Elements of First-Order Predicate Logic

First Order Language:

- An alphabet consists of the following classes of symbols:
  1. variables denoted by \( X, Y, Z, Boo, \ldots \) (infinite)
  2. constants denoted by \( 1, a, boo, john, \ldots \)
  3. functors denoted by \( f, g, +, -, \ldots \)
  4. predicate symbols denoted by \( p, q, dog, \ldots \)
  5. connectives, which are: \( \neg \) (negation), \( \lor \) (disjunction), \( \land \) (conjunction), \( \rightarrow \) (implication) and \( \iff \) (equivalence),
  6. quantifiers, which are: \( \exists \) (there exists) and \( \forall \) (for all),
  7. parentheses, which are: ( and ) and the comma, that is: ,.

- Each functor and predicate symbol has a fixed \textit{arity}, they are often represented in \textit{Functor/Arity} form, e.g. f/3.
- A constant can be seen as a functor of arity 0.
- Propositions are represented by a predicate symbol of arity 0.
Important: Notation Convention Used

(A bit different from standard notational conventions in logic, but good for compatibility with LP systems)

- Variables: start with a capital letter or a “” (X, Y, a, 1)
- Atoms, functors, predicate symbols: start with a lower case letter or are enclosed in ’ ‘ (f, g, a, 1, x, y, z, ’X’, ’1’)

Terms and Atoms

We define by induction two classes of strings of symbols over a given alphabet.

- The class of terms:
  - a variable is a term,
  - a constant is a term,
  - if f is an n-ary functor and t_1, ..., t_n are terms then f(t_1, ..., t_n) is a term.

- The class of atoms (different from LP!):
  - a proposition is an atom,
  - if p is an n-ary pred. symbol and t_1, ..., t_n are terms then p(t_1, ..., t_n) is an atom,
  - true and false are atoms.

- The class of Well Formed Formulas (WFFs):
  - an atom is a WFF,
  - if F and G are WFFs then so are ¬F, (F ∨ G), (F ∧ G), (F → G) and (F ↔ G),
  - if F is a WFF and X is a variable then ∃X F and ∀X F are WFF.

- Literal: positive or negative (non-negated or negated) atom.
Examples

Examples of Terms

- Given:
  - constants: a, b, c, 1, spot, john...
  - functors: f/1, g/3, h/2, +/3...
  - variables: X, L, Y...
- Correct: spot, f(john), f(X), +(1,2,3), +(X,Y,L), f(f(spot)), h(f(h(1,2)),L)
- Incorrect: spot(X), +(1,2), g, f(f(h))

Examples of Literals

- Given the elements above and:
  - predicate symbols: dog/1, p/2, q/0, r/0, barks/1...
- Correct: q, r, dog(spot), p(X,f(john))...
- Incorrect: q(X), barks(f), dog(barks(X))

Examples (Contd.)

Examples of WFFs

- Given the elements above
- Correct: q, q → r, r ← q, dog(X) ← barks(X), dog(X), p(X,Y), ∃ X (dog(X) ∧ barks(X) ∧ ¬ q), ∃ Y (dog(Y) → bark(Y))
- Incorrect: q ∨, ∃ p
More about WFFs

- Allow us to represent knowledge and reason about it
  - Marcus was a man \( \text{man}(\text{marcus}) \)
  - Marcus was a pompeian \( \text{pompeian}(\text{marcus}) \)
  - All pompeians were romans \( \forall X \text{ pompeian}(X) \rightarrow \text{roman}(X) \)
  - Caesar was a ruler \( \text{ruler}(\text{caesar}) \)
  - All romans were loyal to Caesar or they hated him
    \[ \forall X \text{ roman}(X) \rightarrow (\text{loyalto}(X, \text{caesar}) \lor \text{hate}(X, \text{caesar})) \]
  - Everyone is loyal to someone \( \forall X \exists Y \text{ loyalto}(X, Y) \)

- We can now reason about this knowledge using standard deductive mechanisms.
- But there is in principle no guarantee that we will prove a given theorem.

Towards Efficient Automated Deduction

- Automated deduction is search.
- Complexity of search: directly dependent on branching factor at nodes (exponentially!).
- It is vital to cut down the branching factor:
  - Canonical representation of nodes (allows identifying identical nodes).
  - As few inference rules as possible.
Towards Efficient Automated Deduction (Contd.)

