Inference in First-Order Logic

First-Order Deduction

- 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 |– B then A |= B
  - Completeness: If A |= B then A |– B
- Forward and backward chaining are sound and can be reasonably efficient but are incomplete.
- Resolution is sound and complete for FOPC but can be very inefficient.

Inference Rules for Quantifiers

- Let SUBST(θ, α) denote the result of applying a substitution or binding list θ to the sentence α.
  - SUBST({x/Tom, y,/Fred}, Uncle(x,y)) = Uncle(Tom, Fred)
- Inference rules
  - **Universal Elimination**: ∀*v* α |– SUBST({*v*/*g*},α)
    for any sentence, α, variable, *v*, and ground term, *g*

    ∀x Loves(x, FOPC) |– Loves(Ray, FOPC)
  - **Existential Elimination**: ∃*v* α |– SUBST({*v*/*k*},α)
    for any sentence, α, variable, *v*, and constant symbol, *k*, that doesn't occur elsewhere in the KB (**Skolem constant**)

    ∃x (Owns(Mary,x) ∧ Cat(x)) |– Owns(Mary,MarysCat) ∧ Cat(MarysCat)
  - **Existential Introduction**: α |- ∃v SUBST({*g*/*v*},α)
    for any sentence, α, variable, *v*, that does not occur in α, and ground term, *g*, that does occur in α

    Loves(Ray, FOPC) |– ∃x Loves(x, FOPC)

Sample Proof

1) ∀x,y(Parent(x,y) ∧ Male(x) ⇒ Father(x,y))
2) Parent(Tom,John)
3) Male(Tom)

Using Universal Elimination from 1)

4) ∀y(Parent(Tom,y) ∧ Male(Tom) ⇒ Father(Tom,y))

Using Universal Elimination from 4)

5) Parent(Tom,John) ∧ Male(Tom) ⇒Father(Tom,John)

Using And Introduction from 2) and 3)

6) Parent(Tom,John) ∧ Male(Tom)

Using Modes Ponens from 5) and 6)

7) Father(Tom,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:**
  $p_1', p_2', \ldots p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q) \vdash \text{SUBST}(\theta,q)$

  where θ is a substitution such that for all *i* $\text{SUBST}(\theta,p_i')=\text{SUBST}(\theta,p_i)$
- 1) $\forall x,y(\text{Parent}(x,y) \wedge \text{Male}(x) \Rightarrow \text{Father}(x,y))$
  2) Parent(Tom,John)
  3) Male(Tom)

  θ={x/Tom, y/John)

  4) Father(Tom,John)

Canonical Form

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

  $\forall v_1,v_2,\ldots v_n\ p_1 \wedge p_2 \wedge \ldots \wedge p_m \Rightarrow q$
- 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.

  $\forall v_1,v_2,\ldots v_n\ \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n \vee q$

  If θ is the constant False, this simplifies to

  $\forall v_1,v_2,\ldots v_n\ \neg p_1 \vee \neg p_2 \vee \ldots \vee \neg p_n$

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

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.

  UNIFY(p,q)=θ where SUBST(θ,p)=SUBST(θ,q)

  θ is called a **unifier**.
- Examples

  UNIFY(Parent(x,y), Parent(Tom, John)) = {x/Tom, y/John}

  UNIFY(Parent(Tom,x), Parent(Tom, John)) = {x/John}}

  UNIFY(Likes(x,y), Likes(z,FOPC)) = {x/z, y/FOPC}

  UNIFY(Likes(Tom,y), Likes(z,FOPC)) = {z/Tom, y/FOPC}

  UNIFY(Likes(Tom,y), Likes(y,FOPC)) = fail

  UNIFY(Likes(Tom,Tom), Likes(x,x)) = {x/Tom}

  UNIFY(Likes(Tom,Fred), Likes(x,x)) = fail

Unification (cont.)

- Exact variable names used in sentences in the KB should not matter.
- But if Likes(x,FOPC) is a formula in the KB, it does not unify with Likes(John,x) but does unify with Likes(John,y).
- To avoid such conflicts, one can **standardize apart** one of the arguments to UNIFY to make its variables unique by renaming them.

