Sound and Complete Inference Rules in FOL

An inference rule \( i \) is called sound if
\[
    KB \models \alpha \quad \text{whenever} \quad KB \vdash_i \alpha
\]
An inference rule \( i \) is called complete if
\[
    KB \vdash_i \alpha \quad \text{whenever} \quad KB \models \alpha
\]
Generalised Modus-Ponens (equivalently, forward or backward chaining) is sound and complete for Horn KBs but incomplete for general first-order logic.

Example

Let us consider the following formulas:
\[
    \begin{align*}
    \text{PhD}(x) & \Rightarrow \text{HighlyQualified}(x) \\
    \neg \text{PhD}(x) & \Rightarrow \text{EarlyEarnings}(x) \\
    \text{HighlyQualified}(x) & \Rightarrow \text{Rich}(x) \\
    \text{EarlyEarnings}(x) & \Rightarrow \text{Rich}(x)
    \end{align*}
\]
From the above we should be able to infer \( \text{Rich}(Me) \), but GMP won’t do it!
Is there a complete inference procedure for FOL?
The Resolution Inference Rule

Basic **propositional** version:
\[
\frac{\alpha \lor \beta, \neg \beta \lor \gamma}{\alpha \lor \gamma} \quad \text{or equivalently} \quad \frac{-\alpha \Rightarrow \beta, \beta \Rightarrow \gamma}{-\alpha \Rightarrow \gamma}
\]

The Resolution Inference Rule - FOL version

\[
p_1 \lor \cdots \lor p_j \cdots \lor p_m, \quad q_1 \lor \cdots \lor q_k \cdots \lor q_n
\]
\[
\text{SUBST}(\sigma, (p_1 \lor \cdots \lor p_{j-1} \lor p_{j+1} \lor \cdots \lor p_m \lor q_1 \cdots \lor q_{k-1} \lor q_{k+1} \cdots \lor q_n))
\]
\[
\text{where } UNIFY(p_j, -q_k) = \sigma.
\]

**Note:** \(\sigma\) is the **most general unifier** (MGU) of \(p_j\) and \(q_k\). The literals \(p_j\) and \(q_k\) are called **complementary** literals because each one unifies with the negation of the other. The resulting disjunction is called a **resolvent**.
Examples

\[\neg \text{Rich}(x) \lor \text{Unhappy}(x), \quad \text{Rich}(\text{Me})\]
\[
\begin{align*}
\text{Unhappy}(\text{Me})
\end{align*}
\]

with MGU \(\sigma = \{x/\text{Me}\}\)

Examples (cont’d)

\(\text{PhD}(x) \Rightarrow \text{HighlyQualified}(x)\)
\[\neg \text{PhD}(x) \Rightarrow \text{EarlyEarnings}(x)\]
\[\text{HighlyQualified}(x) \Rightarrow \text{Rich}(x)\]
\[\text{EarlyEarnings}(x) \Rightarrow \text{Rich}(x)\]

Let us try resolution to infer \(\text{Rich}(\text{Me})\)!

The standard way of showing that \(KB \vdash \phi\) by resolution is to add \(\neg \phi\) to the \(KB\) and show that we can reach the empty clause by repeated application of the resolution rule.

In our case, we add \(\neg \text{Rich}(\text{Me})\).
Examples (cont’d)

Let us first write all our formulas as disjunctions:

\[ \neg PhD(x) \lor \text{HighlyQualified}(x) \]

\[ PhD(x) \lor \text{EarlyEarnings}(x) \]

\[ \neg \text{HighlyQualified}(x) \lor \text{Rich}(x) \]

\[ \neg \text{EarlyEarnings}(x) \lor \text{Rich}(x) \]

\[ \neg \text{Rich}(Me) \]

Now we can apply resolution repeatedly.

