

Horn Clauses: Programming in Prolog



Prof. Mauro Gaspari
Dipartimento di Informatica Scienze e Ingegneria
(DISI)

mauro.gaspari@unibo.it


Horn clauses in FOL



- FOL can be associated to sound and complete engines, but they can be only semi-decidable.
- Moreover, as for propositional logic, FOL engines are not efficient in the worst case and exponential time algorithms are required for reasoning in practice.
- Again, to improve efficiency we have to reduce the expressiv power.

Definite clauses in FOL



- Clauses in CNF form (disjunctions of literals) with exactly one positive literals (they are a subset of Horn clauses, goals are not considered). 
- We introduce them using the **Prolog syntax** which is based on the following rules:
 - Constants, Functional and Predicates symbols starts with lower case letters: arthur, even, king.
 - Variables symbols starts with uppercase letters: X, Y, N.
 - Implication is written as “:-”.
 - And is a comma: “,”. Or is a semicolon: “;”.
 - All the clauses end with a dot “.”.

Example



`evil(X):- king(X), greedy(x).`

`king(arthur).`

`greedy(y).`

Representing Knowledge with definite clauses



- The coronavirus special laws say that is a crime for Italian to leave their home city.
- John is an italian, he lives in Milan but now he moved to Puerto Escondido.

italian(john).

lives(john,milan).

moved(john,milan,puerto_escondido).

abroad(puerto_escondido).

abroad(london).

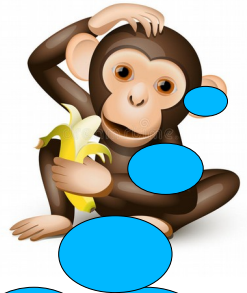
abroad(paris).

escaped(X):-lives(X,Y),moved(X,Y,Z),abroad(Z).

criminal(X):-italian(X),escaped(X).



SLD resolution in FOL



- SLD – resolution: Linear resolution with Selection function for Definite clauses.

$C_1 \wedge \dots \wedge C_m$ is the current goal.

1. Select one of the C_i for expansion.



2. Select a matching rule or fact

$C \leftarrow A_1 \wedge \dots \wedge A_n$ or C in the KB such that
 $\text{unify}(C_i, C) = \theta$

3. Set $(C_1 \wedge \dots \wedge C_{i-1} \wedge A_1 \wedge \dots \wedge A_n \wedge C_{i+1} \wedge \dots \wedge C_m)\theta$
or $(C_1 \wedge \dots \wedge C_{i-1} \wedge C_{i+1} \wedge \dots \wedge C_m)\theta$ as the new goal.

- Repeat this process until the current goal is \square .
- If we apply the composition of the substitutions $\theta^1, \dots, \theta^k$ generated during the reasoning process to the initial goal we find a counterexample: $(C_1 \wedge \dots \wedge C_m)\theta^1 \dots \theta^k$




$C_1 \wedge \dots \wedge C_m$
is represented as
positive but it is actually
a disjunction of literals



SLD resolution for definite clauses



- Soundness: IF $KB^D \cup \{\neg F\} \vdash^{SLD} \square$ THEN $KB^D \models F$
- Completeness: IF $KB^D \models F$ THEN $KB^D \cup \{\neg F\} \vdash^{SLD} \square$

- Semidecidable.
- Efficient: LINEAR with respect to the dimension of the KB.
- Expressive Power: GOOD

Implementing SLD resolution



- SLD resolution is a resolution strategy based on two selection rules:
 1. Selects one of the literals of the goal to reduce.
 2. Select a clause from the KB whose head unifies with the selected literal to resolve.
- SLD resolution does not specify how these two rules should be defined.
- For implementing SLD resolution we have to implement these two rules.

SLD resolution and Prolog



- Prolog implements the two selection rules as follows:
 1. The leftmost literal is selected for reduction.
 2. The first clause in the KB (program) whose head matches the selected literal is used for reduction.
- As a result Prolog explores the resolution search tree with a depth first search.
- If a failure occurs in the search process, Prolog backtrack to the next clause in the program that matches the selected literal.

