A universal prover

Patrick Thévenon, PhD Student

Université de Savoie,Chambéry
Laboratoire de mathématiques,LAMA

Introduction

- Aim : having a prover able to deal with
  - any logic given (Universal prover)
  - Hints given to guide the proof
- Main features :
  - Functor
  - Inverse resolution
  - Clauses balanced by weights

The prover as a functor

```
module Prover : functor (Logic : Logic) ->

sig

Exception Prove_fails

val prove : ( formula * int * constraints) list
          -> formula
          -> unit
(* raises Prove_fails when no proof is found *)

end
```

To have a prover :

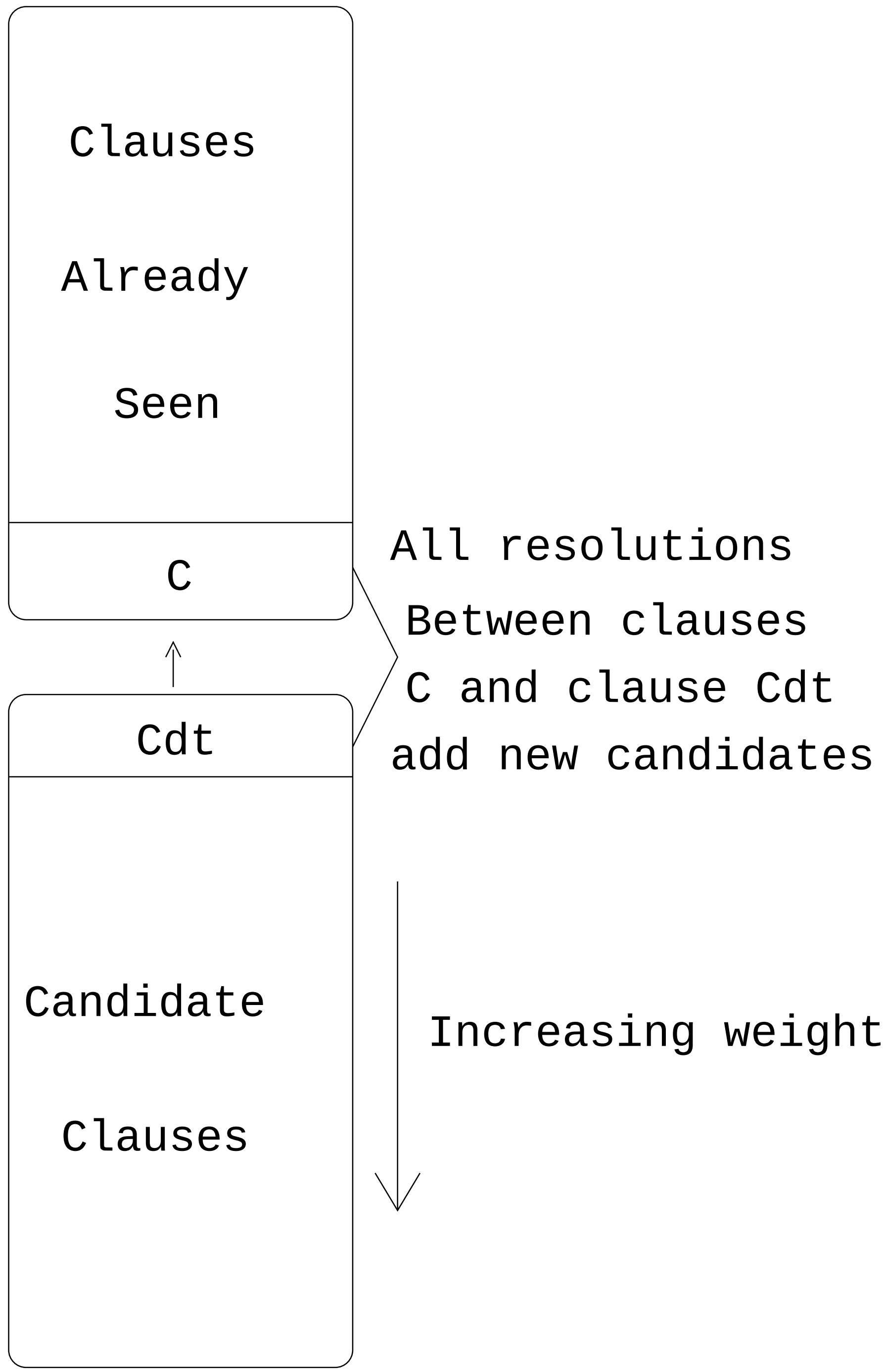
– give a logic

– apply the functor to it.

Logic required

```
module type Logic =

sig

type formula (form)

val elim_all_neg : form -> form

...

type substitution (subs)

type constraints (csts)

val unif : csts -> form -> csts -> form ->
    int * subs * csts * form * form list

val get_rules : csts -> form -> bool ->
    (string * int * subs * csts * form list ) list

end
```

Architecture



Decomposition and resolution

Inverse resolution : a clause is a set of literals, which are formulas that are not necessarily atomic.

Lazy decomposition : formulas are seen as black boxes, and decomposed only when a subformula can be unified with an other literal.

Decomposing formulas can be seen as making resolution with rule clauses...

...

Example :

Let $\{F^{\perp}, \Gamma\}$ be a clause with $F = (A \rightarrow B)$
From $F \leftrightarrow (A \rightarrow B)$ we obtain two clauses :

$\{A, \Gamma\}$ and $\{B^{\perp}, \Gamma\}$

It can be seen as resolutions with the following clauses on the literal $F \equiv X_1 \rightarrow X_2$:

$\{X_1, X_1 \rightarrow X_2\}$ and $\{X_2^{\perp}, X_1 \rightarrow X_2\}$

$\rightarrow$ Decomposing is making resolution with rule clauses.
$\rightarrow$ **get_rules** asks for each formula which rules can be applied.

The constraints

For each unification - applying rules or making resolutions - constraints may be given and used.

Examples :

1. **Skolemization** : Decomposing $\exists x.P(x)$, Make a clause with $P(x)$, x a new variable with a constraint saying that x depends only on the free variables of P. This avoids the use of the choice axiom for higher order logic.
2. **Contraction** : Contracting $C = \{A, A', \Gamma\}$, add a constraint $A \neq A'$ in C, to avoid possibly subsumption with clauses coming from $\{A\sigma, \Gamma\sigma\}$.
3. **Intuitionistic logic** ?
4. **Linear logic** ?

Dealing with Hints

H $:= A \wedge B$
H0 $:= A \rightarrow C$
H1 $:= D$
$\vdash C$
By H0 **and** H **trivial**.

Hypotheses have a lighter weight when they are named.

H $:= \forall x.B(x)$
H0 $:= C(x0)$
$\vdash \exists x.B(x) \wedge C(x)$
By H **with** $x = x0$, **by** H0 **trivial**.

The hint $x = x0$ can be given as a constraint. A unification that changes x into $x0$ has a lighter weight.

Conclusion

A very young prover that

- needs great improvements but
- may offer a good solution for the DemoNat project