

M2 PLS. Coq: \mathcal{L}_{tac} , extraction et preuves de programmes (notions)

Micaela Mayero

Université Paris 13,
LIPN UMR 7030-LCR team
<http://www-lipn.univ-paris13.fr/~mayero>

Mars 2010

1/18

Coq: \mathcal{L}_{tac} , extraction, PP

Le langage de tactiques \mathcal{L}_{tac}

Problématique

Combiner des tactiques

Le langage de tactique : \mathcal{L}_{tac}

L'extraction

Exemples

De la Preuve de programme

Logique de Hoare

Outils

Conclusion

Conclusion

2/18

Coq: \mathcal{L}_{tac} , extraction, PP

Automatiser ?

Exemple :

```
Goal (0<3)%R.
repeat
  (apply Rmult_lt_0_compat || apply Rplus_lt_pos;
   try apply Rlt_0_1 || apply Rlt_R0_R2).
```

- ▶ fastidieux
- ▶ inintéressant
- ▶ épouvantail à utilisateurs
- ▶ ...

Mieux :

```
Goal (0<3)%R.
prove_sup0.
```

3/18

Coq: \mathcal{L}_{tac} , extraction, PP

; || [[]] idtac repeat do try first ...

- ▶ **tac1;tac2** applique tac1 puis tac2 à tous les sous-buts issus de tac1
- ▶ **tac1||tac2** tac1 est essayée et si tac1 échoue alors tac2
- ▶ **tac1|[tac2|tac3]** applique tac1 puis tac2 au premier sous-but généré puis tac3 au deuxième ...
- ▶ **idtac** ne fait rien
- ▶ **repeat tac** applique tac jusqu'à échec (fail)
- ▶ **do num tac** applique num fois tac
- ▶ **try tac** rattrape l'exception levée par tac
- ▶ **first [tac1|tac2]** applique la première tactique qui marche
- ▶ ...

4/18

Coq: \mathcal{L}_{tac} , extraction, PP

Le langage de tactique : \mathcal{L}_{tac}

let

Local definitions

```

let ident1 := expr1
with ident2 := expr2
...
with identn := exprn in
expr

```

idem qu'en Ocaml.

5/18

Coq: \mathcal{L}_{tac} , extraction, PPLe langage de tactique : \mathcal{L}_{tac}

match expr

Filtrage d'une expression

```

match expr with
  cpattern1 => expr_1
| cpattern2 => expr_2
...
| cpatternn => expr_n
| _ => expr_{n+1}
end

```

idem qu'en Ocaml

6/18

Coq: \mathcal{L}_{tac} , extraction, PP

match goal

Filtrage des hypothèses et du but

```

match goal with
  | hyp_11,...,hyp_1m1 |- cpattern_1 => expr_1
  | hyp_21,...,hyp_2m2 |- cpattern_2 => expr_2
  ...
  | hyp_n1,...,hyp_nmn |- cpattern_n => expr_n
  | _ => expr_{n+1}
end

```

avant le `|-` , on filtre un ensemble d'hypothèses

après le `|-` , on filtre le but

après la `=>` , on applique une expression (qui peut être une/des tactique/s).

Le *match goal* fait un [backtracking](#).

7/18

Exemple

```

Ltac prove_sup0 :=
  match goal with
  | |- (0 < 1) => apply Rlt_0_1
  | |- (0 < ?X1) =>
    repeat
      (apply Rmult_lt_0_compat || apply Rplus_lt_pos;
       try apply Rlt_0_1 || apply Rlt_R0_R2)
  | |- (?X1 > 0) => change (0 < X1) in |- *; prove_sup0
  end.