Clausal Form

- The complete set of logical operators (\(\neg, \land, \lor, \neg\),...) is redundant.
- A minimal (canonical) form would be interesting.
- It would be interesting to separate the quantifiers from the rest of the formula so that they did not need to be considered.
- It would also be nice if the formula were flat (i.e. no parenthesis).
- Conjunctive normal form has these properties [Davis 1960].

Deduction Mechanism

- A good example:
  Resolution – only two inference rules (Resolution rule and Replacement rule).

Classical Clausal Form: Conjunctive Normal Form

- General formulas are converted to:
  - Set of Clauses.
  - Clauses are in a logical conjunction.
  - A clause is a disjunction of the form. \(\text{literal}_1 \lor \text{literal}_2 \lor \ldots \lor \text{literal}_n\)
  - The \(\text{literal}_i\) are negated or non-negated atoms.
  - All variables are implicitly universally quantified: i.e. if \(X_1, \ldots, X_k\) are the variables that appear in a clause it represents the formula:
    \(\forall X_1, \ldots, X_k \quad \text{literal}_1 \lor \text{literal}_2 \lor \ldots \lor \text{literal}_n\)
- Any formula can be converted to clausal form automatically by:
  1. Converting to Prenex form.
  2. Converting to conjunctive normal form (conjunction of disjunctions).
  3. Converting to Skolem form (eliminating existential quantifiers).
  4. Eliminating universal quantifiers.
  5. Separating conjunctions into clauses.
- The unsatisfiability of a system is preserved.
Substitutions

A substitution is a finite mapping from variables to terms, written as 
\[ \theta = \{ X_1/t_1, ..., X_n/t_n \} \] where
- the variables \( X_1, ..., X_n \) are different,
- for \( i = 1, ..., n \) \( X_i \equiv t_i \).

A pair \( X_i/t_i \) is called a binding.
- \( \text{domain}(\theta) = \{ X_1, ..., X_n \} \) and \( \text{range}(\theta) = \text{vars}(\{ t_1, ..., t_n \}) \).
- If \( \text{range}(\theta) = \emptyset \) then \( \theta \) is called ground.
- If \( \theta \) is a bijective mapping from variables to variables then \( \theta \) is called a renaming.

Examples:
- \( \theta_1 = \{ X/f(A), Y/X, Z/h(b, Y), W/a \} \)
- \( \theta_2 = \{ X/a, Y/a, Z/h(b, c), W/f(d) \} \) (ground)
- \( \theta_3 = \{ X/A, Y/B, Z/C, W/D \} \) (renaming)

Substitutions (Contd.)

Substitutions operate on expressions, i.e. a term, a sequence of literals or a clause, denoted by \( E \).
- The application of \( \theta \) to \( E \) (denoted \( E\theta \)) is obtained by simultaneously replacing each occurrence in \( E \) of \( X_i \) by \( t_i \), \( X_i/t_i \in \theta \).
- The resulting expression \( E\theta \) is called an instance of \( E \).
- If \( \theta \) is a renaming then \( E\theta \) is called a variant of \( E \).

Example:
- \( \theta_1 = \{ X/f(A), Y/X, Z/h(b, Y), W/a \} \)
- \( p(X, Y, X) \ \theta_1 = p(f(A), X, f(A)) \)
Composition of Substitutions

- Given \( \theta = \{X_1/t_1, \ldots, X_n/t_n\} \) and \( \eta = \{Y_1/s_1, \ldots, Y_m/s_m\} \) their composition \( \theta \eta \) is defined by removing from the set
  \[ \{X_1/t_1\eta, \ldots, X_n/t_n\eta, Y_1/s_1, \ldots, Y_m/s_m\} \]
  those pairs \( X_i/t_i \eta \) for which \( X_i = t_i \eta \), as well as those pairs \( Y_i/s_i \) for which \( Y_i \in \{X_1, \ldots, X_n\} \).

- Example: if \( \theta = \{X/3, Y/f(X, 1)\} \) and \( \eta = \{X/4\} \) then \( \theta \eta = \{X/3, Y/f(4, 1)\} \).

- For all substitutions \( \theta, \eta \) and \( \gamma \) and an expression \( E \)
  i) \( (E\theta)\eta \equiv E(\theta\eta) \)
  ii) \( (\theta\eta)\gamma = \theta(\eta\gamma) \).

- \( \theta \) is more general than \( \eta \) if for some \( \gamma \) we have \( \eta = \theta\gamma \).

- Example: \( \theta = \{X/f(Y)\} \) more general than \( \eta = \{X/f(h(G))\} \).