Examples (cont’d)

From

\[ \neg \text{Rich}(Me) \]

and

\[ \neg \text{HighlyQualified}(z) \lor \text{Rich}(z) \]

with MGU \( \sigma = \{z/Me\} \), we infer

\[ \neg \text{HighlyQualified}(Me). \]
Examples (cont’d)

From
\[ \neg \text{Rich}(Me) \]
and
\[ \neg \text{EarlyEarnings}(w) \lor \text{Rich}(w) \]
using MGU \( \sigma = \{w/Me\} \), we infer
\[ \neg \text{EarlyEarnings}(Me). \]

Examples (cont’d)

From
\[ \neg \text{PhD}(x) \lor \text{HighlyQualified}(x) \]
and
\[ \text{PhD}(y) \lor \text{EarlyEarnings}(y) \]
with MGU \( \sigma = \{x/y\} \), we infer
\[ \text{HighlyQualified}(y) \lor \text{EarlyEarnings}(y). \]
Examples (cont’d)

From

\[ \text{HighlyQualified}(v) \lor \text{EarlyEarnings}(v) \]

and

\[ \neg \text{EarlyEarnings}(Me) \]

using MGU \( \sigma = \{ v/Me \} \), we infer

\[ \text{HighlyQualified}(Me). \]

Examples (cont’d)

From

\[ \text{HighlyQualified}(Me) \]

and

\[ \neg \text{HighlyQualified}(Me) \]

using MGU \( \sigma = \{ \} \), we infer the empty clause. Thus we have reached a contradiction!
**Conjunctive Normal Form**

To be able to do resolution, the given formulas have to be in conjunctive normal form.

**Definition.** A literal is an atomic formula or the negation of an atomic formula. An atomic formula is also called a positive literal, and the negation of an atomic formula is called a negative literal. A clause is a disjunction of literals. There is a special clause called empty which is equivalent to false.

**Definition.** A FOL formula is in conjunctive normal form (CNF) if it is a conjunction of disjunctions of literals (equivalently if it is a set of clauses).

**Proposition.** Every FOL formula is equivalent to a formula in CNF.

---

**Conversion to CNF**

1. **Eliminate equivalences and implications** using the laws:
   
   \[(\phi \leftrightarrow \psi) \equiv (\phi \Rightarrow \psi \land \psi \Rightarrow \phi)\]
   
   \[\phi \Rightarrow \psi \equiv \neg \phi \lor \psi\]

2. **Move \(\neg\) inwards** using the equivalences
   
   \[\neg(\phi \lor \psi) \equiv \neg \phi \land \neg \psi\]
   
   \[\neg(\phi \land \psi) \equiv \neg \phi \lor \neg \psi\]
   
   \[\neg(\forall x) \phi \equiv (\exists x) \neg \phi\]
   
   \[\neg(\exists x) \phi \equiv (\forall x) \neg \phi\]
   
   \[\neg \neg \phi \equiv \phi\]
Conversion to CNF (cont’d)

3. Rename variables so that each quantifier has a unique variable.

3. Eliminate existential quantifiers.
   If an existential quantifier does not occur in the scope of a universal quantifier, we simply drop the quantifier and replace all occurrences of the quantifier variable by a new constant called a Skolem constant.
   If an existential quantifier $\exists x$ is within the scope of universal quantifiers $\forall y_1, \ldots, \forall y_n$, we drop the quantifier and replace all occurrences of the quantifier variable $x$ by the term $f(y_1, \ldots, y_n)$ where $f$ is a new function symbol called a Skolem function.

Conversion to CNF (cont’d)

5. Drop all universal quantifiers.

6. Distribute $\land$ over $\lor$ using the equivalence
   $$(\phi \land \psi) \lor \theta \equiv (\phi \lor \theta) \land (\psi \lor \theta)$$

7. Flatten nested conjunctions or disjunctions. Then, write each disjunction on a separate line and standardize variables apart (i.e., make sure disjunctions use different variables).
Example

Let us convert to CNF the following sentence:

$$(\forall x)((\forall y)P(x,y) \Rightarrow (\forall y)(Q(x,y) \Rightarrow R(x,y)))$$

1. Eliminate implications:

$$(\forall x)((\neg(\forall y)P(x,y) \lor (\forall y)((\neg Q(x,y) \lor R(x,y))))$$

