Inference in First-Order Logic

- Want to be able to draw logically sound conclusions from a knowledge-base expressed in first-order logic.

- Several styles of inference:
  - Forward chaining
  - Backward chaining
  - Resolution refutation

- Properties of inference procedures:
  - Soundness: If $A \vdash B$ then $A \models B$
  - Completeness: If $A \models B$ then $A \vdash B$

- Forward and backward chaining are sound and can be reasonably efficient but are incomplete.

- Resolution is sound and complete for FOPL but can be very inefficient.

Inference Rules for Quantifiers

- Let $\text{SUBST}(\theta, \alpha)$ denote the result of applying a substitution or binding list $\theta$ to the sentence $\alpha$.
  - $\text{SUBST}((x/Tom, y/Fred), \text{Uncle}(x,y)) = \text{Uncle}(Tom, Fred)$

- Inference rules
  - Universal Elimination: $\forall \alpha \vdash \text{SUBST}((v/g), \alpha)$ for any sentence, $\alpha$, variable, $v$, and ground term, $g$
    - $\forall x \text{Loves}(x, \text{FOPC}) \vdash \text{Loves}(Ray, \text{FOPC})$
  - Existential Elimination: $\exists v \alpha \vdash \text{SUBST}((v/k), \alpha)$ for any sentence, $\alpha$, variable, $v$, and constant symbol, $k$, that doesn’t occur elsewhere in the KB (Skolem constant)
    - $\exists x (\text{Owns(Mary,x) \land Cat(x))} \vdash \text{Owns(Mary,MarysCat) \land Cat(MarysCat)}$
  - Existential Introduction: $\alpha \vdash \exists v \text{SUBST}((v/g), \alpha)$ for any sentence, $\alpha$, variable, $v$, that does not occur in $\alpha$, and ground term, $g$, that does occur in $\alpha$
    - $\text{Loves}(Ray, \text{FOPC}) \vdash \exists x \text{Loves}(x, \text{FOPC})$

Sample Proof

1) $\forall x,y (\text{Parent}(x,y) \land \text{Male}(x) \Rightarrow \text{Father}(x,y))$
2) $\text{Parent}(\text{Tom}, \text{John})$
3) $\text{Male}(\text{Tom})$

Using Universal Elimination from 1)
4) $\forall y (\text{Parent}(\text{Tom}, y) \land \text{Male}(\text{Tom}) \Rightarrow \text{Father}(\text{Tom}, y))$

Using Universal Elimination from 4)
5) $\text{Parent}(\text{Tom}, \text{John}) \land \text{Male}(\text{Tom}) \Rightarrow \text{Father}(\text{Tom}, \text{John})$

Using And Introduction from 2) and 3)
6) $\text{Parent}(\text{Tom}, \text{John}) \land \text{Male}(\text{Tom})$

Using Modes Ponens from 5) and 6)
7) $\text{Father}(\text{Tom}, \text{John})$
Generalized Modus Ponens

- Combines three steps of “natural deduction” (Universal Elimination, And Introduction, Modus Ponens) into one.

- Provides direction and simplification to the proof process for standard inferences.

**Generalized Modus Ponens:**

\[ p_1', p_2', \ldots, p_n', (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \mid \text{SUBST}(\theta, q) \]

where \( \theta \) is a substitution such that for all \( i \)

\[ \text{SUBST}(\theta, p_i') = \text{SUBST}(\theta, p_i) \]

1) \( \forall x, y (\text{Parent}(x, y) \land \text{Male}(x) \Rightarrow \text{Father}(x, y)) \)
2) \( \text{Parent}(\text{Tom}, \text{John}) \)
3) \( \text{Male}(\text{Tom}) \)
4) \( \text{Father}(\text{Tom}, \text{John}) \)

Canonical Form

- In order to utilize generalized Modus Ponens, all sentences in the KB must be in the form of Horn sentences:

\[ \forall v_1, v_2, \ldots, v_n p_1 \land p_2 \land \ldots \land p_n \Rightarrow q \]

- Also called Horn clauses, where a clause is a disjunction of literals, because they can be rewritten as disjunctions with at most one non-negated literal.

\[ \forall v_1, v_2, \ldots, v_n \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \lor q \]

If \( \theta \) is the constant False, this simplifies to

\[ \forall v_1, v_2, \ldots, v_n \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \]

Otherwise the sentence is called a definite clause (exactly one non-negated literal).

