Recap (Lecture 1): Propositional logic program

A **definite clause** is a formula of one of the two shapes below

\[ q \] (a Prolog **fact** \( q \).)
\[ p_1 \land \cdots \land p_k \rightarrow q \] (a Prolog **rule** \( q : - p_1, \ldots, p_k \).)

where \( p_1, \ldots, p_k, q \) are all **atoms** (that is, atomic statements).

A **logic program** is a list \( F_1, \ldots, F_n \) of definite clauses

A **goal** is a list \( g_1, \ldots, g_m \) of atoms.

The job of the system is to ascertain whether the logical consequence below holds.

\[ F_1, \ldots, F_n \models g_1 \land \cdots \land g_m . \]

---

**Example logic program**

\[ \text{hotterSun} \rightarrow \text{warmerClimate} \]
\[ \text{carbonIncrease} \rightarrow \text{warmerClimate} \]
\[ \text{warmerClimate} \rightarrow \text{iceMelts} \]
\[ \text{iceMelts} \rightarrow \text{albedoDecrease} \]
\[ \text{albedoDecrease} \rightarrow \text{warmerClimate} \]
\[ \text{carbonIncrease} \]

---

Recap (Lecture 1): Inference system

The definite clauses \( F_1, \ldots, F_n \) in the program are taken as axioms.

We use a single inference rule to derive new facts from facts already established

\[
\begin{array}{cccc}
p_1 & p_2 & \cdots & p_k \\
\hline
p_1 \land p_2 \land \cdots \land p_k \rightarrow q
\end{array}
\] (Backchain)

\[ q \]

A **derivation** (or **proof**) of an individual goal, \( g_i \), is a tree of applications of the Backchain rule, in which every leaf is one of the definite clauses \( F_1, \ldots, F_n \) comprising the program, and the root of the tree is \( g_i \).

A **derivation** of the compound goal \( g_1, \ldots, g_m \), is provided by a list of \( m \) derivations, one for each individual goal \( g_1, \ldots, g_m \).
A derivation of the goal \textit{iceMelts}

\begin{align*}
\text{carbonIncrease} & \quad \text{carbonIncrease} \rightarrow \text{warmerClimate} \\
\text{warmerClimate} & \quad \text{warmerClimate} \rightarrow \text{iceMelts} \\
\text{iceMelts} & 
\end{align*}

\textbf{Top-down proof search}

A top-down search for a derivation maintains a list \([g_1, \ldots, g_n]\) of atoms, the \textit{current goal}:

1. If \(g_1\) is an axiom it gets removed from the list.
2. Otherwise, find an axiom (a Prolog rule) of the form
   \[ h_1 \land \cdots \land h_k \rightarrow g_1 \]
   in the program. Replace \(g_1\) with \(h_1, \ldots, h_k\). So the current goal becomes
   \[[h_1, \ldots, h_k, g_2, \ldots, g_n]\]

The choice of axiom in step 2 means that the search space branches, producing a \textit{search tree}.

The goal of the proof search procedure is to find a branch ending in a leaf labelled with the empty list \([],\).  

\textbf{Search tree for goal \([\text{iceMelts}]\)}

\textbf{Prolog proof search}

Prolog proof search is \textit{depth first}:

- The search always moves to an \textit{unvisited} (i.e., not previously visited) \textit{child} (i.e., immediately below) node of the current node, whenever such a node exists.
- If there is no such node, the search backtracks to the most recently visited node from which such an unvisited child node is available.

Prolog proof search follows \textit{program order}.

- Child nodes are visited in the order that the axioms (Prolog rules) that determine the child node appear in the program (i.e., from left to right in the trees as we are drawing them).
Prolog search for goal \[\text{iceMelts}\]

Resulting Prolog search

Reordering program can improve efficiency . . .

    carbonIncrease \rightarrow \text{warmerClimate}
    \text{hotterSun} \rightarrow \text{warmerClimate}
    \text{warmerClimate} \rightarrow \text{iceMelts}
    \text{albedoDecrease} \rightarrow \text{warmerClimate}

... or reduce efficiency . . .

    \text{hotterSun} \rightarrow \text{warmerClimate}
    \text{albedoDecrease} \rightarrow \text{warmerClimate}
    \text{carbonIncrease} \rightarrow \text{warmerClimate}
    \text{iceMelts} \rightarrow \text{albedoDecrease}

In Sicstus Prolog

Program:
warmerClimate :- hotterSun.
warmerClimate :- albedoDecrease.
warmerClimate :- carbonIncrease.
iceMelts :- warmerClimate.
albedoDecrease :- iceMelts.
carbonIncrease.
hotterSun :- false.

Query:
?- iceMelts.
! Resource error: insufficient memory

Incompleteness of Prolog proof search

The correct answer to the query iceMelts is yes, because iceMelts is a logical consequence of the program (irrespective of how we order the axioms in the program).

