Logic Inference

Chapter 7.5, 9

Some material adopted from notes and slides by Tim Finin, Marie desJardins, Andreas Geyer-Schulz and Chuck Dyer
Outline

• Model checking
• Inference in first-order logic
  – Inference rules
  – Forward chaining
  – Backward chaining
  – Resolution
    • Clausal form
    • Unification
    • Resolution as search
Model checking

• Given KB, does sentence S hold?
• Basically generate and test:
  – Generate all the possible models
  – Consider the models M in which KB is TRUE
  – If $\forall M S$, then S is **provably true**
  – If $\forall M \neg S$, then S is **provably false**
  – Otherwise ($\exists M_1 S \land \exists M_2 \neg S$): S is **satisfiable** but neither provably true or provably false
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 elimination
  - Existential introduction
  - Existential elimination
  - Generalized Modus Ponens (GMP)
Automating FOL inference with Generalized Modus Ponens
Automated inference for FOL

- Automated inference using FOL is harder than PL
  - Variables can potentially take on an infinite number of possible values from their domains
  - Hence there are potentially an infinite number of ways to apply the Universal Elimination rule of inference

- Gödel's Completeness Theorem says that FOL entailment is only semidecidable
  - If a sentence is true given a set of axioms, there is a procedure that will determine this
  - If the sentence is false, then there is no guarantee that a procedure will ever determine this—i.e., it may never halt
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 $R(c)$
• General case: **Given**
  – atomic sentences $P_1$, $P_2$, ..., $P_N$
  – implication sentence $(Q_1 \land Q_2 \land ... \land Q_N) \rightarrow R$
    • $Q_1$, ..., $Q_N$ and $R$ are atomic sentences
  – substitution $\text{subst}(\theta, P_i) = \text{subst}(\theta, Q_i)$ for $i=1,...,N$
  – Derive new sentence: $\text{subst}(\theta, R)$
• Substitutions
  – $\text{subst}(\theta, \alpha)$ denotes the result of applying a set of substitutions defined by $\theta$
    to the sentence $\alpha$
  – 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
  – $\text{subst}(\{x/\text{IceCream}, y/\text{Ziggy}\}, \text{eats}(y,x)) = \text{eats}(\text{Ziggy}, \text{IceCream})$
Horn clauses

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

where
  – there are 0 or more \(P_i\)s and 0 or 1 \(Q\)
  – the \(P_i\)s and \(Q\) are positive (i.e., non-negated) literals

• Equivalently: \(P_1(x) \lor P_2(x) \ldots \lor P_n(x)\) 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
Horn clauses II

• Special cases
  – $P_1 \land P_2 \land \ldots \land P_n \rightarrow Q$
  – $P_1 \land P_2 \land \ldots \land P_n \rightarrow \text{false}$
  – $\text{true} \rightarrow Q$

• These are not Horn clauses:
  – $p(a) \lor q(a)$
  – $(P \land Q) \rightarrow (R \lor S)$
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:
  – allergies(X) → sneeze(X)
  – cat(Y) ∧ allergic-to-cats(X) → allergies(X)
  – cat(Felix)
  – allergic-to-cats(Lise)

• Goal:
  – sneeze(Lise)
Forward chaining algorithm

procedure FORWARD-CHAIN(KB, p)
    if there is a sentence in KB that is a renaming of p then return
    Add p to KB
    for each \((p_1 \land \ldots \land p_n \Rightarrow q)\) in KB such that for some \(i\), UNIFY\((p_i, p) = \theta\) succeeds do
        FIND-AND-INFER(KB, \([p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n]\), q, \(\theta\))
    end

procedure FIND-AND-INFER(KB, premises, conclusion, \(\theta\))
    if premises = [] then
        FORWARD-CHAIN(KB, SUBST(\(\theta\), conclusion))
    else for each \(p'\) in KB such that UNIFY\((p', \text{SUBST}(\theta, \text{FIRST}(\text{premises}))) = \theta_2\) do
        FIND-AND-INFER(KB, \text{REST}(\text{premises}), conclusion, \text{COMPOSE}(\theta, \theta_2))
    end
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)
Backward chaining algorithm

function BACK-CHAIN(KB, q) returns a set of substitutions

\[ \text{BACK-CHAIN-LIST}(KB, [q], \{\}) \]

function BACK-CHAIN-LIST(KB, qlist, \( \theta \)) returns a set of substitutions

inputs: KB, a knowledge base
\[ \text{qlist}, \text{a list of conjuncts forming a query (\( \theta \) already applied)} \]
\[ \theta, \text{the current substitution} \]

static: answers, a set of substitutions, initially empty

if qlist is empty then return \{\( \theta \)\}

\[ q \leftarrow \text{FIRST}(qlist) \]

for each \( q_i' \) in KB such that \( \theta_i \leftarrow \text{UNIFY}(q, q_i') \) succeeds do