- Single positive literals (facts) are Horn clauses with no antecedent.

- Quantifiers can be dropped since all variables can be assumed to be universally quantified by default.

- Many statements can be transformed into Horn clauses, but many cannot (e.g. \( P(x) \lor Q(x), \neg P(x) \))

Unification

- In order to match antecedents to existing literals in the KB, need a pattern matching routine.

- UNIFY\((p, q)\) takes two atomic sentences and returns a substitution that makes them equivalent.

\[ \text{UNIFY}(p, q) = \theta \text{ where } \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q) \]

\( \theta \) is called a unifier.

**Examples**

\[ \text{UNIFY}(\text{Parent}(x, y), \text{Parent}(\text{Tom}, \text{John})) = \{x/\text{Tom}, y/\text{John}\} \]

\[ \text{UNIFY}(\text{Parent}(\text{Tom}, x), \text{Parent}(\text{Tom}, \text{John})) = \{x/\text{John}\} \]

\[ \text{UNIFY}(\text{Likes}(x, y), \text{Likes}(z, \text{FOPC})) = \{x/z, y/\text{FOPC}\} \]

\[ \text{UNIFY}(\text{Likes}(\text{Tom}, y), \text{Likes}(z, \text{FOPC})) = \{z/\text{Tom}, y/\text{FOPC}\} \]

\[ \text{UNIFY}(\text{Likes}(\text{Tom}, y), \text{Likes}(y, \text{FOPC})) = \text{fail} \]

\[ \text{UNIFY}(\text{Likes}(\text{Tom}, \text{Tom}), \text{Likes}(x, x)) = \{x/\text{Tom}\} \]

\[ \text{UNIFY}(\text{Likes}(\text{Tom}, \text{Fred}), \text{Likes}(x, x)) = \text{fail} \]

Unification (cont.)

- Exact variable names used in sentences in the KB should not matter.

- But if \( \text{Likes}(x, \text{FOPC}) \) is a formula in the KB, it does not unify with \( \text{Likes}(\text{John}, x) \) but does unify with \( \text{Likes}(\text{John}, y) \).

- To avoid such conflicts, one can standardize apart one of the arguments to UNIFY to make its variables unique by renaming them.

\[ \text{Likes}(x, \text{FOPC}) \Rightarrow \text{Likes}(x_1, \text{FOPC}) \]

\[ \text{UNIFY}(\text{Likes}(\text{John}, x), \text{Likes}(x_1, \text{FOPC})) = \{x_1/\text{John}, x/\text{FOPC}\} \]

- There are many possible unifiers for some atomic sentences.

\[ \text{UNIFY}(\text{Likes}(x, y), \text{Likes}(z, \text{FOPC})) = \{x/z, y/\text{FOPC}\} \]

\[ \{x/\text{John}, z/\text{John}, y/\text{FOPC}\} \]

\[ \{x/\text{Fred}, z/\text{Fred}, y/\text{FOPC}\} \]

\[ \ldots \]

UNIFY should return the most general unifier which makes the least commitment to variable values.
Forward Chaining

- Use modus ponens to always deriving all consequences from new information.
- Inferences cascade to draw deeper and deeper conclusions.

To avoid looping and duplicated effort, must prevent addition of a sentence to the KB which is the same as one already present.

Must determine all ways in which a rule (Horn clause) can match existing facts to draw new conclusions.

