CMSC 471
Fall 2012

Class #13

Thursday, October 11
Logical Inference

Kevin Winner, winnerk1@umbc.edu

Today's Class

- Project language/groups announcement
- Inference in first-order logic
  - Inference rules
  - Forward chaining
  - Backward chaining
  - Resolution
    - Clausal form
    - Unification
    - Resolution as search
- Review HW2
- HW3 questions
- Midterms

Logical Inference

Chapter 9

Some material adopted from notes
by Andreas Geyer-Schulz,
Chuck Dyer, and Lise Getoor

Inference

- Given a KB and a goal sentence, prove that the goal sentence is entailed by the KB
  - In other words, given a KB and a goal sentence, derive the goal sentence from the KB
- 3 main families of inference
  - Forward Chaining
  - Backward Chaining
  - Resolution Theorem Proving

Reminder: Inference Rules for FOL

- Inference rules for propositional logic apply to FOL as well
  - Modus Ponens, And-Introduction, And-Elimination, …
- New (sound) inference rules for use with quantifiers:
  - Universal introduction
  - Universal elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)

Generalized Modus Ponens (GMP)

- Apply modus ponens reasoning to generalized rules
- Combines And-Introduction, Universal-Elimination, and Modus Ponens
  - From $P(c)$ and $Q(c)$ and $(\forall x)(P(x) \land Q(x)) \rightarrow R(x)$ derive $R(c)$

Substitutions

- Substitutions (Bindings)
  - subst(θ, α) denotes the result of applying a set of substitutions defined by θ to the sentence α
  - A substitution list $\theta = \{v_1/t_1, v_2/t_2, ..., v_n/t_n\}$ means to replace all occurrences of variable symbol v_i by term t_i
  - Substitutions are made in left-to-right order in the list
  - subst({x/IceCream, y/Ziggy}, eats(y,x)) = eats(Ziggy, IceCream)

Horn Clauses

- A Horn clause is a sentence of the form:
  $(\forall x)\ P_1(x) \wedge P_2(x) \wedge \ldots \wedge P_n(x) \rightarrow Q(x)$

where

- there are 0 or more P_is and 0 or 1 Q
- the P_is and Q are positive (i.e., non-negated) literals

- Equivalently: $P_1(x) \vee P_2(x) \ldots \vee P_n(x)$ where the P_i are all atomic and *at most one* of them is positive
- Prolog is based on Horn clauses
- Horn clauses represent a *subset* of the set of sentences representable in FOL

Forward Chaining

- Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived
- This defines a **forward-chaining** inference procedure because it moves "forward" from the KB to the goal [eventually]
- Inference using GMP is **complete** for KBs containing **only Horn clauses**

Forward Chaining Example

- KB:
  1. allergies(X) → sneeze(X)
  2. cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  3. cat(Felix)
  4. allergic-to-cats(Lisa)
- Goal:
  - sneeze(Lisa)

Backward Chaining

- **Backward-chaining** deduction using GMP is also **complete** for KBs containing **only Horn clauses**
- Proofs start with the goal query, find rules with that conclusion, and then prove each of the antecedents in the implication
- Keep going until you reach premises
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  - Has already been proved true
  - Has already failed

Backward Chaining Example

- KB:
  - allergies(X) → sneeze(X)
  - cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)
- Goal:
  - sneeze(Lise)

Forward vs. Backward Chaining

- FC is data-driven
  - Automatic, unconscious processing
  - E.g., object recognition, routine decisions
  - May do lots of work that is irrelevant to the goal
- BC is goal-driven, appropriate for problem-solving
  - Where are my keys? How do I get to my next class?
  - Complexity of BC can be much less than linear in the size of the KB