Example



- Representing integers and the addition operation:
- Terms: 0 , $s(t)$
 $s(0), s(s(0)), s(s(s(0))) \dots$
- Addition:
 - a) $\text{plus}(0, X, X).$
 - b) $\text{plus}(s(X), Y, s(Z)) :- \text{plus}(X, Y, Z).$

Prolog search example



Goal: $\text{plus}(\text{s}(\text{s}(0)), \text{s}(0), W)$

Unifies with clause B

$\theta^1 = [X/\text{s}(0), Y/\text{s}(0), W/\text{s}(Z)]$

Every time that I use
a clause I should rename
its variables

Resolvent: $\text{plus}(X, Y, Z)\theta^1$

$\text{plus}(\text{s}(0), \text{s}(0), Z)$

Unifies with clause B $\theta^2 = [X1/0, Y1/\text{s}(0), Z/\text{s}(Z1)]$

Resolvent: $\text{plus}(X1, Y1, Z1)\theta^2$



$\text{plus}(0, \text{s}(0), Z1)$

Unifies with clause A $\theta^3 = [X2/\text{s}(0), Z1/X2]$



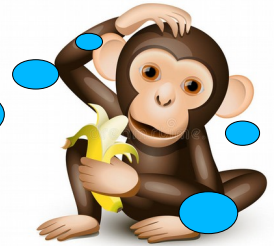
$W = \text{s}(Z) = \text{s}(\text{s}(Z1)) = \text{s}(\text{s}(X2)) = \text{s}(\text{s}(\text{s}(0)))$

Counterexample: $\text{plus}(\text{s}(\text{s}(0)), \text{s}(0), \text{s}(\text{s}(\text{s}(0))))$

A counterexample is
given by applying the
computed substitutions
to the variable of the
goal



Backtracking



This is the inverse!

Unifies with clause A $\theta^1 = [V/0, W/X, X/s(s(0))]$

Goal: $\text{plus}(V, W, s(s(0)))$

θ^1 is cancelled

unifies with clause B $\theta^{11} = [V/s(X), W/Y, Z/s(0)]$

$V=0, W=s(s(0))$



Resolvent: $\text{plus}(X, Y, Z)\theta^{11}$



$\text{plus}(X, Y, s(0))$ Unifies with clause A $\theta^2 = [X/0, Y/X1, X1/s(0)]$

θ^2 is cancelled

$V=s(0), W=s(0)$

unifies with clause B $\theta^{21} = [X/s(X1), Y/Y1, Z1/0]$

Resolvent: $\text{plus}(X1, Y1, Z1)\theta^{21}$

$\text{plus}(X1, Y1, 0)$

Unifies with clause A $\theta^3 = [X1/0, Y1/X2, X2/0]$

$V=s(s(0)), W=0$



Means backtracking when I try another clause the previous substitution is cancelled.

Observations



- The order of the clauses of a Prolog KB and the order of literals in the body of rules indicate how the resolution tree is explored.
- The rule order determines the order in which solutions are found.
- Moreover, if the order of the clauses in the Prolog KB is wrong the reasoning process could go forever without terminating.
- A Prolog programmer should deal with these issues.



Example



- Changing the order of clauses for changes the order of search, considering the following program:

```
parent(terach,abraham).  
parent (abraham,isaac).  
parent(isaac,jacob).  
parent(jacob,benjamin).  
ancestor(X, Y) :- parent (X, Y).  
ancestor(X, Z) :- parent (X, Y), ancestor(Y ,Z).
```

- For the query ?- ancestor(terach,X), the solutions will be given in the following order: X=abraham, X=isaac, X=jacob and X=benjamin .
- If the order of the two ancestor rules is swapped, the solutions will appear in a different order, namely X=benjamin, X=jacob , X=isaac and X=abraham.

Termination



- Consider the following program:

`married(X,Y):- married(Y , X).`

`married(abraham, sarah).`



- No computation involving `married` would ever terminate.
- Recursive rules which have the recursive goal as the first goal in the body are known as **left recursive rules**.
- The best solution to the problem of left recursion is avoidance:

`are_married(X , Y):- married(X, Y).`

`are_married(X, Y):- married(Y, X).`

Body literals order



- Goal order is more significant than clause order
- It is the principal means of specifying sequential flow of control in Prolog programs
- Changing the rules order does not change the search tree that must be traversed: The tree is just traversed in a different order
- Changing goal order changes the search tree.

