CMSC 671
Fall 2010

Thu 10/7/10

FOL / Inference in FOL

Prof. Laura Zavala, laura.zavala@umbc.edu, ITE 373, 410-455-8775

Some material adopted from Hwee Tou Ng's course slides

Knowledge based agents and Logic (review/summary)

Declarative approach

- Knowledge-based agents follow a **declarative approach** to system building
  - In contrast a **procedural approach** encodes desired behaviors directly as program code.
  - Knowledge and Inference are separte
  - Inference is domain independent
  - pit[2,2] or pit[3,1]
  - If wumpus[1,1] then not wumpus[2,2]

Representation, reasoning, and logic

- **Logics** are formal languages for representing information such that conclusions can be drawn
- A knowledge representation language is defined by:
  - its **syntax,** which specifies all valid sentences in the language
  - its **semantics**, which defines the "meaning" or truth of each sentence

Propositional logic

- The simplest logic
- Syntax - the proposition symbols P_1, P_2 are sentences
  - $\neg S$ (negation), $S_1 \wedge S_2$ (conjunction), $S_1 \vee S_2$ (disjunction), $S_1 \Rightarrow S_2$ (implication), $S_1 \Leftrightarrow S_2$ (biconditional)
- Semantics

$\neg S$	is true iff	S is false	
$S_1 \wedge S_2$	is true iff	S_1 is true and	S_2 is true
$S_1 \vee S_2$	is true iff	S_1 is true or	S_2 is true
$S_1 \Rightarrow S_2$	is true iff	S_1 is false or	S_2 is true
i.e.,	is false iff	S_1 is true and	S_2 is false
$S_1 \Leftrightarrow S_2$	is true iff	$S_1 \Rightarrow S_2$ is true and	$S_2 \Rightarrow S_1$ is true

First-order logic

- First-order logic (FOL) models the world in terms of
  - **Objects,** which are things with individual identities
  - **Properties** of objects that distinguish them from other objects
  - **Relations** that hold among sets of objects
  - **Functions,** which are a subset of relations where there is only one "value" for any given "input"
- Examples:
  - Objects: Students, lectures, companies, cars ...
  - Relations: Brother-of, bigger-than, outside, part-of, has-color, occurs-after, owns, visits, precedes, ...
  - Properties: blue, oval, even, large, ...
  - Functions: father-of, best-friend, second-half, one-more-than ...

First-order logic (2)

- Uses **Terms** for referring to **Objects**
  - a constant symbol, a variable symbol, or an n-place function of n terms
  - *e.g. John, x, LeftLeg(John)*
- Uses **Predicate Symbols** for referring to **Relations**
  - e.g. *Brother*
- **Atomic Sentences** state facts. e.g. *Brother(John, Sam)*
- **Complex Sentences** are formed from atomic sentences connected by the logical connectives: $\neg P$, $P \vee Q$, $P \wedge Q$, $P \rightarrow Q$, $P \leftrightarrow Q$
- **Quantified sentences** add quantifiers $\forall$ and $\exists$

First-order logic (3)

- A **well-formed formula (wff)** is a sentence containing no “free” variables. That is, all variables are “bound” by universal or existential quantifiers.
  - $(\forall x)P(x,y)$ has x bound as a universally quantified variable, but y is free.

Entailment and derivation

- **Entailment: KB |= Q**
  - The needle (Q) being in the haystack
  - Q is entailed by KB if and only if the conclusion is true in every logically possible world in which all the premises in KB are true.
- **Derivation: KB |- Q**
  - Finding the needle (Q) in the haystack (KB)
  - We can derive Q from KB if there is a proof consisting of a sequence of valid **inference** steps starting from the premises in KB and resulting in Q

Inference

- $KB \vdash_i \alpha$ = sentence α can be derived from *KB* by procedure *i*
- Soundness: derivations produce only entailed sentences
- Completeness: derivations can produce all entailed sentences

Inference rules

- **Logical inference** is used to create new sentences that logically follow from a given set of predicate calculus sentences (KB).
- An inference rule is **sound** if every sentence X produced by an inference rule operating on a KB logically follows from the KB. (That is, the inference rule does not create any contradictions)
- An inference rule is **complete** if it is able to produce every expression that logically follows from (is entailed by) the KB. (Note the analogy to complete search algorithms.)

Logical Inference

Chapter 9

Entailment for FOL is semidecidable

- Algorithms exist that say yes to every entailed sentence, but no algorithm exists that also says no to every nonentailed sentence.