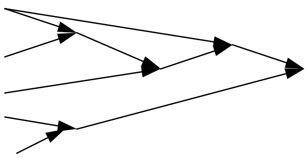
  Likes(x,FOPC) -> Likes(x_1, FOPC)
  UNIFY(Likes(John,x),Likes(x_1,FOPC)) = {x_1/John, x/FOPC}
- There are many possible unifiers for some atomic sentences.

  UNIFY(Likes(x,y),Likes(z,FOPC)) = {x/z, y/FOPC}
  {x/John, z/John, y/FOPC}
  {x/Fred, z/Fred, y/FOPC}
  ......

  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:

  SUBST(COMPOSE(θ1,θ2),p) = SUBST(θ2,SUBST(θ1,p))

---

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*, $[p_1, \ldots, p_{i-1}, p_{i+1}, \ldots, p_n], q, \theta$)
  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

Forward Chaining Example

Assume in KB
1) Parent(x,y) ∧ Male(x) ⇒ Father(x,y)
2) Father(x,y) ∧ Father(x,z) ⇒ 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}

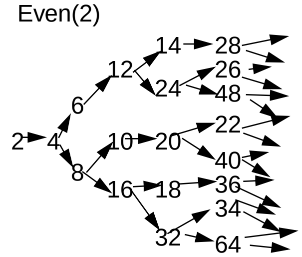
Rule 2) triggered again and adds:
10) Sibling(John, Fred) {x/Tom, y/John, z/Fred}

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) ⇒ Even(plus(x,2))
  Integer(x) ⇒ Even(times(2,x))
  Even(x) ⇒ 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.

  Sibling(x,John) ?
  Father(x,y) ?

  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 it's 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.

```
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 {θ}
   q ← FIRST(qlist)
      for each q'_i in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
         Add COMPOSE(θ, θ_i) to answers
      end
      for each sentence (p_1 ∧ ... ∧ p_n ⇒ q'_i) in KB such that θ_i ← UNIFY(q, q'_i) succeeds do
         answers ← BACK-CHAIN-LIST(KB, SUBST(θ_i, [p_1 ... p_n]), COMPOSE(θ, θ_i)) ∪ answers
      end
   return the union of BACK-CHAIN-LIST(KB, REST(qlist), θ) for each θ ∈ answers
```


Backchaining Examples

KB:
1) Parent(x,y) ∧ Male(x) ⇒ Father(x,y)
2) Father(x,y) ∧ Father(x,z) ⇒ Sibling(y,z)
3) Parent(Tom,John)
4) Male(Tom)
7) Parent(Tom,Fred)

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

Query: Father(Tom,s)
 Subgoal: Parent(Tom,s) ∧ Male(Tom)
  {s/John}
   Subgoal: Male(Tom)
 Answer: {s/John}
  {s/Fred}
   Subgoal: Male(Tom)
 Answer: {s/Fred}
Answers: ({s/John}, {s/Fred})

Backchaining Examples (cont)

Query: Father(f,s)
Subgoal: Parent(f,s) ∧ Male(f)
  {f/Tom, s/John}
   Subgoal: Male(Tom)
 Answer: {f/Tom, s/John}
  {f/Tom, s/Fred}
   Subgoal: Male(Tom)
 Answer: {f/Tom, s/Fred}
Answers: ({f/Tom,s/John}, {f/Tom,s/Fred})

Query: Sibling(a,b)
Subgoal: Father(f,a) ∧ Father(f,b)
  {f/Tom, a/John}
   Subgoal: Father(Tom,b)
    {b/John}
  Answer: {f/Tom, a/John, b/John}
    {b/Fred}
  Answer: {f/Tom, a/John, b/Fred}
  {f/Tom, a/Fred}
   Subgoal: Father(Tom,b)
    {b/John}
  Answer: {f/Tom, a/Fred, b/John}
    {b/Fred}
  Answer: {f/Tom, a/Fred, b/Fred}
Answers: ({f/Tom, a/John, b/John},{f/Tom, a/John, b/Fred}
 {f/Tom, a/Fred, b/John}, {f/Tom, a/Fred, b/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:

  $P(x) \Rightarrow Q(x)$
  $\neg P(x) \Rightarrow R(x)$ (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 thoerem 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 ¬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.