Add \( \text{COMPOSE}(\theta, \theta_i) \) to answers

end

for each sentence \( (p_1 \land \ldots \land p_n \Rightarrow q_i') \) in KB such that \( \theta_i \leftarrow \text{UNIFY}(q, q_i') \) succeeds do

\[ \text{answers} \leftarrow \text{BACK-CHAIN-LIST}(KB, \text{SUBST}(\theta_i, [p_1 \ldots p_n]), \text{COMPOSE}(\theta, \theta_i)) \cup \text{answers} \]

end

return the union of \( \text{BACK-CHAIN-LIST}(KB, \text{REST}(qlist), \theta) \) for each \( \theta \in \text{answers} \)
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
Completeness of GMP

- GMP (using forward or backward chaining) is complete for KBs that contain only Horn clauses.
- It is **not complete** for simple KBs that contain non-Horn clauses.
- The following entail that S(A) is true:
  \[
  (\forall x) P(x) \rightarrow Q(x) \\
  (\forall x) \neg P(x) \rightarrow R(x) \\
  (\forall x) Q(x) \rightarrow S(x) \\
  (\forall x) R(x) \rightarrow S(x)
  \]
- If we want to conclude S(A), with GMP we cannot, since the second one is not a Horn clause.
- It is equivalent to \( P(x) \lor R(x) \)
Automating FOL inference with resolution
Resolution

• Resolution is a **sound** and **complete** inference procedure for FOL

• Reminder: Resolution rule for propositional logic:
  
  \[ P_1 \lor P_2 \lor \ldots \lor P_n \]
  \[ \neg P_1 \lor Q_2 \lor \ldots \lor Q_m \]

  \[ \text{Resolvent: } P_2 \lor \ldots \lor P_n \lor Q_2 \lor \ldots \lor Q_m \]

• Examples
  
  – P and \( \neg P \lor Q \): derive Q (Modus Ponens)
  
  – \((\neg P \lor Q)\) and \((\neg Q \lor R)\): derive \(\neg P \lor R\)
  
  – P and \(\neg P\): derive False [contradiction!]
  
  – \((P \lor Q)\) and \((\neg P \lor \neg Q)\): derive True
Resolution in first-order logic

• Given sentences
  \[ P_1 \lor \ldots \lor P_n \]
  \[ Q_1 \lor \ldots \lor Q_m \]

• 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 \( \theta \), then derive the resolvent sentence:
  \[ \text{subst}(\theta, P_1 \lor \ldots \lor P_{j-1} \lor P_{j+1} \ldots P_n \lor Q_1 \lor \ldots Q_{k-1} \lor Q_{k+1} \lor \ldots \lor Q_m) \]

• Example
  – from clause \( P(x, f(a)) \lor P(x, f(y)) \lor Q(y) \)
  – and clause \( \neg P(z, f(a)) \lor \neg Q(z) \)
  – derive resolvent \( P(z, f(y)) \lor Q(y) \lor \neg Q(z) \)
  – using \( \theta = \{ x/z \} \)
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

\[ \neg \text{allergies}(w) \lor \text{sneeze}(w) \quad \neg \text{cat}(y) \lor \neg \text{allergic-to-cats}(z) \lor \text{allergies}(z) \]

\[ \neg \text{cat}(y) \lor \text{sneeze}(z) \lor \neg \text{allergic-to-cats}(z) \quad \text{cat}(\text{Felix}) \]

\[ \text{sneeze}(z) \lor \neg \text{allergic-to-cats}(z) \quad \text{allergic-to-cats}(\text{Lise}) \]

\[ \text{sneeze}(\text{Lise}) \quad \neg \text{sneeze}(\text{Lise}) \]

\[ \text{false} \]

\[ \text{negated query} \]
We need answers to the following questions

• 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) $\theta$: 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
   
   $(P \rightarrow Q) \Rightarrow (\neg P \lor Q)$

3. Reduce the scope of each negation symbol to a single predicate
   
   $\neg \neg P \Rightarrow P$
   
   $\neg (P \lor Q) \Rightarrow \neg P \land \neg Q$
   
   $\neg (P \land Q) \Rightarrow \neg P \lor \neg Q$
   
   $\neg (\forall x)P \Rightarrow (\exists x)\neg P$
   
   $\neg (\exists x)P \Rightarrow (\forall x)\neg P$

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., \((\forall x)(\exists y)\text{loves}(x,y) \Rightarrow (\forall x)\text{loves}(x, f(x))\)