Unifiers

- If \( A\theta \equiv B\theta \), then
  - \( \theta \) is called a unifier of \( A \) and \( B \)
  - \( A \) and \( B \) are unifiable

- A unifier \( \theta \) of \( A \) and \( B \) is called a most general unifier (mgu) if it is more general than any other unifier of \( A \) and \( B \).

- If two atoms are unifiable then they have a most general unifier.

- \( \theta \) is idempotent if \( \theta \theta = \theta \).

- A unifier \( \theta \) of \( A \) and \( B \) is relevant if all variables appearing either in \( \text{domain}(\theta) \) or in \( \text{range}(\theta) \), also appear in \( A \) or \( B \).

- If two atoms are unifiable then they have an mgu which is idempotent and relevant.

- An mgu is unique up to renaming.
Uniﬁcation Algorithm

- Non-deterministically choose from the set of equations an equation of a form below and perform the associated action.
  1. \( f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n) \rightarrow \text{replace by } s_1 = t_1, \ldots, s_n = t_n \)
  2. \( f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m) \text{ where } f \neq g \rightarrow \text{halt with failure} \)
  3. \( X = X \rightarrow \text{delete the equation} \)
  4. \( t = X \text{ where } t \text{ is not a variable } \rightarrow \text{replace by the equation } X = t \)
  5. \( X = t \text{ where } X \neq t \text{ and } X \text{ has another occurrence in the set of equations } \rightarrow \)
      5.1 if \( X \) appears in \( t \) then halt with failure
      5.2 otherwise apply \( \{X/t\} \) to every other equation

- Consider the set of equations \( \{f(x) = f(f(z)), g(a, y) = g(a, x)\} \):
  - (1) produces \( \{x = f(z), g(a, y) = g(a, x)\} \)
  - then (1) yields \( \{x = f(z), a = a, y = x\} \)
  - (3) produces \( \{x = f(z), y = x\} \)
  - now only (5) can be applied, giving \( \{x = f(z), y = f(z)\} \)
  - No step can be applied, the algorithm successfully terminates.

Unifi cation Algorithm revisited

- Let \( A \) and \( B \) be two formulas:
  1. \( \theta = \epsilon \)
  2. while \( A\theta \neq B\theta \):
      2.1 find leftmost symbol in \( A\theta \) s.t. the corresponding symbol in \( B\theta \) is different
      2.2 let \( t_A \) and \( t_B \) be the terms in \( A\theta \) and \( B\theta \) starting with those symbols
          (a) if neither \( t_A \) nor \( t_B \) are variables or one is a variable occurring in the other \( \rightarrow \) halt with failure
          (b) otherwise, let \( t_A \) be a variable \( \rightarrow \) the new \( \theta \) is the result of \( \theta\{t_A/t_B\} \)
  3. end with \( \theta \) being an m.g.u. of \( A \) and \( B \)
Example: $A = p(X, X)$ $B = p(f(A), f(B))$

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$A\theta$</th>
<th>$B\theta$</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$p(X, X)$</td>
<td>$p(f(A), f(B))$</td>
<td>${X/f(A)}$</td>
</tr>
<tr>
<td>${X/f(A)}$</td>
<td>$p(f(A), f(A))$</td>
<td>$p(f(A), f(B))$</td>
<td>${A/B}$</td>
</tr>
<tr>
<td>${X/f(B), A/B}$</td>
<td>$p(f(B), f(B))$</td>
<td>$p(f(B), f(B))$</td>
<td></td>
</tr>
</tbody>
</table>

Example: $A = p(X, f(Y))$ $B = p(Z, X)$

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>$A\theta$</th>
<th>$B\theta$</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon$</td>
<td>$p(X, f(Y))$</td>
<td>$p(Z, X)$</td>
<td>${X/Z}$</td>
</tr>
<tr>
<td>${X/Z}$</td>
<td>$p(Z, f(Y))$</td>
<td>$p(Z, Z)$</td>
<td>${Z/f(Y)}$</td>
</tr>
<tr>
<td>${X/f(Y), Z/f(Y)}$</td>
<td>$p(f(Y), f(Y))$</td>
<td>$p(f(Y), f(Y))$</td>
<td></td>
</tr>
</tbody>
</table>

Resolution with Variables

- It is a formal system with:
  - A first order language with the following formulas:
    - Clauses: without repetition, and without an order among their literals.
    - The empty clause $\Box$.
  - An empty set of axioms.
  - Two inference rules: resolution and replacement.
Resolution with Variables (Contd.)