2. Move $\neg$ inwards:

$$(\forall x)((\exists y)(\neg P(x,y) \lor (\exists y)(Q(x,y) \land \neg R(x,y))))$$

Example (cont’d)

3. Rename variables:

$$(\forall x)((\exists y)(\neg P(x,y) \lor (\exists z)(Q(x,z) \land \neg R(x,z)))$$

4. Skolemize:

$$(\forall x)(\neg P(x,F_1(x)) \lor (Q(x,F_2(x)) \land \neg R(x,F_2(x))))$$

5. Drop universal quantifiers:

$$\neg P(x,F_1(x)) \lor (Q(x,F_2(x)) \land \neg R(x,F_2(x)))$$

6. Distribute $\land$ over $\lor$:

$$((\neg P(x,F_1(x)) \lor Q(x,F_2(x))) \land (\neg P(x,F_1(x)) \lor \neg R(x,F_2(x))))$$

7. Final form:

$$\neg P(x,F_1(x)) \lor Q(x,F_2(x))$$

$$\neg P(x,F_1(x)) \lor \neg R(x,F_2(x))$$
Resolution: Soundness and Refutation-Completeness

**Theorem.** (Soundness)
Let $KB$ be a knowledge base. If $\phi$ can be proved from $KB$ using resolution then $KB \models \phi$.

**Theorem.** (Refutation-completeness)
If a set $\Delta$ of clauses is unsatisfiable then resolution will derive the empty clause from $\Delta$.

**Note:** The above theorem holds only if $\Delta$ does not involve equality.

**Methodology:** If we are asked to prove $KB \models \alpha$ then we negate $\alpha$ and show that $KB \land \neg \alpha$ is unsatisfiable using resolution.

---

**Example 1**

The crime example we saw in a previous lecture:

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is an American.

Use resolution to conclude that West is a criminal.
Example 1: Formalization in FOL

- “... it is a crime for an American to sell weapons to hostile nations”:
  \[(∀x, y, z) \ (\text{American}(x) ∧ \text{Weapon}(y) ∧ \text{Nation}(z) ∧ \text{Hostile}(z) ∧ \text{Sells}(x, y, z) \Rightarrow \text{Criminal}(x))\]

- “Nono ... has some missiles”:
  \[(∃x) \ (\text{Owns}(\text{Nono}, x) ∧ \text{Missile}(x))\]

- “All of its missiles were sold to it by Colonel West”:
  \[(∀x) \ (\text{Owns}(\text{Nono}, x) ∧ \text{Missile}(x) \Rightarrow \text{Sells}(\text{West}, x, \text{Nono}))\]

Example 1: Formalization in FOL (cont’d)

- Missiles are weapons:
  \[(∀x) \ (\text{Missile}(x) \Rightarrow \text{Weapon}(x))\]

- An enemy of America is a “hostile nation”:
  \[(∀x) \ (\text{Enemy}(x, \text{America}) \Rightarrow \text{Hostile}(x))\]

- “West, who is an American”: \text{American}(\text{West})

- “The country Nono ...”: \text{Nation}(\text{Nono})

- “Nono, an enemy of America ...”: \text{Enemy}(\text{Nono, America}), \text{Nation}(\text{America})
Example 1: CNF form

- “... it is a crime for an American to sell weapons to hostile nations”:
  \[
  \neg \text{American}(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x, y, z) \lor \\
  \neg \text{Hostile}(z) \lor \text{Criminal}(x)
  \]

- “Nono ... has some missiles”:
  \[
  \text{Owns}(\text{Nono}, M1), \quad \text{Missile}(M1)
  \]

- “All of its missiles were sold to it by Colonel West”:
  \[
  \neg \text{Missile}(x) \lor \neg \text{Owns}(\text{Nono}, x) \lor \text{Sells}(\text{West}, x, \text{Nono})
  \]

Example 1: CNF form (cont’d)

- Missiles are weapons:
  \[
  \neg \text{Missile}(x) \lor \text{Weapon}(x)
  \]

- An enemy of America is a “hostile nation”:
  \[
  \neg \text{Enemy}(x, \text{America}) \lor \text{Hostile}(x)
  \]