Automating FOL Inference with Resolution

Resolution

- Resolution is a **sound** and **complete** inference procedure for FOL
- Reminder: Resolution rule for propositional logic:
  - $P_1 \vee P_2 \vee \ldots \vee P_n$
  - $\neg P_1 \vee Q_2 \vee \ldots \vee Q_m$
  - Resolvent: $P_2 \vee \ldots \vee P_n \vee Q_2 \vee \ldots \vee Q_m$
- Examples
  - P and $\neg$ P $\vee$ Q : derive Q (Modus Ponens)
  - ($\neg$ P $\vee$ Q) and ($\neg$ Q $\vee$ R) : derive $\neg$ P $\vee$ R
  - P and $\neg$ P : derive False [contradiction!]
  - (P $\vee$ Q) and ($\neg$ P $\vee$ $\neg$ Q) : derive True

Resolution in First-Order Logic

- Given sentences

  $P_1 \vee \ldots \vee P_n$

  $Q_1 \vee \ldots \vee Q_m$

- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ **unify** with substitution list θ, then derive the resolvent sentence:

  $\text{subst}(\theta, P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \ldots P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m)$

- Example
  - from clause **P(x, f(a)) ∨ P(x, f(y)) ∨ Q(y)**
  - and clause **¬P(z, f(a)) ∨ ¬Q(z)**
  - derive resolvent **P(z, f(y)) ∨ Q(y) ∨ ¬Q(z)**
  - using **θ = {x/z}**

Resolution Refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- **Proof by contradiction:** Add ¬Q to KB and try to prove false.

  i.e., (KB |- Q) ↔ (KB ∧ ¬Q |- False)
- Resolution is **refutation complete**: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is **not entailed** by KB.
- Resolution **won't always give an answer** since entailment is only semidecidable
  - And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Refutation Resolution Proof Tree

¬ allergies(w) v sneeze(w)   ¬ cat(y) v ¬allergic-to-cats(z) v allergies(z)

w/z

¬ cat(y) v sneeze(z) v ¬allergic-to-cats(z)   cat(Felix)

y/Felix

sneeze(z) v ¬allergic-to-cats(z)   allergic-to-cats(Lise)

z/Lise

sneeze(Lise)   ¬ sneeze(Lise)

false

negated query

Questions to Answer

- How to convert FOL sentences to conjunctive normal form (a.k.a. CNF, clause form): **normalization** and **skolemization**
- How to unify two argument lists, i.e., how to find their most general unifier (mgu) q: **unification**
- How to determine which two clauses in KB should be resolved next (among all resolvable pairs of clauses) : **resolution (search) strategy**

Converting to CNF

Converting Sentences to CNF

1. Eliminate all $\leftrightarrow$ connectives

   $(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

2. Eliminate all $\rightarrow$ connectives

   $(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

   $\neg\neg P \Rightarrow P$

   $\neg(P \vee Q) \Rightarrow \neg P \wedge \neg Q$

   $\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$

   $\neg(\forall x)P \Rightarrow (\exists x)\neg P$

   $\neg(\exists x)P \Rightarrow (\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Converting Sentences to Clausal Form

Skolem Constants and Functions

5. Eliminate existential quantification by introducing Skolem constants/functions

   $(\exists x)P(x) \Rightarrow P(C)$

   C is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

   $(\forall x)(\exists y)P(x,y) \Rightarrow (\forall x)P(x, f(x))$

   since $\exists$ is within the scope of a universally quantified variable, use a **Skolem function f** to construct a new value that **depends on** the universally quantified variable

   f must be a brand-new function name not occurring in any other sentence in the KB.