Inference rules for quantifiers

- Applied to sentences with quantifiers to obtain sentences without quantifiers
- Universal instantiation
  - $\forall x\ P(x)\ \therefore\ P(A)$
- Existential instantiation
  - $\exists x\ P(x)\ \therefore P(F)$ $\leftarrow$ **skolem constant F**

Universal instantiation (a.k.a. universal elimination)

- We can infer any sentence obtained by substituting a ground term

Universal instantiation (a.k.a. universal elimination)

- If $(\forall x)\ P(x)$ is true, then $P(C)$ is true, where C is *any* constant in the domain of x
- Example:
  - $(\forall x)$ eats(Ziggy, x) $\Rightarrow$ eats(Ziggy, IceCream)
- The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only

Existential instantiation (a.k.a. existential elimination)

- The variable is replaced by a single new constant symbol that does not appear elsewhere in the knowledge base.

Existential instantiation (a.k.a. existential elimination)

- From $(\exists x)\ P(x)$ infer $P(c)$
- Example:
  - $(\exists x)$ eats(Ziggy, x) $\rightarrow$ eats(Ziggy, Stuff)
- Note that the variable is replaced by a **brand-new constant** not occurring in this or any other sentence in the KB
- Also known as skolemization; constant is a **skolem constant**
- We don't want to accidentally draw other inferences about it by introducing the constant
- Convenient to use this to reason about the unknown object, rather than constantly manipulating the existential quantifier

Inference in FOL

- Propositionalization
  - Use inference rules for quantifiers to convert the knowledge base from first order logic to propositional logic
- Use propositional inference
- Approach is slow, unless the domain is small

Inference in FOL (2)

- Forward Chaining Algorithm
  - derive the goal from the axioms
- Backward chaining Algorithm
  - start with the goal and attempt to resolve them working backwards

- Notice is the same idea as for propositional logic.
- We only need to take care of the quantifiers:
  - Generalized Modus Ponens (GMP)

Automating FOL inference with Generalized Modus Ponens

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 $(\forall x)(P(x) \wedge Q(x)) \rightarrow R(x)$ derive $R(c)$
- General case: **Given**
  - **atomic sentences** $P_1, P_2, \ldots, P_N$
  - **implication sentence** $(Q_1 \wedge Q_2 \wedge \ldots \wedge Q_N) \rightarrow R$
    - $Q_1, \ldots, Q_N$ and R are atomic sentences
  - **substitution** $\text{subst}(\theta, P_i) = \text{subst}(\theta, Q_i)$ for $i=1,\ldots,N$
  - **Derive new sentence: subst(θ, R)**
- Substitutions
  - $\text{subst}(\theta, \alpha)$ denotes the result of applying a set of substitutions defined by θ to the sentence α
  - A substitution list $\theta = \{v_1/t_1, v_2/t_2, \ldots, v_n/t_n\}$ 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)

Generalized Modus Ponens

- $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$
- $\forall y\ Greedy(y)$
- King(John)

- ? Evil(John)

- Applying the **substitution** {x/John, y/John} we can infer Evil(John)

Generalized Modus Ponens

- If there is some substitution θ that makes each of the conjuncts of the premise of the implication identical to the sentences already in the knowledge base, then we can assert the conclusion of the implication, after applying θ.

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

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 instantiation 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**

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

Horn clauses II

- Special cases
  - $P_1 \wedge P_2 \wedge \ldots P_n \rightarrow Q$
  - $P_1 \wedge P_2 \wedge \ldots P_n \rightarrow \text{false}$
  - $\text{true} \rightarrow Q$
- These are not Horn clauses:
  - $p(a) \vee q(a)$
  - $(P \wedge Q) \rightarrow (R \vee S)$

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) $\rightarrow$ sneeze(X)
  - cat(Y) $\wedge$ allergic-to-cats(X) $\rightarrow$ 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 $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$ **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, θ)
  end

---

procedure FIND-AND-INFER(KB, *premises*, *conclusion*, θ)

  if *premises* $= [\,]$ **then**
    FORWARD-CHAIN(KB, SUBST(θ, *conclusion*))
  else for each p' **in** KB such that UNIFY(p', SUBST(θ, FIRST(*premises*))) $= \theta_2$ **do**
    FIND-AND-INFER(KB, REST(*premises*), *conclusion*, COMPOSE(θ, θ_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) $\rightarrow$ sneeze(X)
  - cat(Y) $\wedge$ allergic-to-cats(X) $\rightarrow$ allergies(X)
  - cat(Felix)
  - allergic-to-cats(Lise)
- Goal:
  - sneeze(Lise)