- Resolution:
  \[ r_1: A \lor F_1 \lor \cdots \lor F_n \]
  \[ r_2: \neg B \lor G_1 \lor \cdots \lor G_m \]
  \[ \theta \]
  \[ (\{F_1 \lor \cdots \lor F_n\} \sigma \lor G_1 \lor \cdots \lor G_m) \theta \]

  where
  - \( A \) and \( B \) are unifiable with substitution \( \theta \)
  - \( \sigma \) is a renaming s.t. \( (A \lor F_1 \lor \cdots \lor F_n) \sigma \) and \( \neg B \lor G_1 \lor \cdots \lor G_m \) have no variables in common
  - \( \theta \) is the m.g.u. of \( A \sigma \) and \( B \)

  The resulting clause is called the resolvent of \( r_1 \) and \( r_2 \).

- Replacement: \( A \lor B \lor F_1 \lor \cdots \lor F_n \Rightarrow (A \lor F_1 \lor \cdots \lor F_n) \theta \) where
  - \( A \) and \( B \) are unifiable atoms
  - \( \theta \) is the m.g.u. of \( A \) and \( B \)

Basic Properties

- Resolution is correct – i.e. all conclusions obtained using it are valid.
- There is no guarantee of directly deriving a given theorem.
- However, resolution (under certain assumptions) is refutation complete: if we have a set of clauses \( K = [C_0, C_1, \ldots, C_n] \) and it is inconsistent then resolution will arrive at the empty clause \( \Box \) in a finite number of steps.
- Therefore, a valid theorem (or a question that has an answer) is guaranteed to be provable by refutation. To prove "p" given \( K_0 = [C_0, C_1, \ldots, C_n] \):
  1. Negate it \( \neg p \).
  2. Construct \( K = [\neg p, C_0, C_1, \ldots, C_n] \).
  3. Apply resolution steps repeatedly to \( K \).
- Furthermore, we can obtain answers by composing the substitutions along a path that leads to \( \Box \) (very important for realizing Greene’s dream!).
- It is important to use a good method in applying the resolution steps – i.e. in building the resolution tree (or proof tree).
- Again, the main issue is to reduce the branching factor.
Proof Tree

- Given a set of clauses $K = \{C_0, C_1, \ldots, C_n\}$ the proof tree of $K$ is a tree s.t.: 
  - the root is $C_0$
  - the branch from the root starts with the nodes labeled with $C_0, C_1, \ldots, C_n$
  - the descendent nodes of $C_n$ are labeled by clauses obtained from the parent clauses using resolution
  - a derivation in $K$ is a branch of the proof tree of $K$
- The derivation $C_0C_1\cdots C_nF_0\cdots F_m$ is denoted as $K, F_0\cdots F_m$

Proof Tree (Contd.)

- Example: part of the proof tree for $K$, with:
  
  $K = \{p, \neg p \lor q, \neg q\}$
  
  \[
  \begin{array}{c}
  p \equiv C_0 \\
  \neg p \lor q \equiv C_1 \\
  \neg q \equiv C_2 \\
  \hline
  R(C_0, C_1) \equiv q \\
  \neg p \equiv R(C_1, C_2) \\
  \hline
  \hline
  \fbox{\(\neg p \equiv R(C_1, C_2)\)} \\
  \hline
  \fbox{\(R(C_0, C_1) \equiv q\)} \\
  \end{array}
  \]
Characteristics of the Proof Tree

- It can be infinite: \( K = \{ p(e), \neg p(X) \lor p(f(X)) \} \)
  
  \[
  \begin{align*}
  p(e) & \quad \text{C0} \\
  \neg p(X) \lor p(f(X)) & \quad \text{C1} \\
  p(f(e)) & \quad \emptyset = \{X/e\} \\
  p(f(f(e))) & \quad \emptyset = \{X/f(e)\}
  \end{align*}
  \]

- Even if it is finite, it can be too large to be explored efficiently
- Aim: determine some criteria to limit the number of derivations and the way in which the tree is explored ⇒ strategy
- Any strategy based on this tree is correct: if \( \square \) appears in a subtree of the proof tree of \( K \), then \( \square \) can be derived from \( K \) and therefore \( K \) is unsatisfiable

General Strategies

- Depth-first with backtracking: First descendant to the left; if failure or \( \square \) then backtrack
General Strategies (Contd.)

- Breadth first: all sons of all sibling nodes from left to right

```
  1
 / \
2   3
 |   |
4   5   6  7
|   |   |   |
8   5   6   7
   Fail Fail
```

General Strategies (Contd.) (Contd.)

- Iterative deepening
  - Advance depth-first for a time.
  - After a certain depth, switch to another branch as in breadth-first.