   E.g., $(\forall x)(\exists y)loves(x,y) \Rightarrow (\forall x)loves(x,f(x))$

   In this case, f(x) specifies the person that x loves

Converting Sentences to Clausal Form

6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the "prefix" part

   Ex: $(\forall x)P(x) \Rightarrow P(x)$

7. Put into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws

   $(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$

   $(P \vee Q) \vee R \Rightarrow (P \vee Q \vee R)$

8. Split conjuncts into separate clauses

9. Standardize variables so each clause contains only variable names that do not occur in any other clause

An Example

$$(\forall x)(P(x) \rightarrow ((\forall y)(P(y) \rightarrow P(f(x,y))) \land \neg(\forall y)(Q(x,y) \rightarrow P(y))))$$

2. Eliminate $\rightarrow$

   $(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land \neg(\forall y)(\neg Q(x,y) \lor P(y))))$

3. Reduce scope of negation

   $(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (\exists y)(Q(x,y) \land \neg P(y))))$

4. Standardize variables

   $(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (\exists z)(Q(x,z) \land \neg P(z))))$

5. Eliminate existential quantification

   $(\forall x)(\neg P(x) \lor ((\forall y)(\neg P(y) \lor P(f(x,y))) \land (Q(x,g(x)) \land \neg P(g(x)))))$

6. Drop universal quantification symbols

   $(\neg P(x) \lor ((\neg P(y) \lor P(f(x,y))) \land (Q(x,g(x)) \land \neg P(g(x)))))$

Example

7. Convert to conjunction of disjunctions

   $(\neg P(x) \lor \neg P(y) \lor P(f(x,y))) \land (\neg P(x) \lor Q(x,g(x))) \land$

   $(\neg P(x) \lor \neg P(g(x)))$

8. Create separate clauses

   $\neg P(x) \lor \neg P(y) \lor P(f(x,y))$

   $\neg P(x) \lor Q(x,g(x))$

   $\neg P(x) \lor \neg P(g(x))$

9. Standardize variables

   $\neg P(x) \lor \neg P(y) \lor P(f(x,y))$

   $\neg P(z) \lor Q(z,g(z))$

   $\neg P(w) \lor \neg P(g(w))$

Unification

Unification

- Unification is a **"pattern-matching"** procedure
  - Takes two atomic sentences, called literals, as input
  - Returns "Failure" if they do not match and a substitution list, θ, if they do
- That is, $unify(p,q) = \theta$ means $subst(\theta, p) = subst(\theta, q)$ for two atomic sentences, *p* and *q*
- **θ** is called the **most general unifier** (mgu)
- All variables in the given two literals are implicitly universally quantified
- To make literals match, replace (universally quantified) variables by terms

Unification Algorithm

```
procedure unify(p, q, θ)
    Scan p and q left-to-right and find the first corresponding
      terms where p and q "disagree" (i.e., p and q not equal)
    If there is no disagreement, return θ  (success!)
    Let r and s be the terms in p and q, respectively,
      where disagreement first occurs
    If variable(r) then {
      Let θ = union(θ, {r/s})
      Return unify(subst(θ, p), subst(θ, q), θ)
    } else if variable(s) then {
      Let θ = union(θ, {s/r})
      Return unify(subst(θ, p), subst(θ, q), θ)
    } else return "Failure"
  end
```

Unification: Remarks

- *Unify* is a linear-time algorithm that returns the most general unifier (mgu), i.e., the shortest-length substitution list that makes the two literals match.
- In general, there is not a **unique** minimum-length substitution list, but unify returns one of minimum length
- A variable can never be replaced by a term containing that variable

  Example: x/f(x) is illegal.
- This "occurs check" should be done in the above pseudo-code before making the recursive calls

Unification Examples

- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(Bill), y)
  - {x/Bill, y/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Bill))
  - parents(Bill, father(y), z)
  - {x/Bill, y/Bill, z/mother(Bill)}
- Example:
  - parents(x, father(x), mother(Jane))
  - parents(Bill, father(y), mother(y))
  - Failure

Resolution Example

Practice Cxample

Did Curiosity Kill the Cat?