Backward chaining algorithm

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

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

---

function BACK-CHAIN-LIST(KB, $qlist$, θ) **returns** a set of substitutions
inputs: KB, a knowledge base
$qlist$, a list of conjuncts forming a query (θ already applied)
θ, the current substitution
static: $answers$, a set of substitutions, initially empty

if $qlist$ is empty **then return** $\{\theta\}$
$q \leftarrow$ FIRST($qlist$)
for each q'_i **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
Add COMPOSE(θ, θ_i) to $answers$
end
for each sentence $(p_1 \land \ldots \land p_n \Rightarrow q'_i)$ **in** KB such that $\theta_i \leftarrow$ UNIFY(q, q'_i) succeeds **do**
$answers \leftarrow$ BACK-CHAIN-LIST(KB, SUBST(θ_i, $[p_1 \ldots p_n]$), COMPOSE(θ, θ_i)) $\cup$ $answers$
end
return the union of BACK-CHAIN-LIST(KB, REST($qlist$), θ) for each $\theta \in answers$

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 $P(x) \vee R(x)$

Automating FOL inference with resolution

Resolution

- Resolution is a **sound** and **complete** inference procedure for FOL
- Main idea: Two clauses can be resolved if they contain complementary literals
- 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 (2)

- Propositional literals are complementary if one if the negation of the other

- FOL literals are complementary if one **unifies** with the negation of the other

Resolution in first-order logic

- Given sentences

  $P_1 \vee \ldots \vee P_n$

  $Q_1 \vee \ldots \vee Q_m$

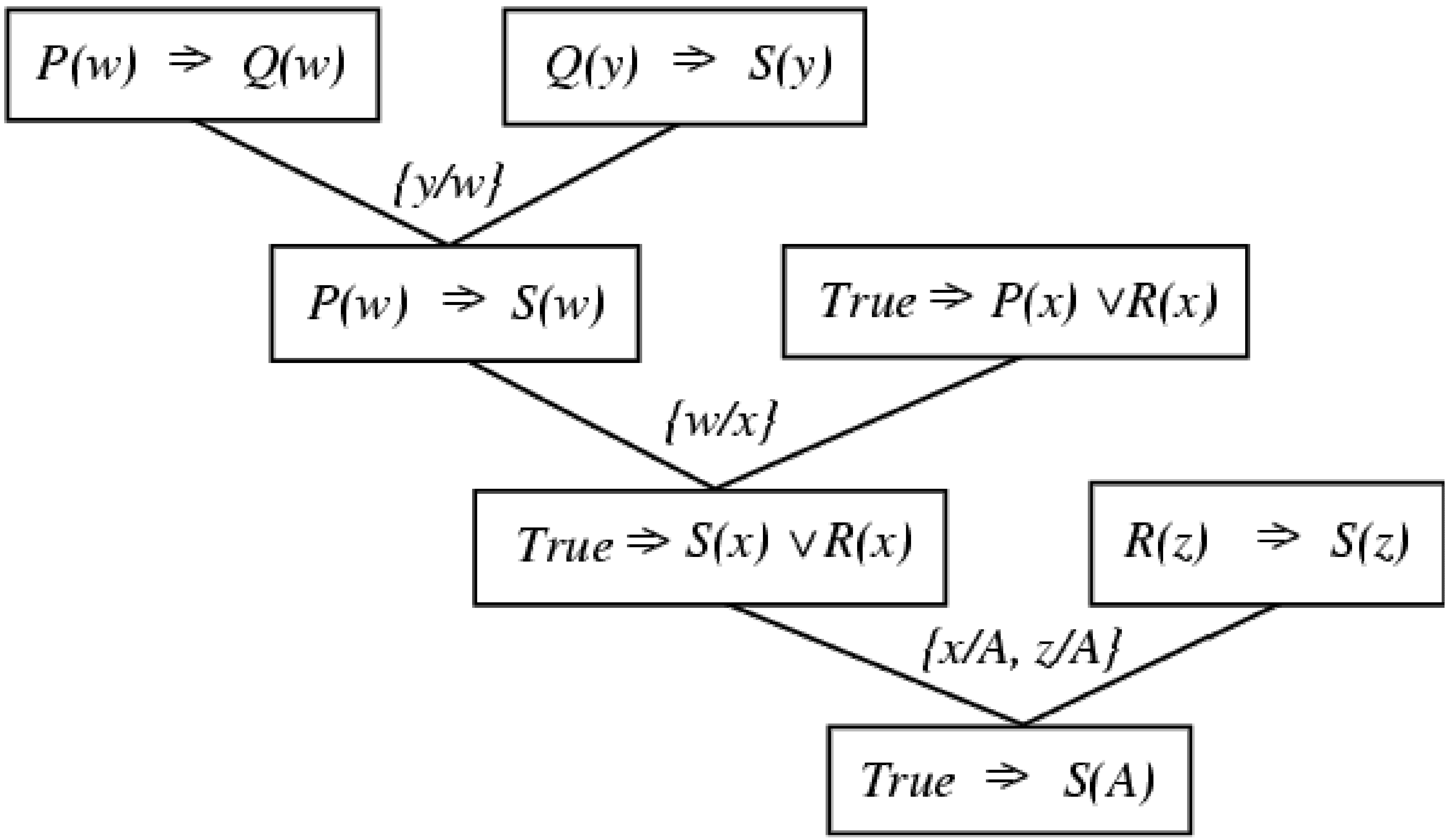
- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,