- “West, who is an American”:
  \[
  \text{American}(\text{West})
  \]

- “The country Nono ...”:
  \[
  \text{Nation}(\text{Nono})
  \]

- “Nono, an enemy of America ...”:
  \[
  \text{Enemy}(\text{Nono}, \text{America}), \quad \text{Nation}(\text{America})
  \]
Example 1: Proof

\[(x) \lor \neg \text{Weapon}(y) \lor \neg \text{Sells}(x,y,z) \lor \neg \text{Hostile}(z) \lor \neg \text{Criminal}(x) \]
\[\neg \text{Criminal}(\text{West})\]
\[\neg \text{American}(\text{West}) \lor \neg \text{Weapon}(x) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\neg \text{American}(\text{West}) \lor \neg \text{Weapon}(x) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{American}(\text{West}) \lor \text{American}(\text{West}) \lor \text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]
\[\text{Missile}(\text{M1}) \lor \text{Missile}(\text{M1}) \lor \neg \text{Sells}(\text{West},y,z) \lor \neg \text{Hostile}(z) \]

Example 2

Let us assume that we know the following:

- Everyone who loves all animals is loved by someone.
- Anyone who kills an animal is loved by no one.
- Jack loves all animals.
- Either Jack or Curiosity killed the cat, who is named Tuna.

From the above facts, can we prove that Curiosity killed Tuna?
Example 2: Formalization in FOL

- Everyone who loves animals is loved by someone.
  \((\forall x)((\forall y)(\text{Animal}(y) \Rightarrow \text{Loves}(x, y)) \Rightarrow (\exists y)\text{Loves}(y, x))\)

- Anyone who kills an animal is loved by no one.
  \((\forall x)((\exists y)(\text{Animal}(y) \land \text{Kills}(x, y)) \Rightarrow (\forall z)\neg\text{Loves}(z, x))\)

- Jack loves all animals.
  \((\forall x)(\text{Animal}(x) \Rightarrow \text{Loves}(Jack, x))\)

- Either Jack or Curiosity killed the cat ...
  \(\text{Kills}(Jack, Tuna) \lor \text{Kills}(Curiosity, Tuna)\)

- ... who is named Tuna.
  \(\text{Cat}(Tuna)\)

We will also need the formula
\((\forall x)(\text{Cat}(x) \Rightarrow \text{Animal}(x))\)
which is background knowledge.

The negation of the formula to be proved is:
\(\neg\text{Kills}(Curiosity, Tuna)\)
Questions:

- How do we use resolution to prove that the following formula is valid?
  \[ \text{Happy}(John) \lor \neg \text{Happy}(John) \]

- How do we use resolution to prove that the following formula is unsatisfiable?
  \[ \text{Happy}(John) \land \neg \text{Happy}(John) \]
Fill-in-the-Blank Questions

So far we have used resolution to see that something follows from a KB. We can also use resolution to answer questions about facts that follow from a KB. In the previous example, we can use resolution to find the answer to the question: Who killed Tuna? This can be expressed using a free variable and writing the fill-in-the-blank query \( \text{Kills}(x, \text{Tuna}) \).

**Definition.** An answer literal for a fill-in-the-blank query \( \phi \) is an atomic formula of the form \( \text{Ans}(v_1, \ldots, v_n) \) where the variables \( v_1, \ldots, v_n \) are the free variables in \( \phi \).

Fill-in-the-Blank Questions (cont’d)

To answer the fill-in-the-blank query \( \phi \) we form the disjunction

\[
\text{Ans}(v_1, \ldots, v_n) \lor \neg \phi
\]

and convert it to CNF.

Then we use resolution and terminate our search when we reach a clause containing only answer literals (instead of terminating when we reach the empty clause).
Fill-in-the-Blank Questions (cont’d)

For fill-in-the-blank questions, we can have:

- **Termination with a clause which is a single answer literal** $Ans(c_1, \ldots, c_n)$. In this case, the constants $c_1, \ldots, c_n$ gives us an answer to the query. There might be more answers depending on whether there are more resolution refutations of $Ans(v_1, \ldots, v_n) \lor \neg \phi$. We can go on looking for more answers but we can never be sure that we have found them all (incompleteness of resolution).