- Jack owns a dog. Every dog owner is an animal lover. No animal lover kills an animal. Either Jack or Curiosity killed the cat, who is named Tuna. Did Curiosity kill the cat?
- These can be represented as follows:

A. $(\exists x)\ Dog(x) \wedge Owns(Jack,x)$

B. $(\forall x)\ ((\exists y)\ Dog(y) \wedge Owns(x, y)) \rightarrow AnimalLover(x)$

C. $(\forall x)\ AnimalLover(x) \rightarrow ((\forall y)\ Animal(y) \rightarrow \neg Kills(x,y))$

D. $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

E. $Cat(Tuna)$

F. $(\forall x)\ Cat(x) \rightarrow Animal(x)$

G. $Kills(Curiosity, Tuna)$ $\longleftarrow$ GOAL

- **Convert to clause form**

  A1. (Dog(D)) ← D is a skolem constant

  A2. (Owns(Jack,D))

  B. (¬Dog(y), ¬Owns(x, y), AnimalLover(x))

  C. (¬AnimalLover(a), ¬Animal(b), ¬Kills(a,b))

  D. (Kills(Jack,Tuna), Kills(Curiosity,Tuna))

  E. Cat(Tuna)

  F. (¬Cat(z), Animal(z))

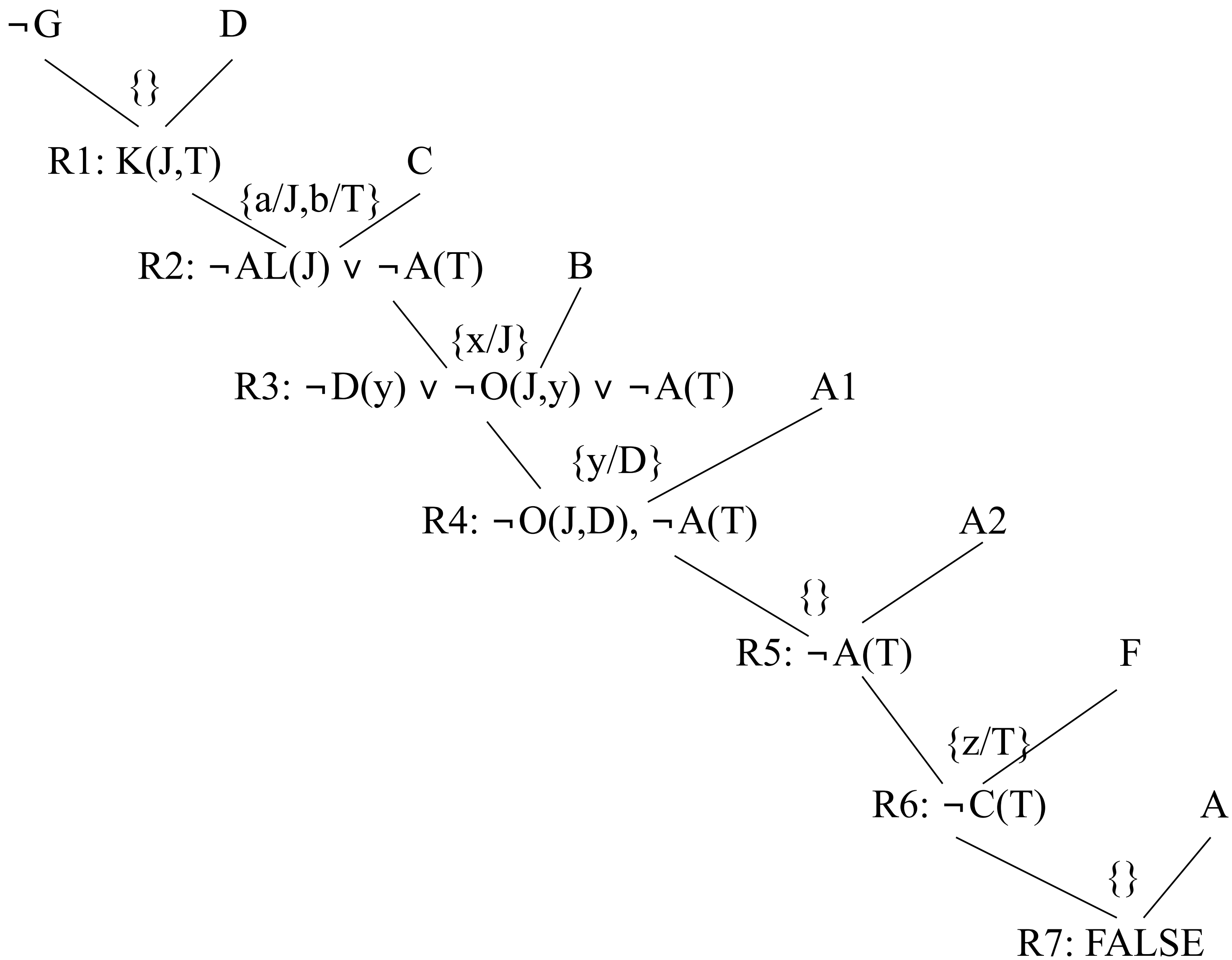
- **Add the negation of query:**

  ¬G: (¬Kills(Curiosity, Tuna))

- **The resolution refutation proof**

R1: ¬G, D, {}	(Kills(Jack, Tuna))
R2: R1, C, {a/Jack, b/Tuna}	(~AnimalLover(Jack), ~Animal(Tuna))
R3: R2, B, {x/Jack}	(~Dog(y), ~Owns(Jack, y), ~Animal(Tuna))
R4: R3, A1, {y/D}	(~Owns(Jack, D), ~Animal(Tuna))
R5: R4, A2, {}	(~Animal(Tuna))
R6: R5, F, {z/Tuna}	(~Cat(Tuna))
R7: R6, E, {}	FALSE

- **The proof tree**



Resolution Search Strategies

Resolution Theorem Proving as Search

- Resolution can be thought of as the **bottom-up construction of a search tree**, where the leaves are the clauses produced by KB and the negation of the goal
- When a pair of clauses generates a new resolvent clause, add a new node to the tree with arcs directed from the resolvent to the two parent clauses
- **Resolution succeeds** when a node containing the **False** clause is produced, becoming the **root node** of the tree
- A strategy is **complete** if its use guarantees that the empty clause (i.e., false) can be derived whenever it is entailed