  $\{\alpha \vee \beta, \neg\beta \vee \gamma\} \vdash \alpha \vee \gamma$ OR $\{\neg\alpha \Rightarrow \beta, \beta \Rightarrow \gamma\} \vdash \neg\alpha \Rightarrow \gamma$

  Reasoning by cases OR transitivity of implication
- First-order form

  For two literals p_j and q_k in two clauses

  $p_1 \vee \ldots p_j \ldots \vee p_m$
  $q_1 \vee \ldots q_k \ldots \vee q_n$

  such that θ=UNIFY(p_j, $\neg q_k$), derive

  $\text{SUBST}(\theta, p_1 \vee \ldots p_{j-1} \vee p_{j+1} \ldots \vee p_m \vee q_1 \vee \ldots q_{k-1} \vee q_{k+1} \ldots \vee q_n)$
- Can also be viewed in implicational form where all negated literals are in a conjunctive antecedent and all positive literals in a disjunctive conclusion.

  $\neg p_1 \vee \ldots \vee \neg p_m \vee q_1 \vee \ldots \vee q_n \Leftrightarrow$

  $p_1 \wedge \ldots \wedge p_m \Rightarrow q_1 \vee \ldots \vee q_n$

Conjunctive Normal Form (CNF)

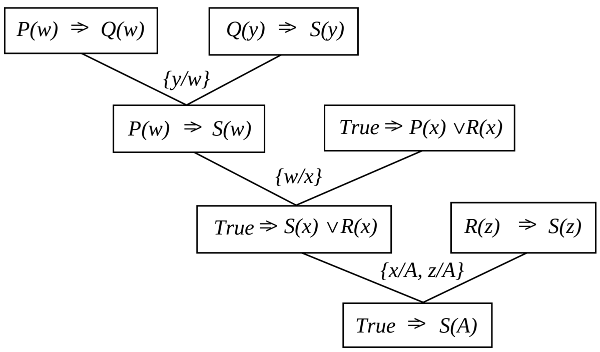
- For resolution to apply, all sentences must be in **conjunctive normal form**, a conjunction of disjunctions of literals

  $(a_1 \vee \ldots \vee a_m) \wedge$
  $(b_1 \vee \ldots \vee b_n) \wedge$
  $\ldots\ldots \wedge$
  $(x_1 \vee \ldots \vee x_v)$
- Representable by a set of clauses (disjunctions of literals)
- Also representable as a set of implications (INF).
- Example

Initial	CNF	INF
$P(x) \Rightarrow Q(x)$	$\neg P(x) \vee Q(x)$	$P(x) \Rightarrow Q(x)$
$\neg P(x) \Rightarrow R(x)$	$P(x) \vee R(x)$	$True \Rightarrow P(x) \vee R(x)$
$Q(x) \Rightarrow S(x)$	$\neg Q(x) \vee S(x)$	$Q(x) \Rightarrow S(x)$
$R(x) \Rightarrow S(x)$	$\neg R(x) \vee S(x)$	$R(x) \Rightarrow S(x)$

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.



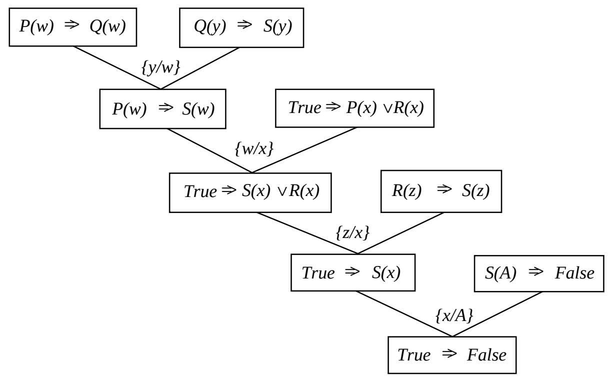
- **Factoring** removes redundant literals from clauses