- **Termination with a clause which is a disjunction of more than one answer literals.** In this case, one of the answer literals contains the answer but we cannot say which one for sure.

Example 1

KB:

Father(Art, John)

Father(Bob, Kim)

$(\forall x)(\forall y)(Father(x, y) \Rightarrow Parent(x, y))$

Query: Who is John’s parent?

To answer the query, we use resolution on the following set of clauses:

Father(Art, John)

Father(Bob, Kim)

$\neg Father(x, y) \lor Parent(x, y)$

Ans(z) $\lor \neg Parent(z, John)$
Example 1 (cont’d)

From

Father(Art, John)

and

\neg Father(x, y) \lor Parent(x, y)

with MGU \{x/Art, y/John\}, we have

Parent(Art, John)

which in turn resolves with

Ans(z) \lor \neg Parent(z, John)

with MGU \{z/Art\}, to give

Ans(Art).

Example 2

KB:

Father(Art, John) \lor Father(Bob, John)

(\forall x)(\forall y)(Father(x, y) \Rightarrow Parent(x, y))

Query: Who is John’s parent?

To answer the query, we use resolution on the following set of clauses:

Father(Art, John) \lor Father(Bob, John)

\neg Father(x, y) \lor Parent(x, y)

Ans(z) \lor \neg Parent(z, John)
Example 2 (cont’d)

From

\[ Father(\text{Art}, \text{John}) \lor Father(\text{Bob}, \text{John}) \]

and

\[ \neg Father(x, y) \lor Parent(x, y) \]

with MGU \( \{x/\text{Art}, y/\text{John}\} \), we have

\[ Parent(\text{Art}, \text{John}) \lor Father(\text{Bob}, \text{John}) \]

which in turn resolves with

\[ \neg Father(x, y) \lor Parent(x, y) \]

with MGU \( \{x/\text{Bob}, y/\text{John}\} \), to give

\[ Parent(\text{Art}, \text{John}) \lor Parent(\text{Bob}, \text{John}). \]

---

Example 2 (cont’d)

From

\[ Parent(\text{Art}, \text{John}) \lor Parent(\text{Bob}, \text{John}) \]

and

\[ \text{Ans}(z) \lor \neg Parent(z, \text{John}) \]

with MGU \( \{z/\text{Art}\} \), we have

\[ \text{Ans}(\text{Art}) \lor Parent(\text{Bob}, \text{John}) \]

which in turn resolves with

\[ \text{Ans}(z) \lor \neg Parent(z, \text{John}) \]

with MGU \( \{z/\text{Bob}\} \), to give

\[ \text{Ans}(\text{Art}) \lor \text{Ans}(\text{Bob}). \]
Dealing with Equality

If we want to use equality in our resolution proofs, we can do it in two ways:

- Add appropriate formulas that axiomatize equality in our KB. What are these formulas?
- Use special inference rules that take equality into account.
- Use equational unification (a special kind of unification that takes equality into account).

The same is true for other special predicates such as arithmetic ones <, ≤ etc.

Axioms for Equality

- Reflective: $(\forall x) x = x$
- Symmetric: $(\forall x)(\forall y)(x = y \Rightarrow y = x)$
- Transitive: $(\forall x)(\forall y)(\forall z)(x = y \land y = z \Rightarrow x = z)$
- Substitution of equals:
  
  $(\forall x)(\forall y)(x = y \Rightarrow (P_1(x) \Leftrightarrow P_1(y)))$
  
  $(\forall x)(\forall y)(x = y \Rightarrow (P_2(x) \Leftrightarrow P_2(y)))$
  
  $\cdots$
  
  $(\forall w)(\forall x)(\forall y)(\forall z)(w = y \land x = z \Rightarrow (F_1(w, x) = F_1(y, z)))$
  
  $(\forall w)(\forall x)(\forall y)(\forall z)(w = y \land x = z \Rightarrow (F_2(w, x) = F_2(y, z)))$
  