- 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 $\mathbf{P(x, f(a)) \vee P(x, f(y)) \vee Q(y)}$
  - and clause $\mathbf{\neg P(z, f(a)) \vee \neg Q(z)}$
  - derive resolvent $\mathbf{P(z, f(y)) \vee Q(y) \vee \neg Q(z)}$
  - using $\boldsymbol{\theta = \{x/z\}}$

A resolution proof tree

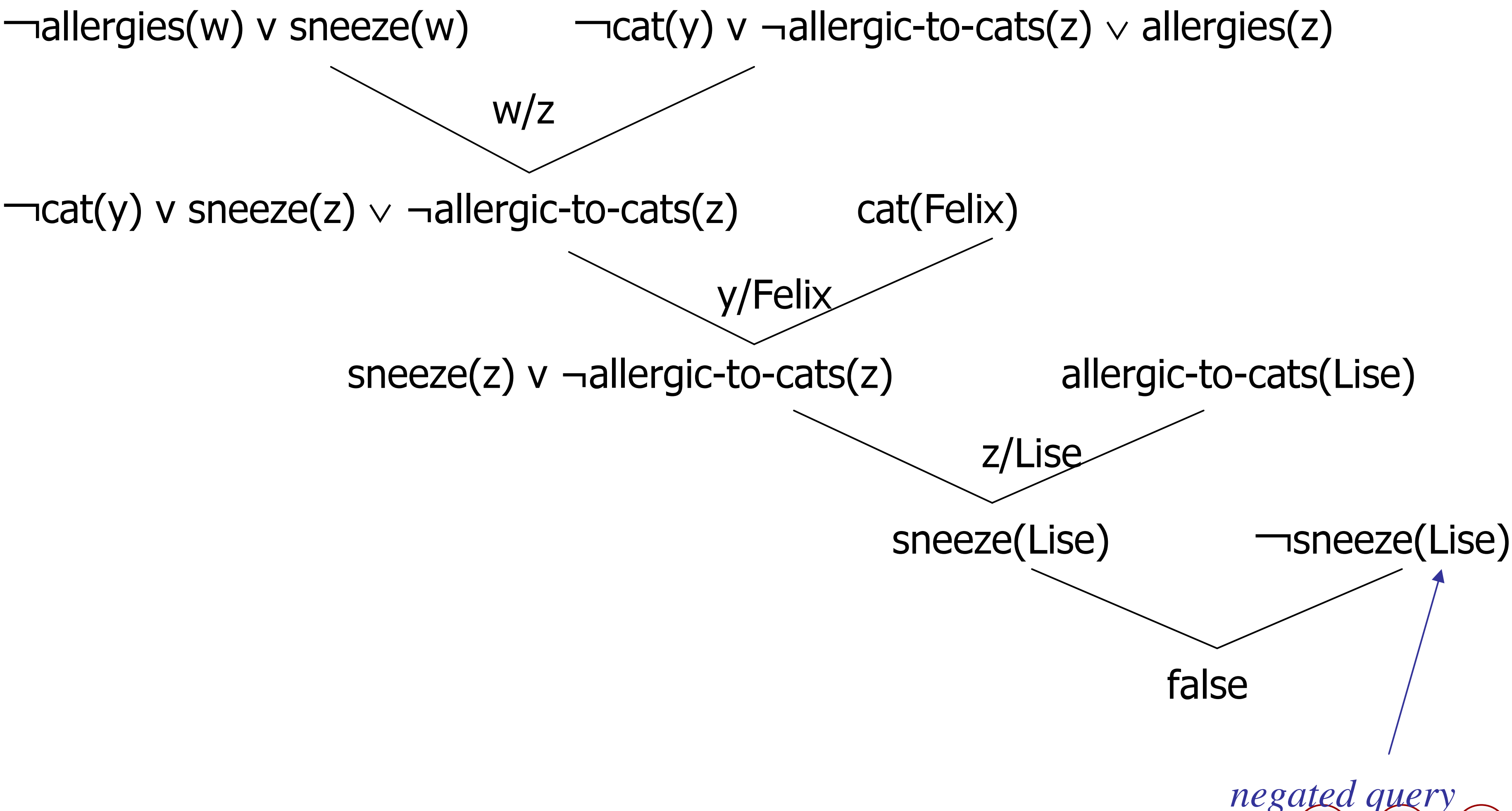


Resolution refutation

- 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



Required intermediate tasks

- 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**) θ: **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

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

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

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

$$(\forall \mathbf{x})(\mathbf{P}(\mathbf{x}) \rightarrow ((\forall \mathbf{y})(\mathbf{P}(\mathbf{y}) \rightarrow \mathbf{P}(\mathbf{f}(\mathbf{x},\mathbf{y}))) \wedge \neg(\forall \mathbf{y})(\mathbf{Q}(\mathbf{x},\mathbf{y}) \rightarrow \mathbf{P}(\mathbf{y}))))$$

2. Eliminate $\rightarrow$

$$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge \neg(\forall y)(\neg Q(x,y) \vee P(y))))$$

3. Reduce scope of negation

$$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists y)(Q(x,y) \wedge \neg P(y))))$$

4. Standardize variables

$$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (\exists z)(Q(x,z) \wedge \neg P(z))))$$

5. Eliminate existential quantification

$$(\forall x)(\neg P(x) \vee ((\forall y)(\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \neg P(g(x)))))$$

6. Drop universal quantification symbols

$$(\neg P(x) \vee ((\neg P(y) \vee P(f(x,y))) \wedge (Q(x,g(x)) \wedge \neg P(g(x)))))$$

Example

7. Convert to conjunction of disjunctions

$(\neg P(x) \vee \neg P(y) \vee P(f(x,y))) \wedge (\neg P(x) \vee Q(x,g(x))) \wedge$
$(\neg P(x) \vee \neg P(g(x)))$

8. Create separate clauses

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

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

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

9. Standardize variables

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

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

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

Unification

(see slides 25-28)

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

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))