Forward Chaining Algorithm

- A sentence is a renaming of another if it is the same except for a renaming of the variables.
- The composition of two substitutions combines the variable bindings of both such that:

  \[ \text{SUBST}(\text{COMPOSE}(\theta_1, \theta_2), p) = \text{SUBST}(\theta_2, \text{SUBST}(\theta_1, p)) \]

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

\[
\begin{align*}
\text{procedure} & \quad \text{FIND-AND-INFERENCE}(KB, \text{premises}, \text{conclusion}, \theta) \\
\text{if} & \quad \text{premises} = [] \text{ then} \\
& \quad \text{FORWARD-CHAIN}(KB, \text{SUBST}(\theta, \text{conclusion})) \\
\text{else} & \quad \text{for each } p' \text{ in } KB \text{ such that UNIFY}(p', \text{SUBST}(\theta, \text{FIRST(premises)})) = \theta \text{ do} \\
& \quad \quad \text{FIND-AND-INFERENCE}(KB, \text{REST(premises)}, \text{conclusion}, \text{COMPOSE}(\theta, \theta_2)) \\
\text{end}
\end{align*}
\]

Forward Chaining Example

Assume in KB
1) Parent(x,y) \land Male(x) \Rightarrow Father(x,y)
2) Father(x,y) \land Father(x,z) \Rightarrow Sibling(y,z)

Add to KB
3) Parent(Tom,John)

Rule 1) tried but can’t “fire”

Add to KB
4) Male(Tom)

Rule 1) now satisfied and triggered and adds:
5) Father(Tom, John)

Rule 2) now triggered and adds:
6) Sibling(John,John) \{x/Tom, y/John, z/John\}

Add to KB
7) Parent(Tom,Fred)

Rule 1) triggered again and adds:
8) Father(Tom,Fred)

Rule 2) triggered again and adds:
9) Sibling(Fred,Fred) \{x/Tom, y/Fred, z/Fred\}

Rule 2) triggered again and adds:
10) Sibling(John,Fred) \{x/Tom, y/John, z/Fred\}

Rule 2) triggered again and adds:
11) Sibling(Fred, John) \{x/Tom, y/Fred, z/John\}

Problems with Forward Chaining

- Inference can explode forward and may never terminate.

Even(x) \Rightarrow Even(plus(x,2))
Integer(x) \Rightarrow Even(times(2,x))
Even(x) \Rightarrow Integer(x)
Even(2)

- Inference is not directed towards any particular conclusion or goal. May draw lots of irrelevant conclusions.
**Backward Chaining**

- Start from query or atomic sentence to be proven and look for ways to prove it.

- Query can contain variables which are assumed to be existentially quantified.

\[ \text{Sibling}(x, \text{John}) \quad ? \]
\[ \text{Father}(x, y) \quad ? \]

Inference process should return all sets of variable bindings that satisfy the query.

- First try to answer query by unifying it to all possible facts in the KB.

- Next try to prove it using a rule whose consequent unifies with the query and then try to recursively prove all of its antecedents.

**Backward Chaining Algorithm**

- Given a conjunction of queries, first get all possible answers to the first conjunct and then for each resulting substitution try to prove all of the remaining conjuncts.

- Assume variables in rules are renamed (standardized apart) before each use of a rule.

\[ \text{function } \text{BACK}-\text{CHAIN}(KB, q) \text{ returns a set of substitutions} \]

\[ \text{function } \text{BACK}-\text{CHAIN}-\text{LIST}(KB, qlist, \theta) \text{ returns a set of substitutions} \]

**Function Definition:**

\[ \text{function } \text{BACK}-\text{CHAIN}-\text{LIST}(KB, qlist, \theta) \text{ returns a set of substitutions} \]

**Inputs:**
- \( KB \), a knowledge base
- \( qlist \), a list of conjuncts forming a query (\( \theta \) already applied)
- \( \theta \), 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 \( \theta_i \) in \( KB \) such that \( \theta_i \leftarrow \text{UNIFY}(q, \theta_i) \) succeeds

