In the Lecture Series Logic Programming and Constraints

Selected Linear Definite Resolution

Presented by Stéphane Bressan

Syntax of Propositional Logic

Formulae in propositional logic are formed with propositions and connectives

- Given a set of propositions
- For the connectives {$\wedge$, $\vee$, $\Rightarrow$, $\neg$}
  - p is a formula if p is a proposition
  - (F1) $\wedge$ (F2) is a formula if F1 and F2 are formulae
  - (F1) $\vee$ (F2) is a formula if F1 and F2 are formulae
  - (F1) $\Rightarrow$ (F2) is a formula if F1 and F2 are formulae
  - $\neg$ (F) is a formula is F is formulae
  - And nothing else is a formula

Model Semantic of Propositional Logic

Semantics of connectives is given by truth tables

A	B	(A $\vee$ B)	(A $\wedge$ B)	(A $\Rightarrow$ B)	$\neg$ (A)
T	T	T	T	T	F
F	T	T	F	T	T
T	F	T	F	F	F
F	F	F	F	T	T

($\neg$ A $\vee$ B)

Model Semantic of Propositional Logic

- An ***interpretation*** of a formula is a mapping of each proposition in the formula to true (T) or false (F)

$$(p1 \vee p2) \wedge \neg (p1 \wedge p3)$$

I = {p1$\rightarrow$ T , p2 $\rightarrow$ F, p3 $\rightarrow$ F}

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

I is a ***model***.

Proof Semantics of Propositional Logic

Let us consider formulae composed of:

- propositions p,
- negation of propositions $\neg$ p and
- implications F1 $\vee$ F2

Let us consider the inference rule:

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

(all models of "F and F $\Rightarrow$ Q" are models of "Q")
(If "Q" has no model – is false – then "F and F $\Rightarrow$ Q" has no model)

Proof Semantics of Propositional Logic

Let us consider formulae composed of:

propositions p,
negation of propositions $\neg$ p and
implications F1 $\Rightarrow$ F2

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 l1 $\vee$ … $\vee$ ln
(l1 and l2 are either positive or negative literals)

Let us consider the inference rule:

if p $\vee$ F1 and $\neg$p $\vee$ F2 then F1 $\vee$ F2 (resolution)

Clauses and Horn Clauses

Formulae of the form l1 $\vee$ ... $\vee$ ln are called ***clauses***

The ***empty clause*** (sometimes written $\square$, $\perp$ or $\varnothing$) 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

$\forall X \exists Y (p(X) \Rightarrow (q(X, Y) \vee \neg r(X, Y)))$

Prolog programs are made of Horn Clauses

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

$\forall X \forall Y \forall Z$ (grand_parent(X, Y) $\Leftarrow$ (parent(X, Z) $\wedge$ parent(Z, Y)))

$\forall X \forall Y \forall Z$ (grand_parent(X, Y) $\vee \neg$ parent(X, Z) $\vee \neg$ parent(Z, Y))

Prolog programs are made of Horn Clauses

```
parent(X, Y):- father(X, Y).
parent(X, Y):- mother(X, Y).

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



father("Louis XVIII", "Dauphin Louis").
father("Maria LECZINSKA", "Stanislaw
    LECZINSKI").
father("Dauphin Louis", "Louis XV").
father("Louis XV", "Louis, Duke of
    Burgundy").

mother("Dauphin Louis", "Maria
    LECZINSKA").
mother("Louis XVIII", "Marie-Josephe").
mother("Louis XV", "Marie Adelaide").

:-grand_parent("Louis XVIII", Y).
```

```
∀X ∀Y (parent(X, Y) ∨ ¬ father(X, Y))
∀X ∀Y (parent(X, Y) ∨ ¬ mother(X, Y))

∀X ∀Y ∀Z (grand_parent(X, Y) ∨ ¬
    parent(X, Z) ∨ ¬ parent(Z, Y))



father("Louis XVIII", "Dauphin Louis").
father("Maria LECZINSKA", "Stanislaw
    LECZINSKI")
father("Dauphin Louis", "Louis XV")
father("Louis XV", "Louis, Duke of
    Burgundy")

mother("Dauphin Louis", "Maria
    LECZINSKA")
mother("Louis XVIII", "Marie-Josephe")
mother("Louis XV", "Marie Adelaide")

∀Y (¬ grand_parent("Louis XVIII", Y))
```


Selected Linear Definite (SLD) Resolution

:- grand_parent("Louis XVIII", Y)

$\forall Y$ ($\neg$ grand_parent("Louis XVIII", Y))

