Overview

- Model checking for PL
- Inference in first-order logic
  - Inference rules and generalized modes ponens
  - Forward chaining
  - Backward chaining
  - Resolution
    - Clausal form
    - Unification
    - Resolution as search

PL 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 ($\exists M1 S \land \exists M2 \neg S$): S is **satisfiable** but neither provably true or provably false

Efficient PL model checking

- Davis-Putnam algorithm (DPLL) is a generate-and-test model checking with:
  - *Early termination*: short-circuiting of disjunction and conjunction
  - *Pure symbol heuristic*: Any symbol that only appears negated or unnegated must be FALSE/TRUE respectively
    e.g., in $([A \lor \neg B], \neg B \lor \neg C, C \lor A)$, A & B are pure. Make pure symbol literal true; if there’s a model for S, then making a pure symbol true is also a model
  - *Unit clause heuristic*: Any symbol that appears in a clause by itself can immediately be set to TRUE or FALSE
- **WALKSAT**: Local search for satisfiability: Pick a symbol to flip (toggle TRUE/FALSE), either using min-conflicts or choosing randomly
- …or you can use *any* local or global search algorithm!
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)

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
• Godel'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

Automating FOL inference with Generalized Modus Ponens

• Modus Ponens
  – \( P, \ P \rightarrow Q \vdash Q \)
• Generalized Modus Ponens (GMP) extends this to rules in FOL
  – Combines And-Introduction, Universal-Elimination, and Modus Ponens, e.g.
    – from \( P(c) \text{ and } Q(c) \text{ and } \forall x \ P(x) \land Q(x) \rightarrow R(x) \) derive \( R(c) \)
• Need to deal with
  – more than one condition on left side of rule
  – variables

Generalized Modus Ponens

• Modus Ponens
  – \( P, \ P \rightarrow Q \vdash Q \)
• Generalized Modus Ponens (GMP) extends this to rules in FOL
  – Combines And-Introduction, Universal-Elimination, and Modus Ponens, e.g.
    – from \( P(c) \text{ and } Q(c) \text{ and } \forall x \ P(x) \land Q(x) \rightarrow R(x) \) derive \( R(c) \)
• Need to deal with
  – more than one condition on left side of rule
  – variables
**Generalized Modus Ponens**

- **General case:** Given
  - atomic sentences $P_1, P_2, ..., P_N$
  - implication sentence $(Q_1 \land Q_2 \land ... \land Q_N) \rightarrow R$
    - $Q_i, ..., Q_N$ 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 made in left-to-right order in the list
  - $\text{subst}(\{x/\text{Cheese}, y/\text{Mickey}\}, \text{eats}(y,x)) = \text{eats}(\text{Mickey}, \text{Cheese})$

**Our rules are Horn clauses**

- A Horn clause is a sentence of the form:
  $$P_1(x) \land P_2(x) \land ... \land P_n(x) \rightarrow Q(x)$$
  - $\geq 0$ $P_i$s and 0 or 1 $Q$
  - the $P_i$s and $Q$ are positive (i.e., non-negated) literals
- Equivalently: $P_1(x) \lor P_2(x) \lor ... \lor P_n(x)$ where the $P_i$ are all atomic and at most one 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**
  - Typical rule: $P_1 \land P_2 \land ... \land P_n \rightarrow Q$
  - Constraint: $P_1 \land P_2 \land ... \land P_n \rightarrow \text{false}$
  - A fact: $\text{true} \rightarrow Q$
- These are not Horn clauses:
  - $p(a) \lor q(a)$
  - $(P \land Q) \rightarrow (R \lor S)$
- Note: can’t assert or conclude disjunctions, no negation
- No wonder reasoning over Horn clauses is easier