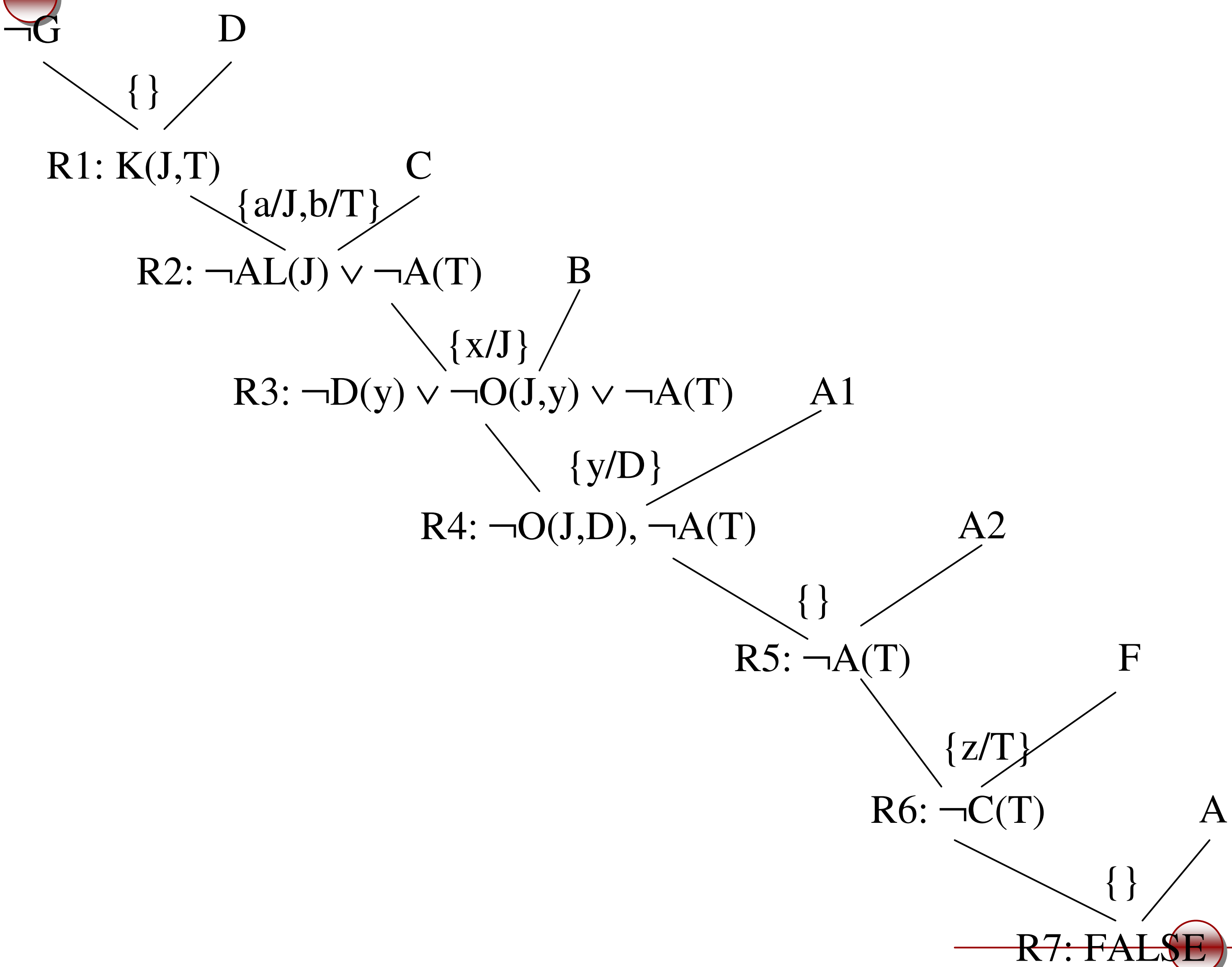
- **Add the negation of query:**

  ¬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

- Repeated applications of the resolution inference rule will eventually find a proof if one exists.
- Some strategies help to find proofs efficiently

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:
  - Unit preference
  - 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

1,4 10. ¬Battery-OK ∨ ¬Bulbs-OK

1,5 11. ¬Bulbs-OK ∨ Headlights-Work

2,3 12. ¬Battery-OK ∨ ¬Starter-OK ∨ Empty-Gas-Tank ∨ Flat-Tire ∨ Car-OK

2,5 13. ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts

2,6 14. ¬Battery-OK ∨ Empty-Gas-Tank ∨ Engine-Starts

2,7 15. ¬Battery-OK ¬ Starter-OK ∨ Engine-Starts

16. … [and we're still only at Level 1!]

Unit preference (unit resolution)

- **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

1,5 10. ¬Bulbs-OK ∨ Headlights-Work
2,5 11. ¬Starter-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
2,6 12. ¬Battery-OK ∨ Empty-Gas-Tank ∨ Engine-Starts
2,7 13. ¬Battery-OK ¬ Starter-OK ∨ Engine-Starts
3,8 14. ¬Engine-Starts ∨ Flat-Tire
3,9 15. ¬Engine-Starts ¬ 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. ¬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) $\vee$ 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
    ($\therefore$ 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

Using FOL

Using FOL

- **Domain M:** the set of all objects in the world (of interest)
- **Assertions:** sentences added to KB by using TELL (as in propositional logic)
  - TELL (KB, Person(Richard))
- **Queries/Goals:** ask questions of the KB. Any query that is logically entailed by the knowledge base should be answered affirmatively.
  - ASK (KB, Person(Richard))

Using FOL (2)

- **Can be cumbersome and prone to mistakes**

- **Database systems approach**
  - Unique names assumption
  - Closed-World assumption (not unknown truth)
  - Domain closure

Using forward chaining

- Systems implementing forward chaining are known as **production systems**.
  - Perform efficient updates with very large rule sets
  - Example: Datalog

Using backward chaining

- Backward chaining is used in **logic programming systems**.
  - Employ sophisticated compiler technology to provide fast inference.
  - It is the mechanism underlying the execution of Prolog programs

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

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)

Prolog (2)

- Proof method of Prolog is resolution refutation
- Backward reasoning with sub-goaling: we assert the negated goal and try to work backwards, unifying and resolving clauses until we get to the empty clause

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