Artificial Intelligence Programming
Inference in First-order Logic

Chris Brooks

Department of Computer Science
University of San Francisco
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 ($\land$, $\lor$, $\neg$, $\Rightarrow$, $\Leftrightarrow$)
- Example: $P_{1,1} \lor P_{2,2}$.

First order logic concerns:
- Objects and relations between them, which form atomic sentences.
- Atomic sentences can be joined with logical connectives.
- We can introduce variables and quantifiers to refer to indefinite objects.
- Example: $\text{siblings}(\text{Bart, Lisa}), \forall x \text{likes}(x, \text{Marge}) \Rightarrow \text{likes}(x, \text{homer})$
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)
Universal Instantiation: we can always substitute a ground term for a variable.

- $\forall x \text{LivesIn}(x, \text{Springfield}) \Rightarrow \text{knows}(x, \text{Homer})$
- Since this is true for all $x$, we can substitute $\{x = \text{Bart}\}$
- $\text{LivesIn}(\text{Bart}, \text{Springfield}) \Rightarrow \text{knows}(\text{Bart}, \text{Homer})$
13-3: Removing quantifiers

6 Existential Instantiation: we can give a name to the object that satisfies a sentence.
   △ \( \exists x \text{LivesIn}(x, \text{Springfield}) \land \text{knows}(x, \text{Homer}) \)
   △ We know this must hold for at least one object. Let’s call that object \( k \).
   △ \( \text{LivesIn}(k, \text{Springfield}) \land \text{knows}(k, \text{Homer}) \)
   △ \( k \) is called a Skolem constant.
   △ \( k \) must be unused - gives us a way of referring to an existential object.

6 Once we’ve removed quantifiers, we can use propositional inference rules.
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.
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:

\[ \forall x \text{LivesIn}(x, \text{Springfield}) \land \text{WorksAt}(x, \text{PowerPlant}) \Rightarrow \text{knows}(x, \text{Homer}) \]
\[ \forall y \text{LivesIn}(y, \text{Springfield}) \]
\[ \text{WorksAt}(\text{MrSmithers}, \text{PowerPlant}) \]

We should be able to conclude $\text{knows}(\text{MrSmithers}, \text{Homer})$ directly.

Substitution: $\{x/\text{MrSmithers}, y/\text{MrSmithers}\}$
13-6: Generalized Modus Ponens

This reasoning is a generalized form of Modus Ponens.

Basic idea: Let’s say we have:
- An implication of the form \( P_1 \land P_2 \land \ldots \land P_i \Rightarrow Q \)
- Sentences \( P'_1, P'_2, \ldots, P'_i \)
- a set of substitutions such that \( P_1 = P'_1, P_2 = P'_2, \ldots, P'_n = P_n \)

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.
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.

- \(\text{WorksAt}(x, \text{PowerPlant}), \text{WorksAt}(\text{Homer}, \text{PowerPlant})\) produces \(\{x/\text{Homer}\}\).
- \(\text{WorksAt}(x, \text{PowerPlant}), \text{WorksAt}(\text{Homer}, y)\) produces \(\{x/\text{Homer}, y/\text{PowerPlant}\}\)
- \(\text{WorksAt}(x, \text{PowerPlant}), \text{WorksAt}(\text{FatherOf}(\text{Bart}), y)\) produces \(\{x/\text{FatherOf}(\text{Bart}), y/\text{PowerPlant}\}\)
- \(\text{WorksAt}(x, \text{PowerPlant}), \text{WorksAt}(\text{Homer}, x)\) fails - \(x\) can’t bind to both Homer and PowerPlant.
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 $x_{21}$) is one sentence.
   ▲ This is called standardizing apart.
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.
13-10: 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.

- $\text{Sibling}(x, \text{Bart}) \land \text{PlaysSax}(x)$ and $\text{Sibling}(\text{Lisa}, y)$
- We can unify $\text{Sibling}(x, \text{Bart})$ and $\text{Sibling}(\text{Lisa}, y)$ with $\{x/\text{Lisa}, y/\text{Bart}\}$

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.
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
- Jess uses a generalized version of definite clauses.
while (1) :
    for rule in rules :
        if (can_unify(rule, facts)) :
            fire_rule(rule)
            assert consequent facts
    if (no rules fired) :
        return
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.

**6 Prove that West is a criminal.**

- It is a crime for an American to sell weapons to hostile nations.
- \( \text{American}(x) \land \text{Weapon}(y) \land \text{Hostile}(z) \land \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x) \)
- Nono has missiles.
- \( \exists x \text{Owns}(\text{Nono}, x) \land \text{Missile}(x) \)
- Use Existential Elimination to substitute \( M_1 \) for \( x \)
- \( \text{Owns}(\text{Nono}, M_1), 4.\text{Missile}(M_1) \)
5. Prove that West is a criminal.
   ▲ All Nono’s missiles were sold to it by Colonel West.
   ▲ 5.\(Missile(x) \land Owns(Nono, x) \Rightarrow Sells(West, x, Nono)\)
   ▲ 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.\(Enemy(Nono, America)\)

6. We want to forward chain until we can show all the antecedents of 1.
Algorithm: Repeatedly fire all rules whose antecedents are satisfied.