```

8/18

Exercice

Ecrire une tactique qui remplace tous les $>$ par des $<$ (hyp et goal).
Open Local Scope R_scope.

```
Ltac toreplace_gt :=
match goal with
| H : (?X1 > ?X2) |- _ => change (X2 < X1) in H
| |- (?X1 > ?X2) => change (X2 < X1)
end.
```

```
Ltac replace_gtR := repeat toreplace_gtR.
```

9/18

Coq: \mathcal{L}_{tac} , extraction, PP

Rappels

- ▶ Curry-De Bruijn-Howard
- ▶ Obtenir du code prouvé
- ▶ Réalisabilité
- ▶ Les sortes (Prop/Set, les univers)
- ▶ Elimination forte

10/18

Coq: \mathcal{L}_{tac} , extraction, PP

Exemples

Le tri par insertion en Coq

```

Lemma sort : forall l, {m| sorted m /\ permut l m}.
induction l.
exists (nil (A:=A)); auto.
case IHl; intros l1 (Hsl1,Hpl1).
case (insertion a l1 Hsl1); intros l2 (Hsl2,Hpl2).
exists l2; split; auto.
apply permut_trans with (a::l1); auto.
Defined.

```

11/18

Coq: \mathcal{L}_{tac} , extraction, PP

Exemples

Le tri par insertion extrait en Ocaml

```

Extraction Sort.
Extraction "inssort.ml" Sort.

let rec insertion a = function
  | Nil -> Cons (a, Nil)
  | Cons (a0, l0) ->
      (match 0.le_total a a0 with
       | Inl _ -> Cons (a, (Cons (a0, l0)))
       | Inr _ -> Cons (a0, (insertion a l0)))

let rec sort = function
  | Nil -> Nil
  | Cons (a, l0) -> insertion a (sort l0)
end

```

12/18

Coq: \mathcal{L}_{tac} , extraction, PP

Exemples

Le tri par tas (heapsort)

```
Require Import Heap.
```

```
Check treesort.
```

```
treesort
```

```
: forall (A : Type) (leA eqA : Relation_Definitions.relation A)
  (forall x y : A, {leA x y} + {leA y x}) ->
  forall eqA_dec : forall x y : A, {eqA x y} + {~ eqA x y},
  (forall x y z : A, leA x y -> leA y z -> leA x z) ->
  forall l : List.list A,
  {m : List.list A | Sorting.sort leA m &
  Permutation.permutation eqA eqA_dec l m}
```

13/18

Coq: \mathcal{L}_{tac} , extraction, PP

Exemples

Le tri par tas extrait en Ocaml

```
Extraction "heapsort" treesort.
```

```
let rec list_to_heap leA_dec eqA_dec = function
  | Nil -> Tree_Leaf
  | Cons (a, l0) ->
      insert leA_dec eqA_dec (list_to_heap leA_dec eqA_dec l0) a
```

```
let heap_to_list leA_dec eqA_dec t =
  is_heap_rect Nil (fun a t1 t2 _ _ _ x _ x0 -> Cons (a,
    (merge leA_dec eqA_dec x x0))) t
```

```
let treesort leA_dec eqA_dec l =
  heap_to_list leA_dec eqA_dec (list_to_heap leA_dec eqA_dec l)
```

14/18

Coq: \mathcal{L}_{tac} , extraction, PP

Définitions

Triplet de Hoare : triplet noté $\{P\}i\{Q\}$ où P et Q sont des formules du premier ordre partageant leurs expressions avec le programme i .

Validité : le triplet $\{P\}i\{Q\}$ est valide si pour tous états E_1 et E_2 tq $E_1(P)$ vrai et $E_1 \xrightarrow{i} E_2$, alors $E_2(Q)$ vrai.

15/18

Coq: \mathcal{L}_{tac} , extraction, PP

Sur un petit programme : ISQRTn

```
count :=0;
sum := 1;
while sum <= n do
  count := count +1;
  sum := sum + 2 * count + 1
done
```

Montrons qu'à la fin du programme, count contient $\lfloor \sqrt{n} \rfloor$:

$$\{n \geq 0\} \text{ ISQRTn } \{count^2 \leq n < (count + 1)^2\}$$

16/18

Coq: \mathcal{L}_{tac} , extraction, PP

Dans la pratique

- ▶ annotations des programmes
- ▶ qui génèrent des obligations de preuves
- ▶ prouveurs automatiques (SMT) ou non
- ▶ langages de spécification Why, ACSL
- ▶ outils (Caduceus), Frama-C, Krakatoa

Coq: \mathcal{L}_{tac} , extraction, PP

- ▶ seulement un aperçu de Coq
- ▶ cf le manuel de référence
- ▶ formaliser et prouver : acquérir de l'expérience
- ▶ rendre les prouveurs attrayants
- ▶ ...