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
   $(P \land Q) \lor R \Rightarrow (P \lor R) \land (Q \lor R)$
   $(P \lor Q) \lor R \Rightarrow (P \lor Q \lor R)$

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

\[(\forall x)(P(x) \to ((\forall y)(P(y) \to P(f(x,y)))) \land \neg(\forall y)(Q(x,y) \to P(y))))\]

2. Eliminate \(\to\)

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

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, $\theta$, if they do
• That is, $\text{unify}(p,q) = \theta$ means $\text{subst}(\theta, p) = \text{subst}(\theta, q)$ for two atomic sentences, $p$ and $q$
• $\theta$ 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 example

*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. \((∃x) \text{Dog}(x) \land \text{Owns}(Jack,x)\)
  
  B. \((∀x) ((∃y) \text{Dog}(y) \land \text{Owns}(x, y)) \rightarrow \text{AnimalLover}(x)\)
  
  C. \((∀x) \text{AnimalLover}(x) \rightarrow ((∀y) \text{Animal}(y) \rightarrow \neg\text{Kills}(x,y))\)
  
  D. \(\text{Kills}(Jack,\text{Tuna}) \lor \text{Kills}(\text{Curiosity},\text{Tuna})\)
  
  E. \(\text{Cat}(\text{Tuna})\)
  
  F. \((∀x) \text{Cat}(x) \rightarrow \text{Animal}(x)\)
  
  G. \(\text{Kills}(\text{Curiosity}, \text{Tuna}) \quad \text{GOAL}\)
• Convert to clause form
  A1. (Dog(D)) \hspace{5cm} \text{D is a skolem constant}
  A2. (Owns(Jack,D))
  B. (\neg Dog(y), \neg Owns(x, y), \text{AnimalLover}(x))
  C. (\neg \text{AnimalLover}(a), \neg \text{Animal}(b), \neg \text{Kills}(a,b))
  D. (\text{Kills}(Jack,Tuna), \text{Kills}(\text{Curiosity},Tuna))
  E. \text{Cat}(Tuna)
  F. (\neg \text{Cat}(z), \text{Animal}(z))

• Add the negation of query:
  \neg G: (\neg \text{Kills}(\text{Curiosity}, \text{Tuna}))
• The resolution refutation proof

R1: \neg G, D, {} \quad (\text{Kills(Jack, Tuna)})

R2: R1, C, \{a/Jack, b/Tuna\} \quad (\sim\text{AnimalLover(Jack)}, \sim\text{Animal(Tuna)})

R3: R2, B, \{x/Jack\} \quad (\sim\text{Dog(y)}, \sim\text{Owns(Jack, y)}, \sim\text{Animal(Tuna)})

R4: R3, A1, \{y/D\} \quad (\sim\text{Owns(Jack, D)}, \sim\text{Animal(Tuna)})

R5: R4, A2, {} \quad (\sim\text{Animal(Tuna)})

R6: R5, F, \{z/Tuna\} \quad (\sim\text{Cat(Tuna)})

R7: R6, E, {} \quad \text{FALSE}
The proof tree

\[ \neg G \quad \neg \neg D \]
\[ \{ \} \]
\[ R1: K(J,T) \quad C \]
\[ \{a/J,b/T\} \]
\[ R2: \neg AL(J) \lor \neg A(T) \quad B \]
\[ \{x/J\} \]
\[ R3: \neg D(y) \lor \neg O(J,y) \lor \neg A(T) \quad A1 \]
\[ \{y/D\} \]
\[ R4: \neg O(J,D), \neg A(T) \quad A2 \]
\[ \{\} \]
\[ R5: \neg A(T) \quad F \]
\[ \{z/T\} \]
\[ R6: \neg C(T) \quad A \]
\[ \{\} \]
\[ R7: \text{FALSE} \]
Resolution search strategies
Resolution TP 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. \neg\text{Battery-OK} \lor \neg\text{Bulbs-OK} \lor \text{Headlights-Work}
2. \neg\text{Battery-OK} \lor \neg\text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts}
3. \neg\text{Engine-Starts} \lor \text{Flat-Tire} \lor \text{Car-OK}
4. \text{Headlights-Work}
5. \text{Battery-OK}
6. \text{Starter-OK}
7. \neg\text{Empty-Gas-Tank}
8. \neg\text{Car-OK}
9. \neg\text{Flat-Tire}