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

  End

  For each sentence \( (p_1 \land \ldots \land p_n) \) \( q \) in \( KB \) such that \( \theta_i \leftarrow \text{UNIFY}(q, \theta_i) \) succeeds

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

  End

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

**Backchaining Examples**

**KB:**
1) \( \text{Parent}(x, y) \land \text{Male}(x) \Rightarrow \text{Father}(x, y) \)
2) \( \text{Father}(x, y) \land \text{Father}(x, z) \Rightarrow \text{Sibling}(y, z) \)
3) \( \text{Parent}(\text{Tom}, \text{John}) \)
4) \( \text{Male}(\text{Tom}) \)
7) \( \text{Parent}(\text{Tom}, \text{Fred}) \)

Query: \( \text{Parent}(\text{Tom}, x) \)
Answers: (\{x/\text{John}\}, \{x/\text{Fred}\})

Query: \( \text{Father}(f, s) \)
Subgoal: \( \text{Parent}(f, s) \land \text{Male}(f) \)
- \( \{f/\text{Tom}, s/\text{John}\} \)
  Subgoal: \( \text{Male}(\text{Tom}) \)
  Answer: \( \{f/\text{Tom}, s/\text{John}\} \)
  \( \{f/\text{Tom}, s/\text{Fred}\} \)
  Subgoal: \( \text{Male}(\text{Tom}) \)
  Answer: \( \{f/\text{Tom}, s/\text{Fred}\} \)
Answers: (\{f/\text{Tom}, s/\text{John}\}, \{f/\text{Tom}, s/\text{Fred}\})

**Backchaining Examples (cont)**

Query: \( \text{Sibling}(a, b) \)
Subgoal: \( \text{Father}(f, a) \land \text{Father}(f, b) \)
- \( \{f/\text{Tom}, a/\text{John}\} \)
  Subgoal: \( \text{Father}(\text{Tom}, b) \)
  \( \{b/\text{John}\} \)
  Answer: \( \{f/\text{Tom}, a/\text{John}, b/\text{John}\} \)
  \( \{b/\text{Fred}\} \)
  Answer: \( \{f/\text{Tom}, a/\text{John}, b/\text{Fred}\} \)
  \( \{f/\text{Tom}, a/\text{Fred}\} \)
  Subgoal: \( \text{Father}(\text{Tom}, b) \)
  \( \{b/\text{John}\} \)
  Answer: \( \{f/\text{Tom}, a/\text{Fred}, b/\text{John}\} \)
  \( \{b/\text{Fred}\} \)
  Answer: \( \{f/\text{Tom}, a/\text{Fred}, b/\text{Fred}\} \)
Answers: (\{f/\text{Tom}, a/\text{John}, b/\text{John}\}, \{f/\text{Tom}, a/\text{John}, b/\text{Fred}\}
\{f/\text{Tom}, a/\text{Fred}, b/\text{John}\}, \{f/\text{Tom}, a/\text{Fred}, b/\text{Fred}\})
Incompleteness

• Rule-based inference is not complete, but is reasonably efficient and useful in many circumstances.

• Still can be exponential or not terminate in worst case.

• Incompleteness example:

  \[ \neg P(x) \Rightarrow R(x) \quad \text{(not Horn)} \]
  \[ Q(x) \Rightarrow S(x) \]
  \[ R(x) \Rightarrow S(x) \]

  Entails \( S(A) \) for any constant \( A \) but not inferable from modus ponens.

Completeness

• In 1930 Gödel showed that a complete inference procedure for FOPC existed, but did not demonstrate one (non-constructive proof).

• In 1965, Robinson showed a resolution inference procedure that was sound and complete for FOPC.

  • However, the procedure may not halt if asked to prove a theorem that is not true, it is said to be semidecidable (a type of undecidability).

    If a conclusion \( C \) is entailed by the KB then the procedure will eventually terminate with a proof. However if it is not entailed, it may never halt.