Our original derivation of iceMelts is a valid derivation (irrespective of how we order the axioms in the program).

However, whether Prolog proof search is successful or not depends on the order in which the axioms in the program are given.

The Prolog proof search procedure is said to be incomplete: it does not always find a derivation, even if a derivation exists.

Notation

Let $F_1, F_2, \ldots, F_n$ be definite clauses (our axioms/program)

- Atom $g$ is a logical consequence of $F_1, F_2, \ldots, F_n$
  \[ F_1, \ldots, F_n \models g \]

- Atom $g$ is derivable from $F_1, F_2, \ldots, F_n$:
  \[ F_1, \ldots, F_n \vdash g \]

- Atom $g$ is Prolog derivable from $F_1, F_2, \ldots, F_n$:
  \[ F_1, \ldots, F_n \vdash_{\text{Prolog}} g \]

i.e., Prolog proof search is successful.
Fundamental properties

Correctness of Prolog proof search (will prove third)
\[
F_1, \ldots, F_n \vdash_{\text{Prolog}} g \quad \text{implies} \quad F_1, \ldots, F_n \vdash g.
\]

Soundness of inference system (will prove first)
\[
F_1, \ldots, F_n \vdash g \quad \text{implies} \quad F_1, \ldots, F_n \models g.
\]

Completeness of inference system (will prove second)
\[
F_1, \ldots, F_n \models g \quad \text{implies} \quad F_1, \ldots, F_n \vdash g.
\]

Incompleteness of Prolog proof search (already shown)
\[
F_1, \ldots, F_n \models g \quad \text{does not imply} \quad F_1, \ldots, F_n \vdash_{\text{Prolog}} g.
\]

Proof of soundness

We need to prove:
\[
F_1, \ldots, F_n \vdash g \quad \text{implies} \quad F_1, \ldots, F_n \models g.
\]

Suppose that \( F_1, \ldots, F_n \vdash g \).
Let \( I \) be an interpretation for which
\[
I \models F_1 \quad \text{and} \quad \ldots \quad \text{and} \quad I \models F_n
\]
We need to show that \( I \models g \).
To this end, let \( \Pi \) be a derivation tree for \( g \).
We show on next slide that, for every atom \( q \) that occurs in \( \Pi \), it holds that \( I \models q \).
Thus, in particular, \( I \models g \), as required.

Proof of soundness (continued)

We show: for every atom \( q \) that occurs in \( \Pi \), it holds that \( I \models q \).
We work from the leaves (axioms) of \( \Pi \) downwards, propagating the property that \( I \models q \) as we go.

- If \( q \) is an axiom (one of \( F_1, \ldots, F_n \)), \( I \models q \) is immediate.
- Otherwise the derivation of \( q \) within \( \Pi \) must look like

\[
\vdots \quad \vdots
\]
\[
p_1 \quad \ldots \quad p_k \quad p_1 \land \cdots \land p_k \rightarrow q
\]

where \( p_1 \land \cdots \land p_k \rightarrow q \) is an axiom.
By the property we are propagating through the tree,
\[
I \models p_1 \quad \text{and} \quad \ldots \quad \text{and} \quad I \models p_k.
\]
Because \( p_1 \land \cdots \land p_k \rightarrow q \) is an axiom,
\[
I \models p_1 \land \cdots \land p_k \rightarrow q.
\]
Thus indeed \( I \models q \).

Illustration of the propagation process

\[
\begin{align*}
(1) I \models cI & \quad (1) I \models cI \rightarrow WC \\
(2) I \models WC & \quad (2) I \models WC \\
(3) I \models iceMelts
\end{align*}
\]
Proof of completeness

We need to prove:

\[ F_1, \ldots, F_n \models g \implies F_1, \ldots, F_n \vdash g. \]

Suppose that \( F_1, \ldots, F_n \models g \).
We need to show that \( F_1, \ldots, F_n \vdash g \).

To this end, define the following interpretation \( I \):

\[
I(q) = \begin{cases} 
\text{true} & \text{if } F_1, \ldots, F_n \vdash q \\
\text{false} & \text{otherwise}
\end{cases}
\]

We show on next slide that

\[ I \models F_1 \text{ and } \ldots \text{ and } I \models F_n \]

Hence \( I \models g \). That is, \( I(g) = \text{true} \). That is, \( F_1, \ldots, F_n \vdash g \).

Proof of correctness of Prolog search

We prove, more generally:

If the search tree for goal \([g_1, \ldots, g_m]\) has a branch ending in a leaf labelled [], then \( F_1, \ldots, F_n \vdash g_i \) for every \( g_i \) in \( g_1, \ldots, g_m \).

To prove this, we show that, for every goal list \([q_1, \ldots, q_i]\) appearing on the branch ending in [], it holds that \( F_1, \ldots, F_n \vdash q_i \) for every \( q_i \) in \( q_1, \ldots, q_i \).