Example



- Also literals order can determine whether a computation terminate. Consider the recursive rule for ancestor:

```
ancestor(X,Y):-  
    parent(X,Z), ancestor(Z,Y).
```



- If the goals in the body are swapped, the ancestor program becomes left recursive, and all Prolog computations with ancestor are nonterminating.

The Role of Unification



- Data manipulation in Prolog is achieved entirely via **unification**, that implements:
 - Single assignment
 - Parameter passing
 - Data structure creation.
 - Field-access in data structures.

Arithmetic

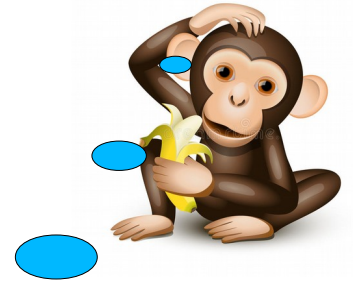


- The logic programs for performing arithmetic, like plus are very elegant, but they are not practical and not efficient.
- Prolog provides as specific operator:

X is Expression

- It is interpreted as follows:
 - Expression is evaluated as an arithmetic expression.
 - The result is unified with X.

Gcd in Prolog



$\text{gcd}(A, 0, Z) :- Z \text{ is } A.$

$\text{gcd}(A, B, Z) :- B > A, \text{gcd}(B, A, Z).$

$\text{gcd}(A, B, Z) :- X \text{ is } A \bmod B, \text{gcd}(B, X, Z).$




Control Predicates



- Prolog provides a single system predicate, called **cut**, denoted as **!**, for affecting the procedural behavior of programs. It can reduce the search space by dynamically pruning the search tree.
- There are two types of cut:
 - **Green cuts:** they can be used to express determinism, they improve the efficiency of programs pruning the search tree without changing their meaning.
 - **Red cuts:** they allow the programmer to omit explicit conditions, and thus they have an effect on the meaning of programs.

Green cuts



- This program `merge(Xs , Ys,Zs)` merges two sorted lists of numbers `Xs` and `Ys` into the combined sorted list `Zs`: 

```
merge([X | Xs],[Y|Ys],[X | Zs]):- X < Y, !, merge(Xs,[Y|Ys],Zs).
```

```
merge([X | Xs],[Y|Ys],[X ,Y|Zs]):- X = Y, !, merge(Xs,Ys,Zs).
```

```
merge([X | Xs],[Y|Ys],[Y|Zs]):- x > Y, !, merge([X | Xs], Ys,Zs).
```

```
merge(Xs,[ ] ,Xs).
```

```
merge([ ],Ys,Ys).
```



- The cut can be used to express the mutually exclusive nature of the tests. It is placed after the arithmetic tests and expresses that `merge` is deterministic