  • It does not follow that either \( C \) or \( \neg C \) is entailed by a KB (may be independent). Therefore trying to prove both a conjecture and its negation does not help.

  • Inconsistency of a KB is also semidecidable.

Resolution

• Propositional version.

  \[ \{ \alpha \lor \beta, \neg \beta \lor \gamma \} \vdash \alpha \lor \gamma \quad \text{OR} \quad \{ \neg \alpha \Rightarrow \beta, \beta \Rightarrow \gamma \} \vdash \neg \alpha \Rightarrow \gamma \]

  Reasoning by cases \hspace{1cm} \text{OR} \hspace{1cm} \text{transitivity of implication}

• First-order form

  For two literals \( p_j \) and \( q_k \) in two clauses

  \[ p_1 \lor \ldots \lor p_j \lor \ldots \lor p_m \]
  \[ q_1 \lor \ldots \lor q_k \lor \ldots \lor q_n \]

  such that \( \theta = \text{UNIFY}(p_j, \neg q_k) \), derive

  \[ \text{SUBST}(\theta, p_1 \lor \ldots \lor p_j \lor \ldots \lor p_m \lor q_1 \lor \ldots \lor q_{k-1} \lor \ldots \lor q_{n}) \]

• Can also be viewed in implicational form where all negated literals are in a conjunctive antecedent and all positive literals in a disjunctive conclusion.

  \[ \neg p_1 \lor \ldots \lor \neg p_m \lor q_1 \lor \ldots \lor q_n \iff \]
  \[ p_1 \land \ldots \land p_m \Rightarrow q_1 \lor \ldots \lor q_n \]

Conjunctive Normal Form (CNF)

• For resolution to apply, all sentences must be in conjunctive normal form, a conjunction of disjunctions of literals

  \[ (a_1 \lor \ldots \lor a_m) \land \]
  \[ (b_1 \lor \ldots \lor b_n) \land \]
  \[ \ldots \land \]
  \[ (x_1 \lor \ldots \lor x_v) \]

• Representable by a set of clauses (disjunctions of literals)

• Also representable as a set of implications (INF).

• Example

  \[ \begin{array}{ccc}
  \text{Initial} & \text{CNF} & \text{INF} \\
  P(x) \Rightarrow Q(x) & \neg P(x) \lor Q(x) & P(x) \Rightarrow Q(x) \\
  \neg P(x) \Rightarrow R(x) & P(x) \lor R(x) & \text{True} \Rightarrow P(x) \lor R(x) \\
  Q(x) \Rightarrow S(x) & \neg Q(x) \lor S(x) & Q(x) \Rightarrow S(x) \\
  R(x) \Rightarrow S(x) & \neg R(x) \lor S(x) & R(x) \Rightarrow S(x)
  \end{array} \]
Resolution Proofs

- INF (CNF) is more expressive than Horn clauses.
- Resolution is simply a generalization of modus ponens.
- As with modus ponens, chains of resolution steps can be used to construct proofs.

\[
P(w) \Rightarrow Q(w) \quad Q(y) \Rightarrow S(y)
\]

\[
P(w) \Rightarrow S(w)
\]

\[
[w/x]
\]

\[
S(w) \Rightarrow P(x) R(x)
\]

\[
[y/w]
\]

\[
R(z) \Rightarrow S(z)
\]

\[
[z/x]
\]

\[
True \Rightarrow S(A)
\]

- Factoring removes redundant literals from clauses

\[
S(A) \lor S(A) \Rightarrow S(A)
\]

Refutation Proofs

- Unfortunately, resolution proofs in this form are still incomplete.
- For example, it cannot prove any tautology (e.g. \(P \lor \neg P\)) from the empty KB since there are no clauses to resolve.
- Therefore, use proof by contradiction (refutation, reductio ad absurdum). Assume the negation of the theorem \(P\) and try to derive a contradiction (False, the empty clause).

\[
(KB \land \neg P \Rightarrow False) \iff KB \Rightarrow P
\]

Resolution Theorem Proving

- Convert sentences in the KB to CNF (clausal form)
- Take the negation of the posited theorem (query), convert it to CNF, and add it to the KB.
- Repeatedly apply the resolution rule to derive new clauses.
- If the empty clause (False) is eventually derived, stop and conclude that the proposed theorem is true.