We propagate this property up the branch, starting with the leaf labelled [], and working up to the original goal \([g_1, \ldots, g_m]\) at the root of the search tree.

▸ For the leaf [] there is nothing to show, since the current goal list is empty.

Proof of completeness (continued)

\[ I(q) = \begin{cases} 
\text{true} & \text{if } F_1, \ldots, F_n \vdash q \\
\text{false} & \text{otherwise}
\end{cases} \]

We show that \( I \models F_i \) for every \( F_i \) in \( F_1, \ldots, F_n \).

▸ If \( F_i \) is an atom \( q \) then trivially \( F_1, \ldots, F_n \vdash q \). So \( I(q) = \text{true} \). i.e., \( I \models q \).

▸ In the case \( F_i \) is an implication \( p_1 \land \cdots \land p_k \rightarrow q \), we need to show \( I \models p_1 \land \cdots \land p_k \rightarrow q \). Suppose \( I \models p_1 \) and \ldots and \( I \models p_k \). That is, \( F_1, \ldots, F_n \vdash p_1 \) and \ldots and \( F_1, \ldots, F_n \vdash p_k \).
Combine the derivations of \( p_1, \ldots, p_k \) to get one of \( q \):

\[
\vdots \vdots \\
p_1 \quad \ldots \quad p_k \\
\vdots \\
p_1 \land \cdots \land p_k \rightarrow q
\]

\[ q \]

So \( F_1, \ldots, F_n \vdash q \). Thus \( I \models q \). \( \square \)

Proof of correctness of Prolog search (continued)

▸ For any other node \([q_1, \ldots, q_i]\) on the branch, the branch continues (locally)

\[
[q_1, \ldots, q_i] \models [p_1, \ldots, p_k, q_2, \ldots, q_i]
\]

for some implication \( p_1 \land \cdots \land p_k \rightarrow q_1 \) in \( F_1, \ldots, F_n \).

By the property we are propagating up the branch, \( F_1, \ldots, F_n \vdash p_i \) for every \( p_i \) in \( p_1, \ldots, p_k \), and \( F_1, \ldots, F_n \vdash q_i \) for every \( q_i \) in \( q_2, \ldots, q_i \). We build a derivation for \( q_1 \) by:

\[
\vdots \vdots \\
p_1 \quad \ldots \quad p_k \\
\vdots \\
p_1 \land \cdots \land p_k \rightarrow q_1
\]

\[ q_1 \]

So indeed \( F_1, \ldots, F_n \vdash q_i \) for every \( q_i \) in \( q_1, \ldots, q_i \). \( \square \)
Illustration of the propagation process

<table>
<thead>
<tr>
<th>[iM]</th>
<th>(4) Program ⊢ iM</th>
</tr>
</thead>
<tbody>
<tr>
<td>[wC]</td>
<td>(3) Program ⊢ wC</td>
</tr>
<tr>
<td>[cI]</td>
<td>(2) Program ⊢ cI</td>
</tr>
<tr>
<td></td>
<td>(1) nothing to show</td>
</tr>
</tbody>
</table>

Discussion

We have a complete inference system for propositional definite clause logic, but Prolog’s proof search procedure is incomplete. Incompleteness can be remedied in different ways, e.g.:

- Add loop checking to the Prolog search procedure to check when the search reconsiders an atom already encountered. This provides a decision procedure for propositional definite-clause logic: the search will say yes if a derivation exists, and no if no derivation exists.

- Use breadth-first search instead of depth-first search. This gives a complete proof procedure (a derivation will be found whenever one exists) without loop checking, but not a decision procedure. Loop checking needs to be added in order to detect when no derivation is available.

Loop checking

Breadth-first search
Discussion (continued)

There are sound pragmatic reasons, however, that Prolog does not implement such modified search algorithms.

- Loop checking makes proof search less efficient, and it is anyway not a useful modification once we move to the predicate (first-order) logic used by Prolog (where non-terminating behaviour can occur without the same goal ever repeating itself).

- Breadth-first search is in general inefficient, since it always explores all branches, irrespective of their usefulness.

The benefit of program-order depth-first search is that the programmer can order the clauses in the program to maximize the efficiency of Prolog’s proof search algorithm.

Conclusions

Prolog proof search is an example of good engineering design based on an interplay between theoretical and practical considerations.

- It has a strong theoretical foundation, in being based on a complete inference system for definite-clause logic.

- However, its incomplete proof search procedure is a good implementation choice, allowing efficient proof search in the context of predicate logic, and permitting the programmer to tailor programs with regard to efficiency issues.

Main points today

- search tree for top-down proof search
- Prolog search strategy as program-order depth-first search
- incompleteness of Prolog proof search strategy
- correctness of Prolog proof search strategy (and proof)
- soundness of inference system (and proof)
- completeness of inference system (and proof)