Strategies

- There are a number of general (domain-independent) strategies that are useful in controlling a resolution theorem prover
- We'll briefly look at the following:
  - Breadth-first
  - Length heuristics
  - Set of support
  - Input resolution
  - Subsumption
  - Ordered resolution

Example

1. ¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire

negated goal

Breadth-First Search

- Level 0 clauses are the original axioms and the negation of the goal
- Level k clauses are the resolvents computed from two clauses, one of which must be from level k-1 and the other from any earlier level
- Compute all possible level 1 clauses, then all possible level 2 clauses, etc.
- Complete, but very inefficient

BFS Example

1. ¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire
10. (1,4) ¬Battery-OK ∨ ¬Bulbs-OK
11. (1,5) ¬Bulbs-OK ∨ Headlights-Work
12. (2,3) ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Flat-Tire ∨ Car-OK
13. (2,5) ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
14. (2,6) ¬Battery-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
15. (2,7) ¬Battery-OK ¬ Starter-OK ∨ Engine-Starts
16. … [and we're still only at Level 1!]

Length Heuristics

- **Shortest-clause heuristic**:
  Generate a clause with the fewest literals first
- **Unit resolution**:
  Prefer resolution steps in which at least one parent clause is a “unit clause,” i.e., a clause containing a single literal
  - Not complete in general, but complete for Horn clause KBs

Unit Resolution Example

1. ¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire
10. ¬Bulbs-OK ∨ Headlights-Work — 1,5
11. ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts — 2,5
12. ¬Battery-OK ∨ Empty-Gas-Tank ∨ Engine-Starts — 2,6
13. ¬Battery-OK ¬ Starter-OK ∨ Engine-Starts — 2,7
14. ¬Engine-Starts ∨ Flat-Tire — 3,8
15. ¬Engine-Starts ¬ Car-OK — 3,9
16. … [this doesn’t seem to be headed anywhere either!]