**Horn clauses III**

- Where are the quantifiers?
  - Variables appearing in conclusion are universally quantified
  - Variables appearing only in premises are existentially quantified
- Example: grandparent relation
  - $\forall P_1.P_2 \exists P X \text{parent}(P_1, X) \land \text{parent}(X, P_2) \rightarrow \text{grandParent}(P_1, P_2)$
  - $\forall P_1.P_2 \exists P X \text{parent}(P_1, X) \land \text{parent}(X, P_2) \rightarrow \text{grandParent}(P_1, P_2)$
  - Prolog: $\text{grandParent}(P_1,P_2) :- \text{parent}(P_1,X), \text{parent}(X,P_2)$
Forward & Backward Reasoning

• We usually talk about two reasoning strategies:
  Forward and backward ‘chaining’
• Both are equally powerful
• You can also have a mixed strategy

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 sound and complete for KBs containing only Horn clauses

Forward chaining algorithm

procedure FORWARD-CHAIN(KB, p )
if there is a sentence in KB that is a reasoning 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 \( i, \text{UBCH}(p, p_i) = \theta \) succeeds do
  FIND-AND-INSERT(KB, \( \{p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n\}, \gamma, \theta \) )
end

procedure FIND-AND-INSERT(KB, premises, conclusion, \( \theta \) )
if premises = \& then
  FORWARD-CHAIN(KB, \( \text{SUBST}(\theta, \text{conclusion}) \) )
else for each \( p' \) in KB such that \( \text{Unify}(p', \text{SUBST}(\theta, \text{First(premises)})) = \theta \) do
  FIND-AND-INSERT(KB, \( \text{REST(premises)} \), conclusion, \( \text{COMPOSE}(\theta, \theta') \) )
end

Forward chaining example

• KB:
  – allergies(X) → sneeze(X)
  – cat(Y) \land allergicToCats(X) → allergies(X)
  – cat(felix)
  – allergicToCats(mary)
• Goal:
  – sneeze(mary)
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) ∧ allergicToCats(X) → allergies(X)
  - cat(felix)
  - allergicToCats(mary)
- Goal:
  - sneeze(mary)