  $\cdots$
Inference Rules for Equality: Demodulation

For any terms \(x, y\) and \(z\) where \(UNIFY(x, z) = \theta\) and \(m_n[z]\) is a literal containing the term \(z\):

\[
\frac{x = y, \ m_1 \lor \cdots \lor m_n[z]}{m_1 \lor \cdots \lor m_n[\text{SUBST}(\theta, y)]}
\]

Examples:

- From \(\text{TheThief} = \text{John}\) and \(\text{Arrested(TheThief)}\) with MGU \(\{\}\), we can conclude \(\text{Arrested(John)}\).
- From \(S(S(0)) = 2\) and \(P(2) \lor Q(S(S(w)))\) with MGU \(\{w/0\}\), we can conclude \(P(2) \lor Q(2)\).

Inference Rules for Equality: Paramodulation

For any terms \(x, y\) and \(z\) where \(UNIFY(x, z) = \theta\) and \(m_n[z]\) is a literal containing the term \(z\):

\[
\frac{l_1 \lor \cdots l_k \lor x = y, \ m_1 \lor \cdots \lor m_n[z]}{\text{SUBST}(\theta, l_1 \lor \cdots \lor l_k \lor m_1 \lor \cdots \lor m_n[y])}
\]

Example: From \(P(v) \lor F(A, v) = F(B, v)\) and \(Q(B) \lor R(F(A, C))\) with MGU \(\{v/C\}\), we can conclude \(P(C) \lor Q(B) \lor R(F(B, C))\).

Paramodulation is more general than demodulation and results in a refutation complete inference procedure for FOL with equality.
Resolution and Prolog

- Prolog is based on a specific form of resolution called
  **SLD-resolution** (Linear resolution with a Selection function for Definite clauses).
- The Prolog interpreter does resolution at each step of producing a new set of goals from a given set of goals and a clause in the Prolog program.

Horn Clauses

Let us revisit the definition of Horn clauses.

**Definition.** A Horn clause is a disjunction of literals of which at most one is positive.

In other words, a Horn clause is a formula in one of the following three forms:

\[ q \]
\[ \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \lor q \quad (\text{or} \quad p_1 \land p_2 \land \ldots \land p_n \Rightarrow q) \]
\[ \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \]

where \( p_1, \ldots, p_n, q \) are atomic formulas.
Horn Clauses in Prolog

Horn clauses of the first kind are called **facts**.
Horn clauses of the second kind are called **rules**.
If a Horn clause has exactly one positive literal, it is called a **definite clause**.

Horn Clauses in Prolog (cont’d)

Facts and rules (i.e., definite clauses) are of the form

\[ q \]

and

\[ \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \lor q \quad \text{(or)} \quad p_1 \land p_2 \land \ldots \land p_n \Rightarrow q \]

respectively. They are used to write **Prolog programs** using Prolog’s notation as follows:

\[ q \]

\[ q : - p_1, p_2, \ldots, p_n. \]
Horn Clauses in Prolog (cont’d)

Horn clauses of the form

\[ \neg p_1 \lor \neg p_2 \lor \ldots \lor \neg p_n \]

i.e., consisting of only negative literals can be used as **queries** in Prolog using Prolog’s notation as follows:

\[ ?- p_1, p_2, \ldots, p_n. \]

---

**Example**

```
predecessor(X, Z) :- %pr1
    parent(X, Z).
predecessor(X, Z) :- %pr2
    parent(X, Y),
    predecessor(Y, Z).

parent(pam, bob).  parent(bob, ann).
parent(tom, bob).   parent(bob, pat).
parent(tom, liz).   parent(pat, jim).
```
The Prolog interpreter resolves a query with a fact or rule at each step of its operation.

In the previous slide, we want to prove the goal

? − predecessor(pam, bob).

