CMPS112: Introduction to Prolog

Christopher Schuster

CMPS112 Comparative Programming Languages, Winter 2016
University of California, Santa Cruz

January 14, 2016
Outline

1. Logic in Computer Science
   - Horn clauses

2. Prolog execution model
   - Unification and backtracking
   - A Meta-interpreter

3. References
First-order Logic: Example

Thor is a parent of Magni.  \( \text{Parent}(\text{THOR}, \text{MAGNI}) \)
Thor is a parent of Modi.  \( \text{Parent}(\text{THOR}, \text{MODI}) \)
Siblings have a common parent.  \( \forall x \forall y. (\exists p. \text{Parent}(p, x) \land \text{Parent}(p, y)) \rightarrow \text{Sibling}(x, y) \)

Q: Who is the parent of Modi?  \( \text{Parent}(X, \text{MODI}) \)
A: Thor.  \( \text{(Satisfiable with } X = \text{THOR}) \)

Q: Are Magni and Modi siblings?  \( \text{Sibling}(\text{MAGNI}, \text{MODI}) \)
A: Yes.  \( \text{(Tautology)} \)
A horn clause is a disjunction of literals with at most one non-negated literal.

\[ \forall x \forall y. (\exists p. Parent(p, x) \land Parent(p, y)) \rightarrow Sibling(x, y) \]

\[ \forall x \forall y. Sibling(x, y) \leftarrow (\exists p. Parent(p, x) \land Parent(p, y)) \]

\[ \forall x \forall y. Sibling(x, y) \lor \neg (\exists p. Parent(p, x) \land Parent(p, y)) \]

\[ \forall x \forall y. Sibling(x, y) \lor \forall p. \neg (Parent(p, x) \land Parent(p, y)) \]

\[ \forall x \forall y \forall p. Sibling(x, y) \lor \neg (Parent(p, x) \land Parent(p, y)) \]

\[ \forall x \forall y \forall p. Sibling(x, y) \lor \neg Parent(p, x) \lor \neg Parent(p, y) \]

\[ [Sibling(x, y), \neg Parent(p, x), \neg Parent(p, y)] \]

Multiple horn clauses can be used to derive new clauses until either a solution is found or the conjunction of horn clauses is unsatisfiable.
Unification and Resolution

(1) Thor is a parent of Magni. $[Par(\text{THOR, MAGNI})]$
(2) Thor is a parent of Modi. $[Par(\text{THOR, MODI})]$
(3) Siblings have a common parent. $[Sib(x, y), \neg Par(p, x), \neg Par(p, y)]$
(4) Are Magni and Modi siblings? $[\neg Sib(\text{MAGNI, MODI})]$

$\forall x \forall y. (\exists p. Par(p, x) \land Par(p, y)) \rightarrow Sib(x, y)$

$Sib(x, y) \leftarrow Par(p, x) \land Par(p, y)$ sib(X,Y) :- par(P,X), par(P,Y).

$Sib(x, y) \lor \neg Par(p, x) \lor \neg Par(p, y)$ $[Sib(x, y), \neg Par(p, x), \neg Par(p, y)]$

Unify (4) and (3) with $x = \text{MAGNI, } y = \text{MODI}$ and resolve...

(5) $[\neg Par(p, \text{MAGNI}), \neg Par(p, \text{MODI})]$

Unify (5) and (1) with $p = \text{THOR, MAGNI = MAGNI}$ and resolve...

(6) $[\neg Par(\text{THOR, MODI})]$

Unify (6) and (2) with $\text{THOR = THOR, MODI = MODI}$ and resolve...

(7) [ ] Empty disjunction (contradiction)
parent( thor, magni).
parent( thor, modi).
sibling(X,Y) :- parent(P,X), parent(P,Y).

?- visible(+all), trace, sibling(magni, modi).

Call: (7) sibling(magni, modi)
Unify: (7) sibling(magni, modi)
    Call: (8) parent(X, magni)
    Unify: (8) parent(thor, magni)
    Exit: (8) parent(thor, magni)

    Call: (8) parent(thor, modi)
    Unify: (8) parent(thor, modi)
    Exit: (8) parent(thor, modi)
Exit: (7) sibling(magni, modi)    yes.
Backtracking in Prolog

\begin{verbatim}
member(E,[E|_]).
member(E,[_|R]) :- member(E,R).
?- visible(+all), trace, member(X,[1,2]).
\end{verbatim}

Call: (7) member(X, [1, 2])
Unify: (7) member(1, [1, 2])
Exit: (7) member(1, [1, 2]) X = 1 ;
Redo: (7) member(X, [1, 2])
Unify: (7) member(X, [1, 2])
Call: (8) member(X, [2])
Unify: (8) member(2, [2])
Exit: (8) member(2, [2])
Exit: (7) member(2, [1, 2]) X = 2 ;
Redo: (8) member(X, [2])
Unify: (8) member(X, [2])
Call: (9) member(X, [])
Fail: (9) member(X, [])
Fail: (8) member(X, [2])
Fail: (7) member(X, [1, 2]) no.
\end{verbatim}
A Meta-interpreter

resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- clause(member(X,[1,2]),B).  X = 1,B = true ;
   B = member(X,[2]) ;
   no.
A Meta-interpreter

```prolog
resolve(true) :- !.
resolve((L,R)) :- !, resolve(L), resolve(R).
resolve(T) :- clause(T,Body),
            write('For '), write(T),
            write(' need to prove '),
            write(Body), nl,
            resolve(Body).

member(E,[E|_]) :- true.
member(E,[_|R]) :- member(E,R).

?- resolve(member(X,[1,2])).
For member(1,[1,2]) need to prove true  X = 1 ;
For member(X,[1,2]) need to prove member(X,[2])
For member(2,[2]) need to prove true  X = 2 ;
For member(X,[2]) need to prove member(X,[]).  no.
```
DID A PROLOG HOMEWORK QUESTION

TOOK LESS THAN 45 MINUTES PER LINE OF CODE
References

ISBN 9781846282126.

ISBN 9780262193382.
URL http://mitpress.mit.edu/books/art-prolog.

URL http://www.learnprolognow.org/.