Backward chaining algorithm

```
function Back-Chain(KB, q) returns a set of substitutions
  Back-Chain(KB - q, [])

function Back-Chain(KB, q, answer) returns a set of substitutions
  inputs: KB, a knowledge base
          q, a list of conjuncts forming a query (if already applied)
          answer, a set of substitutions, initially empty
  static: answers, a set of substitutions, initially empty
  if q is empty then return answer
  q ← FreshStart
  for each g in KB such that g ← Unify(q, g) succeeds do
    Add Computed(g, #) to answers
  end
  for each sentence (p₁ ∧ ... ∧ pₙ → q) in KB such that g ← Unify(q, g) succeeds do
    answers ← Back-Chain(KB - (p₁ ∧ ... ∧ pₙ), [], [], [], answers)
  end
  return the union of Back-Chain(KB - q, FreshStart()) for each # ∈ 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
  - Efficient when you want to compute all conclusions
- **BC** is goal-driven, better 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
  - Efficient when you want one or a few decisions
Mixed strategy

- Many practical reasoning systems do both forward and backward chaining
- The way you encode a rule determines how it is used, as in
  - % this is a forward chaining rule
    spouse(X,Y) => spouse(Y,X).
  - % this is a backward chaining rule
    wife(X,Y) <= spouse(X,Y), female(X).
- Given a model of the rules you have and the kind of reason you need to do, it’s possible to decide which to encode as FC and which as BC rules.

Completeness of GMP

- GMP (using forward or backward chaining) is complete for KBs that contain only Horn clauses
- **not complete** for simple KBs with non-Horn clauses
- The following entail that S(A) is true:
  1. (∀x) P(x) → Q(x)
  2. (∀x) ¬P(x) → R(x)
  3. (∀x) Q(x) → S(x)
  4. (∀x) R(x) → 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) ∨ R(x)

How about in Prolog?

- Let’s try encoding this in Prolog
  1. q(X) :- p(X).
  2. r(X) :- neg(p(X)).
  3. s(X) :- q(X).
  4. s(X) :- r(X).
- We should not use \(+ or not (in SWI) for negation since it means “negation as failure”
- Prolog explores possible proofs independently
- It can’t take a larger view and realize that one branch must be true since p(x) ∨ ¬p(x) is always true

Automating FOL Inference with Resolution
Resolution

• Resolution is a sound and complete inference procedure for unrestricted FOL
• Reminder: Resolution rule for propositional logic:
  – $P_1 \lor P_2 \lor \ldots \lor P_n$
  – $\neg P_1 \lor Q_2 \lor \ldots \lor Q_m$
  – Resolvent: $P_2 \lor \ldots \lor P_n \lor Q_2 \lor \ldots \lor Q_m$
• We’ll need to extend this to handle quantifiers and variables

Resolution in first-order logic

• Given sentences in conjunctive normal form:
  – $P_1 \lor \ldots \lor P_n$ and $Q_1 \lor \ldots \lor Q_m$
  – $P_i$ and $Q_i$ are literals, i.e., 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:
  subst($\theta$, $P_1 \lor \ldots \lor P_{j-1} \lor P_{j+1} \ldots \lor P_n \lor Q_1 \lor \ldots \lor Q_{k-1} \lor Q_{k+1} \ldots \lor Q_m$)
• Example
  – from clause $P(x, f(a)) \lor P(x, f(y)) \lor Q(y)$
  – and clause $\neg P(x, f(a)) \lor \neg Q(z)$
  – derive resolvent $P(z, f(y)) \lor Q(y) \lor \neg Q(z)$
  – Using $\theta = \{x/z\}$

Resolution covers many cases

• Modes Ponens
  – from $P$ and $P \rightarrow Q$ derive $Q$
  – from $P$ and $\neg P \lor Q$ derive $Q$
• Chaining
  – from $P \rightarrow Q$ and $Q \rightarrow R$ derive $P \rightarrow R$
  – from $(\neg P \lor Q)$ and $(\neg Q \lor R)$ derive $\neg P \lor R$
• Contradiction detection
  – from $P$ and $\neg P$ derive false
  – from $P$ and $\neg P$ derive the empty clause (=false)

A resolution proof tree
Resolution refutation

- Given a consistent set of axioms KB and goal sentence Q, show that KB \models Q
- **Proof by contradiction:** Add \neg Q to KB and try to prove false, i.e.:
  \((KB \models Q) \leftrightarrow (KB \land \neg Q \models \text{False})\)
- Resolution is **refutation complete:** it can establish that a given sentence Q is entailed by KB, but can’t (in general) 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 semi-decidable
  - And you can’t just run two proofs in parallel, one trying to prove Q and the other trying to prove \neg Q, since KB might not entail either one

Resolution example

- KB:
  - allergies(X) \rightarrow sneeze(X)
  - cat(Y) \land allergicToCats(X) \rightarrow allergies(X)
  - cat(felix)
  - allergicToCats(mary)
- Goal:
  - sneeze(mary)
questions to be answered

• 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
   \[(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \land (Q \rightarrow P))\]
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 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
   a better name might be oneWhoIsLovedBy(x)
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.

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

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, $\theta$, 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