When Prolog uses the rule pr1 to arrive at the new goal

? − parent(pam, bob),

this is equivalent to performing resolution of

¬predecessor(pam, bob)

with Horn clause

¬parent(X, Y) ∨ predecessor(X, Y)

to produce

¬parent(pam, bob).
In the next step, the new goal

\(? – parent(pam, bob)\)

is unified with the fact

\(parent(pam, bob)\)

in the database and Prolog answers yes to the original query. This is equivalent to performing resolution of

\(–parent(pam, bob)\)

with

\(parent(pam, bob)\)

to arrive at the empty clause.

Thus we see that \(predecessor(pam, bob)\) logically follows from the Prolog program (viewed as a KB in FOL) thus Prolog correctly gives the answer yes.

**Exercise:** Explain the proof trees on the following slides using resolution.
Proof Tree for predecessor(pam, ann)

predecessor(pam, ann)

by rule pr1

parent(pam, ann)

no

by rule pr2

MGU{X/pam, Z/ann}

parent(pam, Y),
predecessor(Y, ann)

Proof Tree for predecessor(pam, ann) (cont’d)

predecessor(pam, ann)

by rule pr1

parent(pam, ann)

no

by rule pr2

MGU{X/pam, Z/ann}

parent(pam, Y),
predecessor(Y, ann)

by fact parent(pam, bob)

MGU{X/bob}

predecessor(bob, ann)

by rule pr1

MGU{X/bob, Z/ann}

parent(bob, ann)

yes
Computational Complexity and Resolution

Resolution proofs can in general be exponentially long as the following theorem demonstrates.

**Theorem (Haken, 1985).** There is a sequence of PL formulas \( p_1, p_2, p_3, \ldots \), each a tautology, such that the number of symbols of \( \neg p_n \) when converted to CNF is \( O(n^3) \), but the shortest resolution refutation of it contains at least \( c^n \) symbols (for a fixed \( c > 1 \)).

There are various strategies that can be applied to make resolution more efficient in practice (*unit preference, set of support, input resolution, subsumption*).

Other Normal Forms: DNF

**Definition.** A FOL formula is in **disjunctive normal form** (DNF) if it is a disjunction of conjunctions of literals.

**Proposition.** Every FOL formula is equivalent to a formula in DNF.
Other Normal Forms: PNF

**Definition.** A FOL formula is in **prenex normal form (PNF)** if all its quantifiers appear at the front of the formula.

**Proposition.** Every FOL formula is equivalent to a formula in PNF.

---

**Conversion to Prenex Normal Form**

- Steps 1 and 2 of conversion to CNF.
- **Move quantifiers to the front of the formula** using the equivalences
  
  \[
  (\forall x)(\phi \land \psi) \equiv (\forall x)\phi \land \psi \\
  (\forall x)(\phi \lor \psi) \equiv (\forall x)\phi \lor \psi \\
  (\exists x)(\phi \land \psi) \equiv (\exists x)\phi \land \psi \\
  (\exists x)(\phi \lor \psi) \equiv (\exists x)\phi \lor \psi
  \]

  The above equivalences hold only if \(x\) does not appear free in \(\psi\).

  Step 1 and 2 are not necessary if we introduce equivalences for the rest of the connectives.
A Brief History of Reasoning

- **450 b.c.** Stoic and Megarian PL, truth tables, assertion (modus ponens)
- **322 b.c.** Aristotle “syllogisms” (inference rules), quantifiers
- **1847** Boole PL (again)
- **1879** Frege FOL
- **1921/22** Post/Wittgenstein proof by truth tables
- **1930** Gödel ∃ complete algorithm for proofs in FOL
- **1930** Herbrand complete algorithm for proofs in FOL (reduce to propositional)
- **1931** Gödel ¬∃ complete algorithm for arithmetic proofs
- **1960** Davis/Putnam “practical” algorithm for PL resolution
- **1965** Robinson resolution

Soundness and Completeness of FOL Inference

**Theorem.** (Gödel, 1930)

\[ KB \models \phi \iff KB \vdash \phi. \]

**Theorem.** Checking entailment (equivalently: validity or unsatisfiability or provability) of a FOL formula is a **recursively enumerable** problem.
Informal Definitions

A yes/no problem $P$ is called **recursive** or **decidable** if there is an algorithm that, given input $x$, outputs “yes” and terminates whenever $x \in P$, and “no” and terminates when $x \notin P$.