\textit{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. \(\neg\text{Battery-OK} \lor \neg\text{Bulbs-OK} \lor \text{Headlights-Work}\)
2. \(\neg\text{Battery-OK} \lor \neg\text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts}\)
3. \(\neg\text{Engine-Starts} \lor \text{Flat-Tire} \lor \text{Car-OK}\)
4. \(\text{Headlights-Work}\)
5. \(\text{Battery-OK}\)
6. \(\text{Starter-OK}\)
7. \(\neg\text{Empty-Gas-Tank}\)
8. \(\neg\text{Car-OK}\)
9. \(\neg\text{Flat-Tire}\)
10. \(\neg\text{Battery-OK} \lor \neg\text{Bulbs-OK}\)
11. \(\neg\text{Bulbs-OK} \lor \text{Headlights-Work}\)
12. \(\neg\text{Battery-OK} \lor \neg\text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Flat-Tire} \lor \text{Car-OK}\)
13. \(\neg\text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts}\)
14. \(\neg\text{Battery-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts}\)
15. \(\neg\text{Battery-OK} \lor \neg\text{Starter-OK} \lor \text{Engine-Starts}\)
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. \( \neg \text{Battery-OK} \lor \neg \text{Bulbs-OK} \lor \text{Headlights-Work} \)
2. \( \neg \text{Battery-OK} \lor \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts} \)
3. \( \neg \text{Engine-Starts} \lor \text{Flat-Tire} \lor \text{Car-OK} \)
4. \text{Headlights-Work} 
5. \text{Battery-OK} 
6. \text{Starter-OK} 
7. \( \neg \text{Empty-Gas-Tank} \)
8. \( \neg \text{Car-OK} \)
9. \( \neg \text{Flat-Tire} \)
10. \( \neg \text{Bulbs-OK} \lor \text{Headlights-Work} \)
11. \( \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts} \)
12. \( \neg \text{Battery-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts} \)
13. \( \neg \text{Battery-OK} \land \neg \text{Starter-OK} \lor \text{Engine-Starts} \)
14. \( \neg \text{Engine-Starts} \lor \text{Flat-Tire} \)
15. \( \neg \text{Engine-Starts} \land \neg \text{Car-OK} \)
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. $\neg$Battery-OK $\lor$ $\neg$Bulbs-OK $\lor$ Headlights-Work
2. $\neg$Battery-OK $\lor$ $\neg$Starter-OK $\lor$ Empty-Gas-Tank $\lor$ Engine-Starts
3. $\neg$Engine-Starts $\lor$ Flat-Tire $\lor$ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. $\neg$Empty-Gas-Tank
8. $\neg$Car-OK
9. $\neg$Flat-Tire
10. $\neg$Engine-Starts $\lor$ Car-OK
11. $\neg$Battery-OK $\lor$ $\neg$Starter-OK $\lor$ Empty-Gas-Tank $\lor$ Car-OK
12. $\neg$Engine-Starts
13. $\neg$Starter-OK $\lor$ Empty-Gas-Tank $\lor$ Car-OK
14. $\neg$Battery-OK $\lor$ Empty-Gas-Tank $\lor$ Car-OK
15. $\neg$Battery-OK $\lor$ $\neg$Starter-OK $\lor$ Car-OK
16. ... [a bit more focused, but we still seem to be wandering]
Unit resolution + set of support example

1. \( \neg \text{Battery-OK} \lor \neg \text{Bulbs-OK} \lor \text{Headlights-Work} \)
2. \( \neg \text{Battery-OK} \lor \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts} \)
3. \( \neg \text{Engine-Starts} \lor \text{Flat-Tire} \lor \text{Car-OK} \)
4. \( \text{Headlights-Work} \)
5. \( \text{Battery-OK} \)
6. \( \text{Starter-OK} \)
7. \( \neg \text{Empty-Gas-Tank} \)
8. \( \neg \text{Car-OK} \)
9. \( \neg \text{Flat-Tire} \)
10. \( \neg \text{Engine-Starts} \lor \text{Car-OK} \)
11. \( \neg \text{Engine-Starts} \)
12. \( \neg \text{Battery-OK} \lor \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \)
13. \( \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \)
14. \( \text{Empty-Gas-Tank} \)
15. \( \text{FALSE} \)

[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
Example (Pure Symbol)

1. $\neg$Battery-OK $\lor$ Bulbs-OK $\lor$ Headlights-Work
2. $\neg$Battery-OK $\lor$ $\neg$Starter-OK $\lor$ Empty-Gas-Tank $\lor$ Engine-Starts
3. $\neg$Engine-Starts $\lor$ Flat-Tire $\lor$ Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. $\neg$Empty-Gas-Tank
8. $\neg$Car-OK
9. $\neg$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) \(\leq\) P(x) and Q(y) \(\leq\) R(y)
    • A (sub)goal clause may contain more than one literals, i.e., \(\leq\) P1(a), P2(a)
  – Use “closed world” assumption (negation as failure)
    • If it fails to derive P(a), then assume \(\sim\)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