Prolog Itself

- Based on Refutation Resolution which is be Sound, Complete

But...

- “Occurscheck”
  ⇒ not sound...

- Only deals with **Horn clauses**
  \( \leq 1 \) positive literal per clause
  ⇒ not completely expressive
  no REASONING by Cases)

- Fixed **Search Strategy**: SLD

- Beyond Logic
  - Negation as Failure: \(+/-\)
  - Search control: !
  - Operators: **assert**, **retract**, ...
Prolog is Not Sound: Occurscheck

- Prolog does **not** insist on legal substitutions for efficiency reasons.
  
  \[
  p(f(x), x) \text{ unifies w/ } P(Y, Y)
  \]
  
  “succeeds” with \( Y = f(f(f(f(f(f(\ldots\right)\]

- “OccursCheck”

- Why?
  
  Unification w/o OccursCheck is \( O(n) \)
  
  \((n = \text{size of clause})\)

  Unification w/ OccursCheck is \( O(n^2) \)

  But... ⇒ Too many clauses match
  ⇒ Too many conclusions reached
  ⇒ may conclude □ incorrectly
  ⇒ Not sound!

  (In some Prolog's, user can reset OccursCheck)
**Limitation of Horn Clauses**

A \[\text{Red}\]  
B  
C \[\text{Green}\]

(on A B)  
(on B C)  
(Red A)  
(Green C)  
\(\forall x. (\text{Red } x) \lor (\text{Green } x)\)

**Question:** \(\exists x,y. (\text{on } x \ y) \land (\text{Red } x) \land (\text{Green } y)\)

**Answer:** Yes

1. B is either Red or Green

2. If B is Red  
   then \((x/B \ y/C)\)

3. If B is Green  
   then \((x/A \ y/B)\)

- ...not in Prolog.  
\((\text{Cannot express } \forall x. (\text{Red } x) \lor (\text{Green } x))\)
How to find a Proof?

• Problem
  
  – Easy to **recognize** a proof...
    Hard to **generate** a proof.

  – Many potential proofs;
    Most “paths” fail.

  – Goal: How to find a proof, efficiently?

• Good news:
  
  ★ Only one inference Rule: Resolution
  ★ Only considering Horn clauses
    ⇒ many “heuristics” **complete**

• Prolog’s decisions...
General Situation

- Express $KB \cup \{\neg \sigma\}$, as set of clauses

- At any time, have “state”:

$$
\begin{array}{|c|}
\hline
f_1 & \cdots \lor A \lor \cdots \lor \neg B \lor \cdots \\
\hline
f_2 & \cdots \\
\hline
f_3 & \cdots \lor \neg A \lor \cdots \\
\hline
f_4 & \cdots \\
\hline
f_5 & \cdots \lor B \lor \cdots \\
\hline
f_6 & \cdots \\
\hline
f_7 & \cdots \lor \neg B \lor \cdots \lor Q \cdots \\
\hline
\end{array}
$$

- Should we smash...
  
  $f_1$ with $f_3$ ?
  
  $f_1$ with $f_5$ ?
  
  $f_5$ with $f_7$ ?
  
  $\ldots$ ??
**Prolog’s Decision**

Q1: Which two clauses?

A: *Set-Of-Support* (*Backward*)
  ⇒ One clause is query, or descendant

View these as Goals/subGoals

• At any time, “Frontier” of SubGoals:
  Which one to work on?

A: Depth-first

Q2: Within SubGoal, \( \mathcal{g} = \{\ell_1, \ldots, \ell_k\} \),
which literal to work on?

A: *Ordered Resolution*
  ⇒ just \( \ell_i \)

Q3: For given literal, \( \ell_j \) in given SubGoal \( \mathcal{g} \)
What to unify with?

A: Need to consider only \( \left\{ \begin{array}{l} \text{fact} \\ \text{head of rule} \end{array} \right\} \) in \( KB \)

• Which rule/fact?

A: Chronological
Search Strategies

• Trying to find 1 solution to query
  (or determine that there are none)

• Which **SubGoal** to work on next?
  (Ignore order of literals within subgoal
  Ignore which fact/rule to use, ...)

• **Def’n of Level:**
  
  – Initial Goal is at level 0.
  
  – Given (sub)goal at level \( i \),
    its subgoals are at level \( i + 1 \).
    (Whether by matching fact,
     or backward chaining using rule)
  
  (If each (sub)goal has 2 (sub)subgoals,
   then level \( j \) has \( 2^j \) members.)
Example of Breadth-First Search Tree
Search Strategy I: Breadth-First

- Generate all (sub)goals at level \( i \) before “reducing” to level \( i + 1 \).
  - Start with Goal, \( \gamma_0 \), at level 0.
  - Generate ALL of \( \gamma \)'s subgoals, \( \{\gamma^k_1\}_k \)
  - Generate ALL subgoals of each \( \gamma^k_1 \);
    producing \( \{\gamma^l_2\}_l \).
  - Generate ALL of \( \gamma^l_2 \)'s subgoals, \( \{\gamma^m_3\}_m \).
    ...

- Stop when subgoal is success,
  or when no more reductions possible.
Breadth-First Search: Summary

+ Advantages:
  If there is a proof, guaranteed to find it.
  (in fact: finds SHORTEST proof)

+ DisAdvantages:
  Needs to store entire level
  (grows exponentially)
Search Strategy II: Depth-First

- Reduce left-most branch
  - if success, then done
  - if \( \times \) then
    - backtrack and reduce next left-most

- Until all branches done.

Given Goal, \( \gamma_0 \), find first solution (returning bindings).

Let \( CPath \leftarrow < \gamma_0 > \).

L: If \( CPath=<> \) then FAIL. Otherwise:
  Let \( CNode \leftarrow \text{First} (CPath) \);
  Let \( CPath \leftarrow \text{Rest} (CPath) \).
  If \( CNode = \text{success} \), Return bindings.
  Otherwise, If \( CNode = \times \), then Go to L: .
  Otherwise, Let \( < \gamma_i > = \text{Reductions of} \ CNode \).
    Let \( CPath \leftarrow \text{Append}(<\gamma_i>, CPath) \)
    Go to L: .
Example of Depth-First
Depth-First Search: Summary

+ Advantages:
  Does NOT need to store entire level just current branch
  (grows one node at a time)

+ DisAdvantages:
  Does not find all proofs!
  (Recall Ancestry example)
Infinite Loop, Prolog Style

\[ KB_4 = \begin{cases} 
(1) & hb(X) :- \text{isa}(X, \text{human}). \\
(2) & \text{isa}(X, \text{human}) :- hb(X). \\
(3) & \text{isa}(j, \text{human}). 
\end{cases} \]
Which Rule/Fact?

• For each (sub)goal:
  Prolog always re-starts at beginning of DB
  [NOT where it left off.]
  uses *first* which unifies!

• To avoid Infinite Loop:
  Exchange (2) and (3)
Prolog’s Decisions

- To select subgoal:
  Depth First

- To select literal in subgoal:
  Left to Right (only $i = 1$)

- To find KB’s propositions
  Chronological (oldest first)
  (both atomic formulae and rules)

- Called SLD Resolution
  Linear resolution with
  Selection procedure for
  Definite clauses
  but in french

So: Prolog can miss some proofs,
but is relatively fast, and
requires relatively little storage