. $\{x/Nono\}$. Add $Hostile(Nono)$
- Iterate again.
- Now we can match rule 1 with $\{x/West, y/M_1, z/Nono\}$ 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]
sub_list = [] # substitution list

while not empty(goals):
    goal = goals.dequeue
    foreach s in KB:
        if unify(consequent(s), goal, sub_list):
           goals.push(subst(sub_list, antecedents(sentence)))
    if no valid subst found:
        return None
```

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:
  $.6.Missile(M_1), 4.Hostile(z) 5.Sells(West, M_1, z)$
- We can unify 6 with 2 $\{x/M_1\}$ and add $Owns(Nono, M_1)$ to the KB. To prove: $4.Hostile(z) 5.Sells(West, M_1, z)$
- To prove $Hostile(z)$, prove $Enemy(z, America)$. To prove:
  $7.Enemy(z, America), 5.Sells(West, M_1, z), 6.Missile(M_1)$

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: $\forall x(\forall y Animal(y) \Rightarrow loves(x, y)) \Rightarrow (\exists y Loves(y, x))$
- Eliminate implication
- $\forall x(\neg\forall y \neg Animal(y) \vee loves(x, y)) \vee (\exists y Loves(y, x))$
- Move negation inwards
  - $\forall x(\exists y \neg(\neg Animal(y) \vee Loves(x, y))) \vee (\exists y Loves(y, x))$
  - $\forall x(\exists y \neg\neg Animal(y) \wedge \neg Loves(x, y))) \vee (\exists y Loves(y, x))$
  - $\forall x(\exists y Animal(y) \wedge \neg Loves(x, y))) \vee (\exists y Loves(y, x))$
- Standardize apart
- $\forall x(\exists y Animal(y) \wedge \neg Loves(x, y))) \vee (\exists z Loves(z, x))$

CNF conversion example

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

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

Resolution Proof 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

Using entire starting KB as sos

- 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

- Resolution is refutation complete - if a sentences 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 $\exists x likes(x, Homer)$ will be proven, but without an answer for who x is.
- Can use full FOL, rather than just definite clauses.

Rule-based Systems

- The if-then form of constructing a knowledge base turns out to be very effective in practice.
  - A good fit with the ways in which human experts describe knowledge.
  - Rules of this form can also be learned.
- Forward chaining also fits well with decision-making and diagnostic scenarios.
  - Given some inputs, decide on the best course of action.

FOL and Inference

We've come a long way!

- Objects and relations are semantic primitives
- Constants, functions, predicates, equality quantifiers
- Inference: Forward and backward chaining, Resolution

Expressive power to handle interesting situations, including machinery needed for basic natural language processing