Syntax of Propositional Logic

Formulae in propositional logic are formed with propositions and connectives

- Given a set of propositions
- For the connectives \( \land, \lor, \Rightarrow, \neg \)
  - \( p \) is a formula if \( p \) is a proposition
  - \( (F_1 \land F_2) \) is a formula if \( F_1 \) and \( F_2 \) are formulae
  - \( (F_1 \lor F_2) \) is a formula if \( F_1 \) and \( F_2 \) are formulae
  - \( (F_1 \Rightarrow F_2) \) is a formula if \( F_1 \) and \( F_2 \) are formulae
  - \( \neg (F) \) is a formula if \( F \) is formulae
  - And nothing else is a formula

Model Semantic of Propositional Logic

Semantics of connectives is given by truth tables

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>((A \lor B))</th>
<th>((A \land B))</th>
<th>((A \Rightarrow B))</th>
<th>(!!(A)!!)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

\((- A \lor B)\)  

If we give the truth values according to \( I \), the formula is true.

\( I \) is a \textit{model}.

Proof Semantics of Propositional Logic

Let us consider formulae composed of:

- propositions \( p \),
- negation of propositions \( \neg p \) and
- implications \( F_1 \Rightarrow F_2 \)

Let us consider the inference rule:

if \( F \) and \( F \Rightarrow Q \) then \( Q \) (modus ponens)

\((\text{all models of } "F \text{ and } F \Rightarrow Q" \text{ are models of } "Q"

\text{(If } "Q" \text{ has no model } \neg \text{ is false } \text{ then } "F \text{ and } F \Rightarrow Q" \text{ has no model})

Proof Semantics of Propositional Logic

Let us consider formulae composed of:

propositions \( p \),
- negation of propositions \( \neg p \) and
- implications \( F_1 \Rightarrow F_2 \)

Let us consider the inference rule:

if \( \neg Q \) and \( F \Rightarrow Q \) then \( \neg F \) (modus tollens)
Proof Semantics of Propositional Logic

Let us consider formulae composed of:

propositions p, (positive literal)

negation of propositions \(\neg p\) (negative literal) and
disjunctions \(l_1 \lor \ldots \lor l_n\) (I1 and I2 are either positive or negative literals)

Let us consider the inference rule:

if \(p \lor F_1\) and \(\neg p \lor F_2\) then \(F_1 \lor F_2\) (resolution)

Clauses and Horn Clauses

Formulae of the form \(l_1 \lor \ldots \lor l_n\) are called **clauses**

The empty clause (sometimes written \(\Box\), \(\bot\) or \(\emptyset\)) is false

Clauses that contain at most one positive literal are called **Horn clauses** or **definite clauses**

First Order Logic

In first order logic literals are formed with predicates and variables

Variables can be free, existentially quantified or universally quantified

\[\forall X \exists Y (p(X) \Rightarrow (q(X, Y) \lor \neg r(X, Y)))\]

Prolog programs are made of Horn Clauses

grand_parent(X, Y):- parent(X, Z), parent(Z, Y).

\[\forall X \forall Y \forall Z \exists (\text{parent}(X, Z) \land \text{parent}(Z, Y))\]

\[\forall X \forall Y \forall Z (\text{grand_parent}(X, Y) \lor \neg \text{parent}(X, Z) \lor \neg \text{parent}(Z, Y))\]

Prolog programs are made of Horn Clauses

\[\text{parent}(X, Y)\text{- father}(X, Y).\]

\[\text{parent}(X, Y)\text{- mother}(X, Y).\]

\[\text{grand_parent}(X, Y)\text{- parent}(X, Z), \text{parent}(Z, Y).\]

\[\text{father}(\text{"Louis XVIII"}, \text{"Dauphin Louis"}).\]

\[\text{father}(\text{"Maria LEZINSKA"}, \text{"Stanislaw LEZINSKI"}).\]

\[\text{father}(\text{"Dauphin Louis"}, \text{"Louis XV"}).\]

\[\text{father}(\text{"Louis XV"}, \text{"Louis, Duke of Burgundy"}).\]