```pseudo
procedure unify(p, q, $\theta$)
    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 $\theta$ (success!)
    Let r and s be the terms in p and q, respectively, where disagreement first occurs
    If variable(r) then {
        Let $\theta = \text{union}(\theta, \{r/s\})$
        Return unify(subst($\theta$, p), subst($\theta$, q), $\theta$)
    } else if variable(s) then {
        Let $\theta = \text{union}(\theta, \{s/r\})$
        Return unify(subst($\theta$, p), subst($\theta$, q), $\theta$)
    } else return "Failure"
end
```

## Unification: Remarks

- $\text{Unify}$ 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 isn’t a unique minimum-length substitution list, but unify returns one of minimum length
- Common constraint: 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)
  - $\{x/Bill, y/mother(Bill)\}$ yields parents(Bill, father(Bill), mother(Bill))
- Example:
  - parents(x, father(x), mother(y))
  - parents(Bill, father(y), z)
  - $\{x/Bill, y/Bill, z/mother(y)\}$ yields parents(Bill, father(Bill), mother(y))
- 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. $(\exists x) \ Dog(x) \land \ Owns(Jack, x)$
B. $(\forall x) \ ((\exists y) \ Dog(y) \land Owns(x, y)) \rightarrow \ AnimalLover(x)$
C. $(\forall x) \ AnimalLover(x) \rightarrow ((\forall y) \ Animal(y) \rightarrow \neg Kills(x,y))$
D. Kills(Jack,Tuna) \lor Kills(Curiosity,Tuna)
E. Cat(Tuna)
F. $(\forall x) \ Cat(x) \rightarrow Animal(x)$
G. Kills(Curiosity, Tuna)

GOAL

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?

- Convert to clause form
  A1. $(Dog(D))$
  A2. $(Owns(Jack,D))$
  B. $(\neg Dog(y), \neg Owns(x, y), AnimalLover(x))$
  C. $(\neg AnimalLover(a), \neg Animal(b), \neg Kills(a,b))$
  D. $(Kills(Jack,Tuna), Kills(Curiosity,Tuna))$
  E. $Cat(Tuna)$
  F. $(\neg Cat(z), Animal(z))$

- Add the negation of query:
  \neg G: \neg 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

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. Battery-OK ∧ Bulbs-OK → Headlights-Work
2. Battery-OK ∧ Starter-OK → Empty-Gas-Tank v Engine-Starts
3. Engine-Starts → Flat-Tire v Car-OK
4. Headlights-Work
5. Battery-OK
6. Starter-OK
7. ¬Empty-Gas-Tank
8. ¬Car-OK
9. Goal: Flat-Tire ?

Example

1. ¬Battery-OK v ¬Bulbs-OK v Headlights-Work
2. ¬Battery-OK v ¬Starter-OK v Empty-Gas-Tank v Engine-Starts
3. ¬Engine-Starts v Flat-Tire v 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. \( \neg \text{Battery-OK} \lor \neg \text{Bulbs-OK} \lor \text{Headlights-Work} \)
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. \( \neg \text{Battery-OK} \lor \text{Starter-OK} \lor \text{Engine-Starts} \)
16. \( \ldots \) [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. \( \neg \text{Battery-OK} \lor \neg \text{Bulbs-OK} \)
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. \( \neg \text{Battery-OK} \lor \text{Starter-OK} \lor \text{Engine-Starts} \)
16. \( \ldots \) [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 (e.g., like backward chaining)
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. \( \neg \text{Battery-OK} \lor \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Car-OK} \)
13. \( \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \)
14. \( \text{Empty-Gas-Tank} \)
15. \( \text{FALSE} \)
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. \( \neg \text{Battery-OK} \lor \neg \text{Starter-OK} \lor \text{Empty-Gas-Tank} \lor \text{Engine-Starts} \)
13. \( \neg \text{Engine-Starts} \lor \text{Empty-Gas-Tank} \)
14. \( \text{Empty-Gas-Tank} \)
15. \( \text{FALSE} \)
\([\text{Hooray! Now that's more like it!}]\)

Simplification heuristics

- **Subsumption:**
  Eliminate sentences that are subsumed by (more specific than) an existing sentence to keep 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 \( P(A) \lor Q(B) \) 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. \( \neg \text{Battery-OK} \lor \neg \text{Bulbs-OK} \lor \text{Headlights-Work} \)
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. \( \neg \text{Flat-Tire} \)
### 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: logic programming language based on Horn clauses
- Resolution refutation
- Control strategy: goal-directed and depth-first
  - always start from the goal clause
  - always use new resolvent as one of parent clauses for resolution
  - backtracking when the current thread fails
  - complete for Horn clause KB
- Supports answer extraction (can request single or all answers)
- Orders clauses & 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 >1 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 KB 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