- Completeness issues / possible types of branches:
  - Success (always finite)
  - Finite failure
  - Infinite failure (provably infinite branches)
  - Non-provably infinite branches
Linear Strategies

- Those which only explore linear derivations
- A derivation $K, F_0 \cdots F_m$ is linear if
  - $F_0$ is obtained by resolution or replacement using $C_0$
  - $F_i, i < 0$ is obtained by resolution or replacement using $F_{i-1}$
- Examples:

Examples:

```
\begin{align*}
p & \equiv C_0 \\
\neg p \lor q & \equiv C_1 \\
\neg q & \equiv C_2 \\
q & \equiv F_0
\end{align*}
```

```
\begin{align*}
\neg p \lor q & \equiv C_0 \\
p & \equiv C_1 \\
\neg q & \equiv C_2
\end{align*}
```

Characteristics of these Strategies

1. If $\square$ can be derived from $K$ by using resolution with variables, it can also be derived by linear resolution.
2. Let $K$ be $K' \cup \{C_0\}$ where $K'$ is a satisfiable set of clauses, i.e. $\square$ cannot be derived from $K'$ by using resolution with variables. If $\square$ can be derived from $K$ by using resolution with variables it can also be derived by linear resolution with root $C_0$.
- From (1), if the strategy is breadth first, it is complete.
- From (2), if we want to prove that $B$ is derived form $K'$ then we can apply linear resolution to $K = K' \cup \{\neg B\}$.
- Depth first with backtracking is not complete:
Input Strategies

- Those which only explore input derivations
- A derivation \( K, F_0 \cdot \cdot \cdot F_m \) is input if
  - \( F_0 \) is obtained by resolution or replacement using \( C_0 \)
  - \( F_i, i < 0 \) is obtained by resolution or replacement using at least a clause in \( K \)

\[ K = \{
-\neg p \lor q, p \lor \neg r, r, q \lor \neg s, s \lor q
\} \]

\[ \begin{array}{c}
-\neg p \lor \neg q & \text{C0} \\
p \lor \neg r & \text{C1} \\
r & \text{C2} \\
q \lor \neg s & \text{C3} \\
s \lor q & \text{C4} \\
\neg q \lor \neg r & \text{C1 (\&C0)} \\
\end{array} \]

Input + Linear

In an input derivation, if \( F_{i-1} \) does not appear in any derivation of a successor clause, it can be eliminated from the derivation without changing the result

- If \( F_{i-1} \) appears in the derivation of \( F_j, j > 1 \), \( F_{i-1} \) can be allocated in position \( j - 1 \)
- As a result, we can limit ourselves to linear input derivations without losing any input derivable clause

Let \( K \) be \( K^* \cup \{C_0\} \) where \( \Box \) is derived by using resolution with variables, \( C_0 \) is a negative Horn clause and all clauses in \( K^* \) are positive Horn clauses. There is an input derivation with root \( C_0 \) finishing in \( \Box \) and in which the replacement rule is not used (Hernschen 1974)

A Horn clause is a clause in which at most one literal is positive:
  - it is positive if precisely one literal is positive
  - it is negative if all literals are negatives
- As a result, in those conditions, a breadth first input strategy is complete, and a depth first input strategy with backtracking is complete if the tree is finite.
Ordered Strategies

- We consider a new formal system in which:
  1. clauses are ordered sets
  2. ordered resolution of two clauses
     \[ A = p_1 \lor \cdots \lor p_n \quad \text{and} \quad B = q_1 \lor \cdots \lor q_m \]
     where \( p_1 \) is a positive literal and \( q_1 \) is a negative literal is possible iff \( \neg p_1 \) and \( \sigma(q_1) \) are unifiable (\( \sigma \) is a renaming, s.t. \( p_1 \) and \( \sigma(q_1) \) have no variables in common)
  3. the resolvent of \( A \) and \( B \) is \( \theta(p_2 \lor \cdots \lor p_n \lor \sigma(q_2 \lor \cdots \lor q_m)) \) where \( \theta \) is an m.g.u of \( \neg p_1 \) and \( \sigma(q_1) \)

- Let \( K = K' \cup \{ C_0 \} \) be a set of clauses s.t. \( \square \) is derived by using resolution with variables, \( C_0 \) is a negative Horn clause and all clauses in \( K' \) are positive Horn clauses with the positive literal in the first place. There is a sorted input derivation with root \( C_0 \) arriving at \( \square \).

- In this context a sorted linear input with:
  - breadth first: is complete
  - depth first with backtracking: is complete if the tree is finite