\[\text{mother}(\text{"Dauphin Louis"}, \text{"Maria LEZINSKA"}).\]

\[\text{mother}(\text{"Louis XVIII"}, \text{"Marie-Josephe"}).\]

\[\text{mother}(\text{"Louis XV"}, \text{"Marie Adelaide"}).\]

\[\neg \text{grand_parent}(\text{"Louis XVIII"}, Y).\]

\[\forall Y (\neg \text{grand_parent}(\text{"Louis XVIII"}, Y))\]

\[\forall X \forall Y \forall Z (\text{grand_parent}(X, Y) \lor \neg \text{parent}(X, Z) \lor \neg \text{parent}(Z, Y))\]

\[\forall Y \forall Z (\neg \text{parent}(\text{"Louis XVIII"}, Z) \lor \neg \text{parent}(Z, Y))\]

Which is the goal (involves unification – in this case only filtering)

\[\text{parent}(\text{"Louis XVIII"}, Z), \text{parent}(Z, Y)).\]

Selected Linear Definite (SLD) Resolution

\[\Rightarrow \text{grand_parent}(\text{"Louis XVIII"}, Y)\]

\[\forall Y (\neg \text{grand_parent}(\text{"Louis XVIII"}, Y))\]

\[\forall X \forall Y \forall Z (\text{grand_parent}(X, Y) \lor \neg \text{parent}(X, Z) \lor \neg \text{parent}(Z, Y))\]

\[\forall Y \forall Z (\neg \text{parent}(\text{"Louis XVIII"}, Z) \lor \neg \text{parent}(Z, Y))\]

Which is the goal (involves unification – in this case only filtering)

\[\Rightarrow \text{parent}(\text{"Louis XVIII"}, Z), \text{parent}(Z, Y)).\]
Logic Programming and Constraints

SLD Tree

\[ G1 \leftarrow G4, G5. \]
\[ G1 \leftarrow G4, G6. \]
\[ G2. \]
\[ G3. \]
\[ G3 \leftarrow G6. \]
\[ G4. \]
\[ G6. \]

Backtracking

- Prolog computes the SLD tree in pre-order (depth-first)
- Prolog Backtracks when it fails
- Prolog backtrack after a success if the user requests the next solution

The Stack

Depth-first search is associated with the stack data structure.
A stack is a structure where entries are pushed onto the top of the stack and are popped off (ie taken off) the top of the stack. The process is sometimes known as LIFO (Last In, First Out) or FILO (First In, Last Out).

The Cut!

The special procedure \\
always succeeds

The cut removes all choice points between the node of the SLD tree where it is evaluated and the parent node of the node where it was introduced

The Cut!

\[ G1 \leftarrow G4, G5, !. \]
\[ G1 \leftarrow G4, G6. \]
\[ G2. \]
\[ G3. \]
\[ G3 \leftarrow G6. \]
\[ G4. \]
\[ G6. \]
Logic Programming and Constraints

The Cut!

```prolog
:- G1, G2, G3.
G1 :- G4, G5.
G1 :- G4, G6.
G2.
G3 !.
G3 :- G6.
G4.
G6.
```

Negation as ! Failure

Negation as failure uses !/0 and fail/0 (and possibly call/1)

```prolog
neg(Goal) :- Goal,! fail.
neg(Goal).

neg(Goal) :- call(Goal),!,fail.
neg(Goal).
```

Control

- !
  - Cut - succeeds and removes all choice points between cut and parent goal.
- fail
  - Does not succeed. A synonym of false/0.
- not Goal
  - Succeeds if Goal cannot be satisfied. Uses negation as failure.
- +Goal1 ; +Goal2
  - Semicolon (OR) operator - Succeeds if the goal Goal1 succeeds or if the goal Goal2 succeeds.
- +Condition -> +Then ; +Else
  - Conditional construct - succeeds if either Condition succeeds, and then goal Then succeeds; or else if Condition fails, and then Else succeeds.
- call(+Goal)
  - Succeeds if Goal succeeds.
- repeat
  - succeeds as often as tried.

Credits

Clipart and media are licensed from Microsoft Office Online Clipart and Media

Copyright © 2008 by Stéphane Bressan