Guide de survie
Les tactiques de base en Coq-ssr

première version

B. Werner – INF 565

16 janvier 2009

1 Résumé

Ce tableau donne les tactiques de base pour prouver un but de la forme $\wedge, \vee, \Rightarrow, \forall, \exists$ ou $=$, et respectivement pour utiliser une hypothèse `h` de cette forme.

Notation Math.	Notation Coq	prouver	utiliser
$A \wedge B$	`A /\ B`	`split`	`move : h => [a b]`
$A \vee B$	`A \/ B`	`left` ou `right`	`move : h => [a | b]`
$A \Rightarrow B$	`A -> B`	`move => a`	`apply h`
$\forall x, A$	`forall x, A`	`move => x`	`apply h`
$\exists x, A$	`exists x, A`	`exists t`	`move : h => [x a]`

$a = b$	`a = b`	`reflexivity`
		`trivial`
		`done`
			`rewrite h`
			`rewrite -h`

$\bot$	`False`		`done` `case h`

2 Tactiques automatiques

`trivial`

Cette tactique essaye de résoudre le but avec *un peu* de résolution automatique. Elle prend peu de temps. Elle n'échoue jamais : soit le but est résolu, soit il ne se passe rien.

`auto`

Comme trivial, mais un peu plus sophistiqué. Elle peu prendre du temps.

`done`

Une forme un peu amélioré de `trivial`. Elle détecte les contradictions comme `true = false`. `En revanche, elle échoue si elle ne résoud pas le but.` Cette dernière caractéristique sert à structurer les scripts de preuve.

2.1 `by ...`

Le texte `by` t1 ; t2 ; t3. où t1, t2, t3 sont des tactiques est équivalents à t1 ; t2 ; t3 ; `done`.

3 Égalité : `rewrite`

Si `h` est une preuve de `a=b` alors `rewrite h` remplace tous les `a` du but par `b`. On fait le contraire avec `rewrite -h`. On réécrire avec plusieurs égalités de suite : `rewrite h1 h2 -h3` par exemple.

Variantes : `rewrite !h` va réécrire avec `h` autant de fois que possible. `rewrite ?h` va essayer de réécrire avec `h` mais n'échouera pas si ce n'est pas possible (équivalent à `try rewrite h`).

On peut également insérer des instructions spéciales :

- `rewrite //` essaye un `done`.
- `rewrite /=` essaye de simplifier par calcul.
- `rewrite /=` simplifie par calcul puis essaye `done`.

4 Contrôle

Certaines tactiques échoue et interrompent le script. On peur jouer là-dessus à l'aide des tacticiels suivant :

`try t`

Va exécuter `t`. Si elle échoue, l'erruer est ratrappée et le but inchangé.

`fail`

Échoue toujours.

`idtac`

Ne fait rien.

`by` et `done`

Voir ci-dessus.

5 Récurrence

Une récurrence sur `n` se fait par `elim n`. Sur le premier objet du but par `elim`. Le plus souvent on continuera directement par `=>`. Par exemple :

```
elim : n => [| n Hn].
```

6 Comprendre `move`

On peut mieux comprendre `move` en pensant que :

- une preuve de $A \wedge B$ est une paire composée d'une preuve de A et d'une preuve de B. `move => [a b]` permet de décomposer la paire.
- une preuve de $A \vee B$ est soit une preuve de A soit une preuve de B. `move => [a|b]` permet de raisonner par cas ; soit A soit B.
- une preuve de $\exists x, A$ est aussi une paire formée par un objet `t` et la preuve que `t` vérifie A. On décompose la paire par `move => [t a]`.