A yes/no problem $P$ is called **recursively enumerable** or **semi-decidable** if there is an algorithm that, given input $x$, outputs “yes” and terminates whenever $x \in P$ but computes for ever when $x \notin P$.

The above algorithm is not a very useful because, if it has not terminated, we cannot know for sure whether we have waited long enough to get an answer.

---

Gödel’s Incompleteness Theorem

**Theorem.** (Gödel, 1930)
For any set $A$ of true sentences of number theory, and, in particular, any set of basic axioms, **there are other true sentences of number arithmetic that cannot be proved from $A$**.

**Sad conclusion:** We can never prove all the theorems of mathematics **within any given system of axioms**.
Soundness and Completeness (cont’d)

Theorem. (Herbrand, 1930)
If a finite set $\Delta$ of clauses is unsatisfiable then the Herbrand base of $\Delta$ is unsatisfiable.

Theorem. (Robinson, 1965)
**Soundness of Resolution.** If there is a resolution refutation of a clause $\phi$ from a set of clauses $KB$ then $KB \models \phi$.

Theorem. (Robinson, 1965)
**Completeness of Resolution.** If a set of clauses $KB$ is unsatisfiable then there is a resolution refutation of the empty clause from $KB$.

Question: How can we use a refutation-complete proof procedure (e.g., resolution) to determine whether a sentence $\phi$ is entailed by a set of sentences $KB$?

Answer: We can negate $\phi$, add it to $KB$ and then use resolution. But we will not know whether $KB \models \phi$ until resolution finds a contradiction and returns.

While resolution has not returned, we do not know whether the system has gone into a loop or the proof is about to pop out!!!
Some Good News

There are many interesting subsets of FOL that are decidable:
- Monadic logic (only unary predicates).
- Horn clauses
- ...

Many practical problems can be encoded in these subsets!

Knowledge-Based Agents

function KB-AGENT(percept) returns an action

static KB, a knowledge-base

    t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1

return action

Using the FOL machinery we presented, how can we implement knowledge-based agents?
Logical Reasoning Systems

- **Logic programming** languages (most notably Prolog).
  Prolog was developed in 1972 by Alain Colmerauer and it is based on the idea of backward chaining. Prolog’s motto (after Kowalski) is:
  
  \[ \text{Algorithm} = \text{Logic} + \text{Control} \]
  
  Logic programming and Prolog was the basis of much exciting research and development in logic programming in the 70’s and 80’s.
  
  Logic programming and its extensions is still a very lively area of research that has been applied in many areas (databases, natural language processing, expert systems etc.). Of particular, importance is constraint logic programming (CLP) that integrates logic programming with CSPs. CLP has been used with success recently in many combinatorial optimisation applications (e.g., scheduling, planning, etc.)

Logical Reasoning Systems (cont’d)

- **Production systems** based on the idea of forward-chaining (where the conclusion of an implication is interpreted as an action to be executed).
  
  Production systems were used a lot in early AI work (particularly in rule-based expert systems).
  
  There are various implemented production systems such as OPS-5 or CLIPS.
Logical Reasoning Systems (cont’d)

• **Theorem provers** are more powerful tools than Prolog since they can
deal with full first-order logic.
Examples: OTTER, PTTP, etc.
Theorem provers have come up with novel mathematical results (lattice
theory, a formal proof of Godel’s incompleteness theorem, Robbins
algebra).
They are also used in verification and synthesis of both hardware and
software because both domains can be given correct axiomatizations.

Readings

• AIMA, Chapter 9.
• M. Genesereth and N. Nilsson. “Logical Foundations of
Artificial Intelligence”, Chapter 4.
This chapter gives a more formal treatment with detailed
proofs of the theorems we presented.
• The book “Mechanizing Proof: Computing, Risk and Trust” by
Donald MacKenzie (MIT Press, 2001) is an interesting
historical and sociological account of automated theorem
proving. Read this book for fun when you have time!