Set of Support

- At least one parent clause must be the negation of the goal *or* a "descendant" of such a goal clause (i.e., derived from a goal clause)
- *(When there's a choice, take the most recent descendant)*
- Complete (assuming all possible set-of-support clauses are derived)
- Gives a goal-directed character to the search

Set of Support Example

1. ¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire

9,3 — 10. ¬Engine-Starts ∨ Car-OK
10,2 — 11. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Car-OK
10,8 — 12. ¬Engine-Starts
11,5 — 13. ¬Starter-OK ∨ Empty-Gas-Tank ∨ Car-OK
11,6 — 14. ¬Battery-OK ∨ Empty-Gas-Tank ∨ Car-OK
11,7 — 15. ¬Battery-OK ∨ ¬Starter-OK ∨ Car-OK
16. … [a bit more focused, but we still seem to be wandering]

Unit Resolution + Set of Support Example

1. ¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire

9,3 10. ¬Engine-Starts ∨ Car-OK

10,8 11. ¬Engine-Starts

12,2 12. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank

12,5 13. ¬Starter-OK ∨ Empty-Gas-Tank

13,6 14. Empty-Gas-Tank

14,7 15. FALSE

[Hooray! Now that’s more like it!]

Simplification Heuristics

- **Subsumption:**
  Eliminate all sentences that are subsumed by (more specific than) an existing sentence to keep the KB small
  - If P(x) is already in the KB, adding P(A) makes no sense – P(x) is a superset of P(A)
  - Likewise adding P(A) ∨ Q(B) would add nothing to the KB
- **Tautology:**
  Remove any clause containing two complementary literals (tautology)
- **Pure symbol:**
  If a symbol always appears with the same "sign," remove all the clauses that contain it
  - Equivalent to assuming that symbol to be always-true or always-false (∴ can't draw any inferences about other symbols in the clause)

Example (Pure Symbol)

1. ~~¬Battery-OK ∨ ¬Bulbs-OK ∨ Headlights-Work~~
2. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
3. ¬Engine-Starts ∨ Flat-Tire ∨ Car-OK
4. ~~Headlights-Work~~
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. ¬Flat-Tire

Input Resolution

- At least one parent must be one of the input sentences (i.e., either a sentence in the original KB or the negation of the goal)
- Not complete in general, but complete for Horn clause KBs
- **Linear resolution**
  - Extension of input resolution
  - One of the parent sentences must be an input sentence *or* an ancestor of the other sentence
  - Complete

Ordered Resolution

- Search for resolvable sentences in order (left to right)
- This is how Prolog operates
- Resolve the first element in the sentence first
- This forces the user to define what is important in generating the "code"
- The way the sentences are written controls the resolution

Prolog

- A logic programming language based on Horn clauses
  - Resolution refutation
  - Control strategy: goal-directed and depth-first
    - always start from the goal clause
    - always use the new resolvent as one of the parent clauses for resolution
    - backtracking when the current thread fails
    - complete for Horn clause KB
  - Support answer extraction (can request single or all answers)
  - Orders the clauses and literals within a clause to resolve non-determinism
    - Q(a) may match both Q(x) <= P(x) and Q(y) <= R(y)
    - A (sub)goal clause may contain more than one literals, i.e., <= P1(a), P2(a)
  - Use "closed world" assumption (negation as failure)
    - If it fails to derive P(a), then assume ~P(a)

Summary

- Logical agents apply inference to a knowledge base to derive new information and make decisions
- Basic concepts of logic:
  - Syntax: formal structure of sentences
  - Semantics: truth of sentences wrt models
  - Entailment: necessary truth of one sentence given another
  - Inference: deriving sentences from other sentences
  - Soundness: derivations produce only entailed sentences
  - Completeness: derivations can produce all entailed sentences
- FC and BC are linear time, complete for Horn clauses
- Resolution is a sound and complete inference method for propositional and first-order logic