Cut semantics

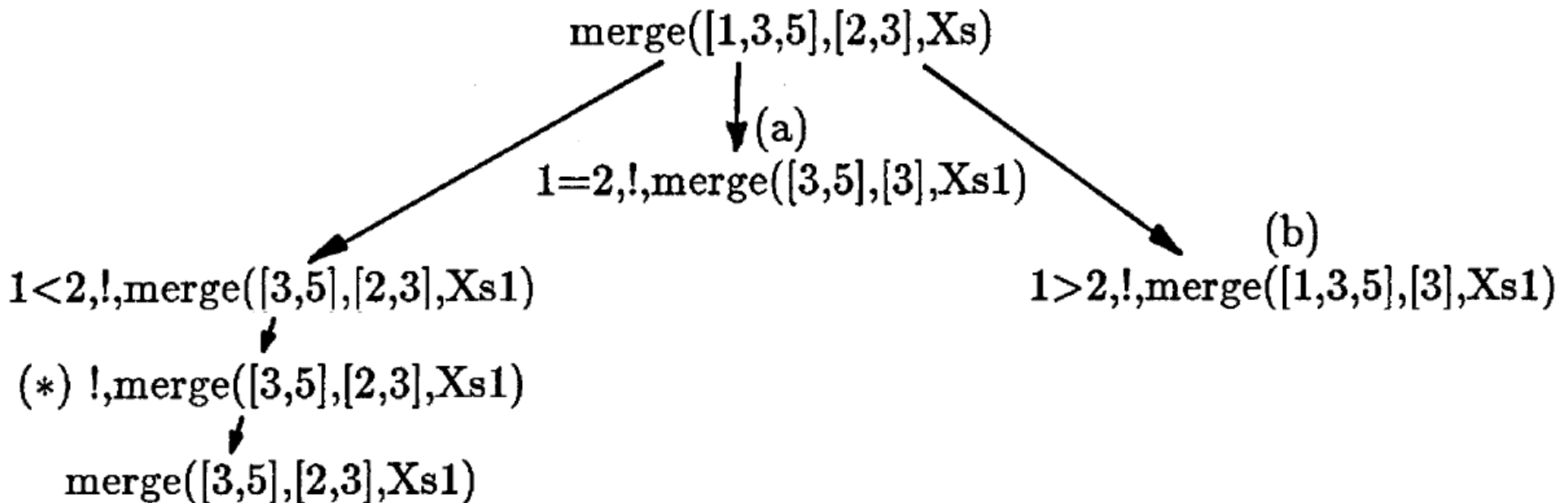


1. The cut (when is passed) prunes all clauses below it.
 2. The cut prunes all alternative solutions to the conjunction of goals which appear to its left in the clause: a conjunctive goal followed by a cut will produce at most one solution.
- The cut does not affect the goals to its right in the clause. They can produce more than one solution, in the event of backtracking.

Example



- After reducing to the conjunctive query $1 < 2, !, \text{merge}([3,5],[2,3],Xs1)$, the goal $1 < 2$ is successfully solved, reaching the node marked (*) in the search tree. The effect of executing the cut is to prune the branches marked (a) and (b).



Red cuts



- They prunes the search tree changing the meaning of programs.
- The cut is the basis of implementing a limited form of negation in Prolog called negation as (finite) failure:
 $\text{\textbackslash+ } X:- X, !, \text{fail.}$
 $\text{\textbackslash+ } X.$
- The semantics of the program is changed: this behaviour it is not possible without the cut operator.



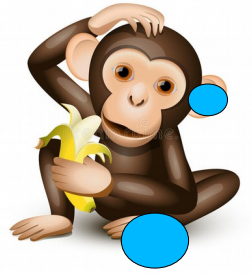
Programs as data



- The system predicate for accessing the program as data is `clause(Head, Body)` it must be called with `Head` instantiated and returns one after the other performing backtracking all the clauses that matches `Head`.
- The basic predicate for adding clauses is `assert(Clause)`, which adds `Clause` (which must be instantiated) as the last clause of the corresponding procedure. For asserting rules an extra level of brackets is needed for technical reasons concerning the precedence of terms. There is a variant of `assert`, `asserta`, that adds the clause at the beginning of a procedure.
- The predicate `retract(C)` removes from the program the first clause in the program unifying with `C`, `retractall(C)` removes all the predicates whose head unifies with `C`.



Failure driven loops



repeat.

repeat:- repeat.

Interactive interpreter:

Echo:- repeat, read(X), echo(X), !.

echo(end_of_file):- !.

echo(X):- write(X), nl, fail.

Consult loads a prolog file:

consult(File):- **see**(File), consultLoop, seen.

consultLoop:-repeat, read(Clause), process(Clause) , !.

process(end_of_file):- !.

process(Clause):- assert(Clause), fail.

repeat: always succeeds
fail: always fails
nl: prints a newline
see(File): open a file
seen: closes a file.

Tools and References



- **SWI Prolog** can be used with propositional Horn Clauses.
 - <https://www.swi-prolog.org/>
- **GNU Prolog** is a free Prolog compiler with constraint solving over finite domains developed by Daniel Diaz/
 - <http://www.gprolog.org/>
- **Sicstus Prolog**: an advanced commercial system.
 - <https://sicstus.sics.se/>
- The Art of Prolog:
 - <http://freecomputerbooks.com/The-Art-of-Prolog.html>