

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ○○○ ○○○ ○	○○ ○○ ○	○

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>

28 novembre 2014

1/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
● ○ ○○○○○	○○○ ○○○ ○○○ ○	○○ ○○ ○	○

Problématique

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.
```

3/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ○○○ ○○○ ○	○○ ○○ ○	○

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

Problématique
Combiner des tactiques
Le langage de tactique : \mathcal{L}_{tac}

L'extraction

Mécanisme
Exemples
Conclusion

De la Preuve de programme

Logique de Hoare
Outils

Conclusion

Conclusion

2/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ● ○○○○○	○○○ ○○○ ○○○ ○	○○ ○○ ○	○

Combiner des tactiques

; || [] 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
- ▶ ...

4/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ●○○○	○○○ ○○○ ○	○○ ○	○
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.

5/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○●○○	○○○ ○○○ ○	○○ ○	○
Le langage de tactique : \mathcal{L}_{tac}			

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](#).

7/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ●○○○	○○○ ○○○ ○	○○ ○	○
Le 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

6/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○●○	○○○ ○○○ ○	○○ ○	○
Le langage de tactique : \mathcal{L}_{tac}			

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.
```

8/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○●	○○○ ○○○ ○	○○ ○○ ○	○
Le langage de tactique : \mathcal{L}_{tac}			

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.
```

9/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ○○○ ○	○○ ○○ ○	○

Difficultés

But : produire des fonctions extraites à partir des fonctions Coq
 (modèle des langages fonctionnels)

- ▶ Langages de programmation : pas de types dépendants ;
- ▶ Fonctions bien typés dans les langages de programmation \neq fonctions bien typés dans le CoC ;
- ▶ CoC : produire des démonstrations, pas des calculs ;
- ▶ Efficacité ;
- ▶ Distinction entre aspects logiques (au moment de la compilation) et calculs sur les données (exécution).

(cf. Coq'Art)

11/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ○○○ ○	○○ ○○ ○	○

Rappels

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

10/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	●○○ ○○○ ○	○○ ○○ ○	○
Mécanisme			

Types et fonctions non polymorphes

Coq	Ocaml
<pre>Inductive positive : Set := xI : positive -> positive x0 : positive -> positive xH : positive.</pre>	<pre>type positive = XI of positive X0 of positive XH</pre>
<pre>Fixpoint Psucc (x:positive):= match x with XI x' => X0 (Psucc x') X0 x' => XI x' xH => x0 xH end.</pre>	<pre>let rec psucc=function XI x' -> X0 (psucc x') X0 x' -> XI x' XH -> X0 XH</pre>

12/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	●○○ ○○○ ○	○○ ○	○
Mécanisme			

Types et fonctions polymorphes (types dépendants)

Coq	Ocaml
Inductive list (A:Set) : Set := nil : list A cons : A -> list A -> list A.	type 'a coqlist = Nil Cons of 'a * 'a coqlist
Fixpoint app (A:Set) (l m: list A){struct l}:= match l with nil => m cons a l1 => cons A a (app A l1 m) end.	let rec app l= (fun m -> match l with Nil -> m Cons(a,l1)-> Cons(a,app(l1,m))

13/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ●○○ ○	○○ ○	○
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.

```

15/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○● ○○○ ○	○○ ○	○
Mécanisme			

Les preuves

Les propositions dans les types inductifs disparaissent :

Coq	Ocaml
Inductive sumbool (A B:Prop) : Set := left : A -> sumbool A B right : B -> sumbool A B.	type sumbool = Left Right

- ▶ La récursion bien fondée (pas ici);
- ▶ La dualité Prop/Set : seule l'existence des preuves compte, pas leur valeur;
- ▶ L'existentielle :

Coq	Extraction
{m:A & {n:B P}}	fonction qui calcule m et n t.q. P
{m:A Ex n:B P}	fonction qui calcule m t.q. $\exists n$ t.q. P
Ex m:A Ex n:B P	pas extraite

14/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
○ ○ ○○○○○	○○○ ●○○ ○	○○ ○	○
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

```

16/21

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

Le langage de tactiques \mathcal{L}_{tac} ○ ○ ○○○○○	L'extraction ○○○ ○○○ ●	De la Preuve de programme ○○ ○	Conclusion ○
Conclusion			

Utilisation

- ▶ Extraction "file.ml" f1 ... fn.
- ▶ Bien approprié pour des programmes effectuant des calculs symboliques.
- ▶ Moins adapté au calcul numérique : nombres représentés de manière symbolique \neq UAL du processeur.

17/21

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

Le langage de tactiques \mathcal{L}_{tac} ○ ○ ○○○○○	L'extraction ○○○ ○○○ ○	De la Preuve de programme ●○ ○	Conclusion ○
Logique de Hoare			

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 $\lfloor \sqrt{n} \rfloor$:

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

19/21

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

Le langage de tactiques \mathcal{L}_{tac} ○ ○ ○○○○○	L'extraction ○○○ ○○○ ○	De la Preuve de programme ●○ ○	Conclusion ○
Logique de Hoare			

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.

18/21

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

Le langage de tactiques \mathcal{L}_{tac} ○ ○ ○○○○○	L'extraction ○○○ ○○○ ○	De la Preuve de programme ○○ ●	Conclusion ○
Outils			

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

20/21

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

Le langage de tactiques \mathcal{L}_{tac}	L'extraction	De la Preuve de programme	Conclusion
<ul style="list-style-type: none"> o o ooooo 	<ul style="list-style-type: none"> ooo oo oo o 	<ul style="list-style-type: none"> oo o 	<ul style="list-style-type: none"> •
Conclusion			

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