Conversion to Clausal Form

- Eliminate implications and biconditionals by rewriting them.

\[
p \Rightarrow q \Rightarrow \neg p \lor q \quad p \Leftrightarrow q \Leftrightarrow (\neg p \lor q) \land (p \lor \neg q)
\]

- Move \(\neg\) inward to only be a part of literals by using deMorgan's laws and quantifier rules.

\[
\neg (p \land q) \Rightarrow \neg p \lor \neg q
\]

\[
\neg (p \lor q) \Rightarrow \neg p \land \neg q
\]

\[
\neg \forall x p \Rightarrow \exists x \neg p
\]

\[
\neg \exists x p \Rightarrow \forall x \neg p
\]

\[
\neg p \Rightarrow p
\]

- Standardize variables to avoid use of the same variable name by two different quantifiers.

\[
\forall x P(x) \lor \exists x P(x) \Rightarrow \forall x_1 P(x_1) \lor \exists x_2 P(x_2)
\]

- Move quantifiers left while maintaining order. Renaming above guarantees this is a truth-preserving transformation.

\[
\forall x_1 P(x_1) \lor \exists x_2 P(x_2) \Rightarrow \forall x_1 \exists x_2 (P(x_1) \lor P(x_2))
\]
**Sample Clausal Conversion**

\[
\forall x ((\text{Prof}(x) \lor \text{Student}(x)) \Rightarrow (\exists y (\text{Class}(y) \land \text{Has}(x,y)) \land \\
\exists y (\text{Book}(y) \land \text{Has}(x,y))))
\]

\[
\forall x (\neg (\text{Prof}(x) \lor \text{Student}(x)) \lor (\exists y (\text{Class}(y) \land \text{Has}(x,y)) \land \\
\exists y (\text{Book}(y) \land \text{Has}(x,y))))
\]

\[
\forall x (\neg (\text{Prof}(x) \land \neg \text{Student}(x)) \lor (\exists y (\text{Class}(y) \land \text{Has}(x,y)) \land \\
\exists y (\text{Book}(y) \land \text{Has}(x,y))))
\]

\[
\forall x \exists y \exists z ((\neg (\text{Prof}(x) \land \neg \text{Student}(x))) \lor ((\text{Class}(y) \land \text{Has}(x,y)) \land \\
(\text{Book}(z) \land \text{Has}(x,z))))
\]

\[
(\neg \text{Prof}(x) \land \neg \text{Student}(x)) \lor (\text{Class}(f(x)) \land \text{Has}(x,f(x)) \land \\
\text{Book}(g(x)) \land \text{Has}(x,g(x)))
\]

**Sample Resolution Proof**

- **Jack owns a dog.**
  Every dog owner is an animal lover.
  No animal lover kills an animal.
  Either Jack or Curiosity killed Tuna the cat.
  Did Curiosity kill the cat?

- **A) **\(\exists x \text{Dog}(x) \land \text{Owns}(Jack,x)\)
  B) \(\forall x (\exists y \text{Dog}(y) \land \text{Owns}(x,y)) \Rightarrow \text{AnimalLover}(x)\)
  C) \(\forall x \text{AnimalLover}(x) \Rightarrow (\forall y \text{Animal}(y) \Rightarrow \neg \text{Kills}(x,y))\)
  D) \(\text{Kills}(Jack,Tuna) \lor \text{Kills}(Curiosity,Tuna)\)
  E) \(\text{Cat}(Tuna)\)
  F) \(\forall x (\text{Cat}(x) \Rightarrow \text{Animal}(x))\)