S(A) ∨ S(A) –> S(A)

Refutation Proofs

- Unfortunately, resolution proofs in this form are still incomplete.
- For example, it cannot prove any tautology (e.g. P∨¬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 ∧ ¬P ⇒ False) ⇔ KB ⇒ P



Resolution Theorem Proving

- Convert sentences in the KB to CNF (clausal form)
- Take the negation of the poposed 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 ⇒ q –> ¬p ∨ q   p ⇔ q -> (¬p ∨ q) ∧ (p ∨ ¬q)

- **Move ¬ inward** to only be a part of literals by using deMorgan's laws and quantifier rules.
  - ¬(p ∨ q) –> ¬p ∧ ¬q
  - ¬(p ∧ q) –> ¬p ∨¬q
  - ¬∀x p –> ∃x ¬p
  - ¬∃x p –> ∀x ¬p
  - ¬¬p –> p
- **Standardize variables** to avoid use of the same variable name by two different quantifiers.

$\forall x\, P(x) \vee \exists x\, P(x)$ –> $\forall x_1\, P(x_1) \vee \exists x_2\, P(x_2)$

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

$\forall x_1\, P(x_1) \vee \exists x_2\, P(x_2)$ –> $\forall x_1 \exists x_2\, \mathbf{(}P(x_1) \vee P(x_2)\mathbf{)}$

Conversion to Clausal Form (cont)

- **Skolemize**: Remove existential quantifiers by replacing each existentially quantified variable with a **Skolem constant** or **Skolem function** as appropriate.
  - If an existential variable is not within the scope of any universally quantified variable, then replace every instance of the variable with the same unique constant that does not appear anywhere else.

    $\exists x\,(P(x) \wedge Q(x))$ –> $P(C_1) \wedge Q(C_1)$

  - If it is within the scope of *n* universally quantified variables, then replace it with a unique *n*-ary function over these universally quantified variables.

    $\forall x_1 \exists x_2\, \mathbf{(}P(x_1) \vee P(x_2)\mathbf{)}$ –> $\forall x_1 \mathbf{(}P(x_1) \vee P(f_1(x_1)))$

    $\forall x(\text{Person}(x) \Rightarrow \exists y(\text{Heart}(y) \wedge \text{Has}(x,y)))$ –>
    $\forall x(\text{Person}(x) \Rightarrow \text{Heart}(\text{HeartOf}(x)) \wedge \text{Has}(x,\text{HeartOf}(x)))$

  - Afterwards, all variables can be assumed to be universally quantified, so remove all quantifiers.

Conversion to Clausal Form (cont)

- **Distribute $\wedge$ over $\vee$** to convert to conjunctions of clauses

  $(a \wedge b) \vee c$ –> $(a \vee c) \wedge (b \vee c)$
  $(a \wedge b) \vee (c \wedge d)$ –> $(a \vee c) \wedge (b \vee c) \wedge (a \vee d) \wedge (b \vee d)$

  Can exponentially expand size of sentence.

- **Flatten nested conjunctions and disjunctions** to get final CNF

  $(a \vee b) \vee c$ –> $(a \vee b \vee c)$
  $(a \wedge b) \wedge c$ –> $(a \wedge b \wedge c)$

- **Convert clauses to implications** if desired for readability

  $(\neg a \vee \neg b \vee c \vee d)$ –> $a \wedge b \Rightarrow c \vee d$

Sample Clausal Conversion

$\forall x((\text{Prof}(x) \vee \text{Student}(x)) \Rightarrow (\exists y(\text{Class}(y) \wedge \text{Has}(x,y)) \wedge \exists y(\text{Book}(y) \wedge \text{Has}(x,y))))$

$\forall x(\neg(\text{Prof}(x) \vee \text{Student}(x)) \vee (\exists y(\text{Class}(y) \wedge \text{Has}(x,y)) \wedge \exists y(\text{Book}(y) \wedge \text{Has}(x,y))))$