$\forall X \forall Y \forall Z$ (grand_parent(X, Y) $\vee \neg$ parent(X, Z) $\vee \neg$ parent(Z, Y))

$\forall Y \forall Z$ ($\neg$ parent(("Louis XVIII", , Z) $\vee \neg$ parent(Z, Y))

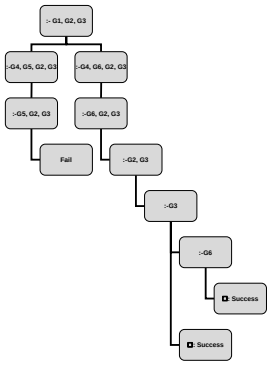
Which is the goal (involves unification – in this case only *filtering*)

:- parent(("Louis XVIII", Z), parent(Z, Y)).

SLD Tree

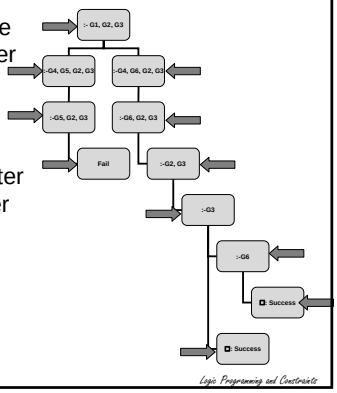
:- G1, G2, G3.

G1 :- G4, G5.
G1 :- G4, G6.
G2.
G3.
G3 :- 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 (*L*ast *I*n, *F*irst *O*ut) or FILO (*F*irst *I*n, *L*ast *O*ut).

The Cut!

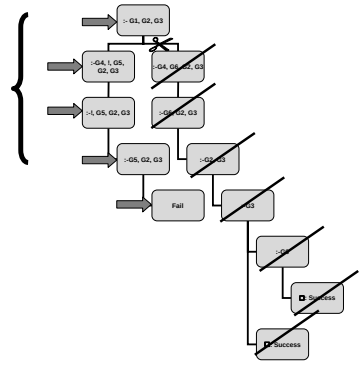
The special procedure !/0, called the ***cut***, always succeeds

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, G2, G3.

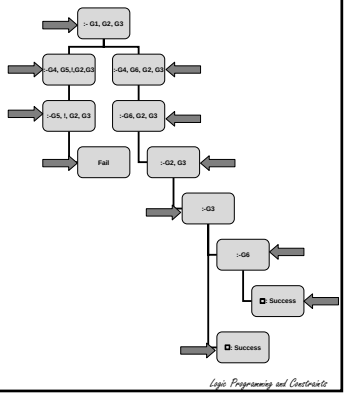
G1 :- G4, !, G5.
G1 :- G4, G6.
G2.
G3.
G3 :- G6.
G4.
G6.



The Cut!

:- G1, G2, G3.

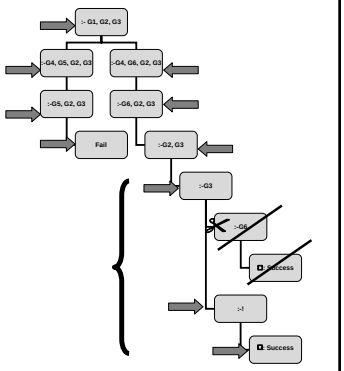
G1 :- G4, G5, !.
G1 :- G4, G6.
G2.
G3.
G3 :- G6.
G4.
G6.



The Cut!

:- G1, G2, G3.

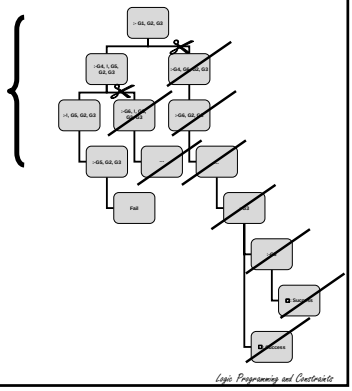
G1 :- G4, G5.
G1 :- G4, G6.
G2.
G3:-!.
G3 :- G6.
G4.
G6.



The Cut!

:- G1, G2, G3.

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



Negation as ! Failure

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

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

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


Negation as ! Failure

:- neg(grand_parent("Louis XVIII", "Dauphin Louis")).

:- neg(grand_parent("Louis XVIII", "Louis XV")).

:- neg(grand_parent("Louis XVIII", X)).

(we will see a sound negation when we study delayed goals)

:- neg(neg(grand_parent("Louis XVIII", X))).

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