Query: \(\text{Kills}(Curiosity,Tuna)\)

- **A1) **\(\text{Dog}(D)\)
  A2) \(\text{Owns}(Jack,D)\)
  B) \(\text{Dog}(y) \land \text{Owns}(x,y) \Rightarrow \text{AnimalLover}(x)\)
  C) \(\forall x \text{AnimalLover}(x) \land \text{Animal}(y) \land \text{Kills}(x,y) \Rightarrow \text{False}\)
  D) \(\text{Kills}(Jack,Tuna) \lor \text{Kills}(Curiosity,Tuna)\)
  E) \(\text{Cat}(Tuna)\)
  F) \(\text{Cat}(x) \Rightarrow \text{Animal}(x)\)

Query: \(\text{Kills}(Curiosity,Tuna) \Rightarrow \text{False}\)
Resolution Proof

Answer Extraction

- If the query contains existentially quantified variables, these become universally quantified in the negation.
  \( \exists w \text{ Kills}(w, \text{Tuna}) \rightarrow \text{Kills}(w, \text{Tuna}) \Rightarrow \text{False} \)

- If you compose the substitutions from all unifications made in the course of a proof, you obtain an answer substitution that gives a binding for the query variables.

To find all answers, must find all distinct resolution proofs since each one may provide a different answer.

Resolution Strategies

- Need heuristics and strategies to decide what resolutions to make in order to control the search for a proof.

  - **Unit preference**: Prefer to make resolutions with single literals (facts, unit clauses) since this generates a shorter clause and the goal is to derive the empty clause.
    
    \[
    P + \neg P \lor Q_1 \lor \ldots \lor Q_n \Rightarrow Q_1 \lor \ldots \lor Q_n
    \]

  - **Set of Support**: Always resolve with a clause from the query or a clause previously generated from such a resolution. Directs search towards answering the query rather than deducing arbitrary consequences of the KB. Assuming the original KB is consistent, this strategy is complete.

  - **Input Resolution**: One of the resolving clauses should always be from the input (i.e. from the KB or the negated query). Complete for Horn clauses but not in general.

Resolution Strategies (cont)

- **Linear Resolution**: Generalization of input resolution. Allow resolutions of clauses \( P \) and \( Q \) if \( P \) is in the input or is an ancestor of \( Q \) in the proof tree.

- **Subsumption**: Clauses that are more specific than other clauses should be eliminated as redundant. Such clauses are said to be **subsumed**.

  \( P(x) \) subsumes \( P(A) \)
  \( P \) subsumes \( P \lor Q \)
  \( P(x,y) \) subsumes \( P(z,z) \lor Q(y) \)

Clause \( A \) **subsumes** clause \( B \) is there exists a substitution \( \theta \) such that the literals in \( \text{SUBST}(\theta,A) \) are a subset of the literals in \( B \).
**Gödel's Incompleteness Theorem**

- If FOPC is extended to allow for the use of mathematical induction for showing that statements are true for all natural numbers, there are true statements that can never be proven.

- The logical theory of numbers starts with a single constant 0, the function S (successor) for generating the natural numbers, and axioms defining functions for multiplication, addition, and exponentiation.

- Proof relies on producing a unique number for each sentence in the logic (Gödel number) and constructing a sentence whose number is \( n \) which states “Sentence number \( n \) is not provable.”

- If this sentence is provable from the axioms, then it is a false statement which is provable and therefore the axioms are inconsistent.

- If this sentence is not provable from the axioms, then it is a true statement which is not provable and inference is incomplete.

---

**Logicist Program**

- Encode general knowledge about the world and/or any given domain as a set of sentences in first-order logic.

- Use general logical inference to solve problems and answer questions.

- Focus on **epistemological problems** of what and how to represent knowledge rather than the **heuristic problems** of how to efficiently conduct search.

- Problems with the logicist program:
  - Knowledge representation problem
  - Knowledge acquisition problem
  - Intractable search problem