$\forall x((\neg\text{Prof}(x) \wedge \neg\text{Student}(x)) \vee (\exists y(\text{Class}(y) \wedge \text{Has}(x,y)) \wedge \exists y(\text{Book}(y) \wedge \text{Has}(x,y))))$

$\forall x((\neg\text{Prof}(x) \wedge \neg\text{Student}(x)) \vee (\exists y(\text{Class}(y) \wedge \text{Has}(x,y)) \wedge \exists z(\text{Book}(z) \wedge \text{Has}(x,z))))$

$\forall x\exists y\exists z((\neg\text{Prof}(x)\wedge\neg\text{Student}(x))\vee ((\text{Class}(y) \wedge \text{Has}(x,y)) \wedge (\text{Book}(z) \wedge \text{Has}(x,z))))$

$(\neg\text{Prof}(x)\wedge\neg\text{Student}(x))\vee (\text{Class}(f(x)) \wedge \text{Has}(x,f(x)) \wedge \text{Book}(g(x)) \wedge \text{Has}(x,g(x))))$

$(\neg\text{Prof}(x) \vee \text{Class}(f(x))) \wedge$
$(\neg\text{Prof}(x) \vee \text{Has}(x,f(x))) \wedge$
$(\neg\text{Prof}(x) \vee \text{Book}(g(x))) \wedge$
$(\neg\text{Prof}(x) \vee \text{Has}(x,g(x))) \wedge$
$(\neg\text{Student}(x) \vee \text{Class}(f(x))) \wedge$
$(\neg\text{Student}(x) \vee \text{Has}(x,f(x))) \wedge$
$(\neg\text{Student}(x) \vee \text{Book}(g(x))) \wedge$
$(\neg\text{Student}(x) \vee \text{Has}(x,g(x))))$

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?

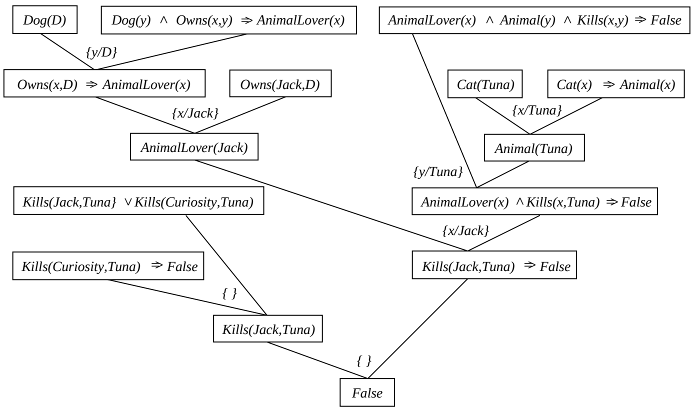
- A) $\exists x\ \text{Dog}(x) \wedge \text{Owns}(\text{Jack},x)$
  B) $\forall x\ (\exists y\ \text{Dog}(y) \wedge \text{Owns}(x,y)) \Rightarrow \text{AnimalLover}(x))$
  C) $\forall x\ \text{AnimalLover}(x) \Rightarrow (\forall y\ \text{Animal}(y) \Rightarrow \neg\text{Kills}(x,y))$
  D) $\text{Kills}(\text{Jack},\text{Tuna}) \vee \text{Kills}(\text{Cursiosity},\text{Tuna})$
  E) Cat(Tuna)
  F) $\forall x(\text{Cat}(x) \Rightarrow \text{Animal}(x))$

  Query: Kills(Curiosity,Tuna)

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

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

Resolution Proof



Answer Extraction

- If the query contains existentially quantified variables, these become universally quantified in the negation.

  ∃w Kills(w,Tuna) –> Kills(w,Tuna) ⇒ 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.

  $P \; + \; \neg P \vee Q_1 \vee ... \vee Q_n \quad \text{–>} \quad Q_1 \vee ... \vee Q_n$

- **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 ∨ Q
  P(x,y) subsumes P(z,z) ∨ Q(y)

  Clause A **subsumes** clause B is there exists a substitution θ such that the literals in SUBST(θ,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