Rule 6 matches with Fact 4. \( \{x/M_1\} \). Add \( \text{Weapon}(M_1) \).

Rule 5 matches with 3 and 4. \( \{x/M_1\} \). Add \( \text{Sells}(West, M_1, Nono) \)

Rule 7 matches with 9. \( \{x/Nono\} \). Add \( \text{Hostile}(Nono) \)

Iterate again.

Now we can match rule 1 with \( \{x/West, y/M_1, z/Nono\} \) and conclude \( \text{Criminal}(West) \).
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.
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.
13-18: 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))
13-19: Backward Chaining Example

To Prove: 1. *Criminal*(West)

To prove 1, prove:

2 unifies with 8. To prove:

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)
13-20: Backward Chaining Example

6. We can unify 7 with $\text{Enemy(Nono, America)} \{x/\text{Nono}\}$. To prove: 5.$\text{Sells(West, M_1, Nono)}$, 6.$\text{Missile(M_1)}$

6. To prove $\text{Sells(West, M_1, Nono)}$, prove $\text{Missile(M_1)}$ and $\text{Owns(Nono, M_1)}$. To prove:

8. $\text{Owns(Nono, M_1)}$, 6.$\text{Missile(M_1)}$

6. 8 resolves with the fact we added earlier. To prove:

6.$\text{Missile(M_1)}$

6. $\text{Missile(M_1)}$ resolves with 5. The list of goals is empty, so we are done.
13-21: 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.
Recall Resolution in Propositional Logic:

\[(A \lor C) \land (\neg A \lor B) \Rightarrow (B \lor C)\]

Resolution in FOL works similarly.

Requires that sentences be in 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 $\vee$
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

$\forall x(\neg \forall y \neg \text{Animal}(y) \lor \text{loves}(x, y)) \lor (\exists y \text{Loves}(y, x))$

Move negation inwards

$\forall x(\exists y \neg (\neg \text{Animal}(y) \lor \text{Loves}(x, y))) \lor (\exists y \text{Loves}(y, x))$

$\forall x(\exists y \neg \text{Animal}(y) \land \neg \text{Loves}(x, y))) \lor (\exists y \text{Loves}(y, x))$

$\forall x(\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y))) \lor (\exists y \text{Loves}(y, x))$

Standardize apart

$\forall x(\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y))) \lor (\exists z \text{Loves}(z, x))$
13-25: CNF conversion example

6 Skolemize. In this case we need a Skolem function, rather than a constant.

6 $\forall x (\text{Animal}(F(x)) \land \neg \text{Loves}(x, y)) \lor (\text{Loves}((G(x), x))$

6 Drop universals

6 $(\text{Animal}(F(x)) \land \neg \text{Loves}(x, F(x)))) \lor (\text{Loves}((G(x), x))$

6 Distribute $\land$ over $\lor$

6 $(\text{Animal}(F(x)) \lor \text{Loves}(G(x), x)) \land (\neg \text{Loves}(x, F(x)))) \lor (\text{Loves}((G(x), x))$
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.
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 = []
13-28: Resolution Example

1. \( \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x, y, z) \lor \neg \text{Hostile}(z) \lor \text{Criminal}(x) \)

2. \( \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{west}, x, \text{Nono}) \)

3. \( \neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x) \)

4. \( \neg \text{Missile}(x) \lor \text{Weapon}(x) \)

5. \( \text{Owns}(\text{Nono}, M_1) \)

6. \( \text{Missile}(M_1) \)

7. \( \text{American}(\text{West}) \)

8. \( \text{Enemy}(\text{Nono}, \text{America}) \)

9. \( \neg \text{Criminal}(\text{West}) \) (added)
13-29: Resolution Example

6 Resolve 1 and 9. Add
   10. \( \neg \text{American}(\text{West}) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \)

6 Resolve 10 and 7. Add 11. \( \neg \text{Weapon}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \)

6 Resolve 11 and 4. Add 12. \( \neg \text{Missile}(y) \lor \neg \text{Sells}(\text{West}, y, z) \lor \neg \text{Hostile}(z) \)

6 Resolve 12 and 6. Add 13. \( \text{Sells}(\text{West}, M_1, z) \lor \neg \text{Hostile}(z) \)

6 Resolve 13 and 2. Add 14. \( \neg \text{Missile}(M_1) \lor \neg \text{Owns}(\text{Nono}, M_1) \lor \text{Hostile}(\text{Nono}) \)

6 Resolve 14 and 6. Add 15. \( \neg \text{Owns}(\text{Nono}, M_1) \lor \neg \text{Hostile}(\text{Nono}) \)

6 Resolve 15 and 5. Add 16. \( \neg \text{Hostile}(\text{Nono}) \)

6 Resolve 16 and 3. Add 17. \( \neg \text{Enemy}(\text{Nono}, \text{America}) \)

6 Resolve 17 and 8. Contradiction!
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 $\exists x \text{likes}(x, \text{Homer})$ will be proven, but without an answer for who $x$ is.

Can use full FOL, rather than just definite clauses.
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 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.
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?
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.
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.