Devoir Maison

Michel Lévy

20 avril 2009

1 Objectifs

Le but de ce devoir, est essentiellement de vous faire connaître et de vous inciter à utiliser deux logiciels dont la base théorique est, très partiellement, dans le cours.
Ces logiciels sont disponibles et documentés à l'adresse : `http://www.cs.unm.edu/~mccune/mace4/`
Ces deux logiciels sont appelés `prover9` et `mace4`. Ils sont appelés via la commande `prover9-mace4`, qui ouvre une fenêtre avec deux zones, la zone des hypothèses (assumptions) où vous pouvez entrer une liste de formules, la zone des buts (goals) où l'on ne doit écrire qu'une formule, *le but*.
Si vous écrivez plusieurs buts, seul le premier sera pris en compte.

- `prover9` tente de prouver le but à partir des hypothèses. Il transforme la conjonction des hypothèses et de la négation du but en une forme clausale et montre (si c'est le cas) que cette liste est insatisfaisable en en donnant une preuve par le système de la résolution.
- `mace4` cherche une interprétation finie qui est modèle des hypothèses et contre-modèle du but.

En général, on commence par exécuter `prover9` et si le prouveur a échoué, on appelle `mace4` sur les mêmes données : si `mace4` a construit une interprétation, on sait la raison de l'échec du prouveur, le but n'est pas conséquence des hypothèses.

2 Mode d'emploi abrégé de `prover9-mace4`

On donne des indications brèves sur la syntaxe des formules et le mode d'emploi de `prover9` et de `mace4`. Pour obtenir des informations complètes, on utilisera le manuel à l'adresse :
`http://www.cs.unm.edu/~mccune/prover9/manual/2009-02A/`.

2.1 Syntaxe des formules

Toute formule est terminée par un point comme sur l'exemple : `all x (C(x) & O(x) -> D(x)).`
La syntaxe des formules diffère de celle du cours. On précise comment diminuer cette différence en modifiant les types et les précédences des opérations.

2.1.1 Variables et constantes

Dans toute formule, un identificateur est une variable si et seulement si il est *lié* par un quantificateur ou s'il commence par une des lettres "u, v, w, x, y, z".
Une formule suivie d'un point est (implicitement) universellement quantifiée, ainsi on peut exprimer que tous les hommes sont mortels soit par `all x (H(x) -> M(x)).`, soit par `H(x) -> M(x).` : dans le deuxième cas, la formule est identifiée avec sa fermeture universelle.

2.1.2 Symboles logiques

Les symboles logiques du logiciel sont `$T,$F, -, &, |, ->, <-> , all, exists,=, !=`. Il est possible de les changer, mais nous avons choisi de ne pas le faire.
La conjonction et la disjonction sont notées respectivement par `&, |`.
Les déclarations par défaut des connectives logiques ne sont pas conformes à celles du cours, aussi on suggère de mettre dans la fenêtre `Language Options` les commandes suivantes :

```
op(750,prefix,"-").
op(795,infix_right,"->").
op(805,infix_left,"<->").
```

Le premier argument de la commande `op` est la *précédence* de l'opération qui est en le troisième argument. Plus la précédence est forte, plus la priorité est faible.
La priorité de la négation devient ainsi égale à celle des quantificateurs. L'implication devient associative à droite, grâce à son type `infix_right`. L'équivalence a une priorité inférieure à celle de l'implication et est associative à gauche, grâce à son type `infix_left`.

2.2 Le prouveur `prover9`

2.2.1 Activer `prover9`

Pour montrer que Socrate est mortel est une conséquence de ce que tout homme est mortel et que Socrate est un homme, on se donne les hypothèses :

```
% Tous les hommes sont mortels
all x (H(x) -> M(x)).
% Socrate est un homme
H(socrate).
```

Et le but :

```
% Socrate est mortel.
M(socrate).
```

Les lignes commençant par % sont des commentaires. En appuyant sur le bouton `start`, la preuve est produite.

2.2.2 Diriger `prover9`

On peut diriger le fonctionnement du prouveur au moyen de nombreuses options. Par exemple, on peut limiter le temps de la recherche d'une preuve à 10 secondes (au lieu de 60 secondes par défaut).

Si vous souhaitez que le prouveur n'utilise que les règles de la résolution vue en cours, il faut cocher parmi les `Meta Options` la case `raw` puis cocher les règles d'inférences (`Inference Rules`) `binary_resolution` et `factor` (attention cette case apparait sous la mention `Other Rules`).

L'interêt de ce choix restreint de règles est de donner des preuves presque conformes à celles étudiés en cours et de montrer que le cours n'a couvert qu'une faible partie des techniques mises en oeuvre dans la démonstration automatique.

Par contre dès que vous voulez utiliser l'égalité ou aborder des problèmes difficiles, on vous conseille d'utiliser les options par défaut. Et si la démonstration automatique vous interesse, vous pouvez explorer les multiples options du prouveur.

2.3 Le constructeur de modèle `mace4`

2.3.1 Rôle de `mace4`

Le constructeur de modèle `mace4` recherche une interprétation finie modèle des hypothèses et contre-modèle de la conclusion en *commençant* avec un domaine, qui, dans les options par défaut, a au moins deux éléments.

On peut modifier la taille minimum du domaine en changeant la valeur de `start_size` et sa taille maximum en cheangeant la valeur de `end_size`.

Par exemple avec l'hypothèse :
`all x all y exists z P(x,y, z).`
et la conclusion :
`all x P(x,y, f(x, y)).`
`mace4` trouve une interprétation à 2 éléments, 0 et 1.

La sortie par défaut n'est pas évidente à lire. On trouve :
`function(f(_,_), [0,0,0,0])` ce qui veut dire que :
`f(0,0)=0. f(0,1)=0.f(1,0)=0.f(1,1)=0.`
On trouve aussi :
`relation(P(_,_,_), [0,1,1,0,1,0,1,0])])` ce qui veut dire que :
`P(0,0,0)=0.P(0,0,1)=1.P(0,1,0)=1.P(0,1,1)=0. P(1,0,0)=1.P(1,0,1)=0. P(1,1,0)=1.P(1,1,1)=0.`

On recommande de reformater la sortie par défaut avec l'option `cooked` qui donne une présentation plus claire de l'interprétation, avec sur cet exemple :
`f(0,0)=0. f(0,1)=0.f(1,0)=0.f(1,1)=0.`
`-P(0,0,0).P(0,0,1).P(0,1,0).-P(0,1,1). P(1,0,0).-P(1,0,1). P(1,1,0).-P(1,1,1).`
On peut constater que le format de sortie des formules suit la syntaxe du logiciel.

2.3.2 L'égalité dans `mace4`

Les entiers peuvent être mis dans les formules et sont considérés comme des constantes ayant comme valeur elles-mêmes, comme il a été indiqué dans le cours.
Le logiciel interpréte le symbole = comme l'identité sur les entiers du domaine et le symbole `!=` comme la non identité sur les entiers.
Prenons comme hypothèse la formule :
`c!=0 & c!=1 & c!=2..`

L'appel de `mace4` termine par une erreur : en effet `mace4` commence par rechercher un modèle à deux éléments $0, 1$ et ne peut pas donner la valeur 2 à l'entier 2 sur ce domaine.

Avec un domaine de taille au moins 3, obtenu en donnant la valeur 3 à `start_size`, `mace4` trouve un modèle à quatre éléments 0, 1, 2, 3 avec `c=3`.

Avec un domaine de taille 3, obtenu en donnant la valeur 3 à `start_size` et à `end_size`, `mace4` termine *sans erreur* en ne trouvant pas de modèle.

3 Travail à faire

On fournira un compte-rendu suivant les indications données dans chaque exercice. Ce compte-rendu sera de préférence un fichier texte (par exemple éditer Cependant, on demande de la clarté dans la présentation, en particulier il ne faut pas oublier de rappeler au moins le numéro des exercices auxquels on répond.

Exercice 1 (Débuter avec prover9) *Soit la formule :*
`exists x (p(x) | q(x)) <-> exists x p(x) | exists x q(x).`
Prouver qu'elle est valide en la donnant comme but à `prover9`.
Dans le compte-rendu, reporter la forme clausale obtenue et la preuve. □

Exercice 2 (Débuter avec prover9 et mace4) *Soit la formule :*
`exists x (p(x) & q(x)) <-> exists x p(x) & exists x q(x).`
Vérifier tout d'abord que `prover9` n'arrive pas à montrer que cette formule est valide. puis construire un contre-modèle de cette formule grâce à `mace4`.
Dans le compte-rendu, reporter le modèle construit sous une forme compréhensible.
Indiquer dans le compte-rendu si `exists x (p(x) & q(x))` est conséquence de `exists x p(x) & exists x q(x)` ou si `exists x p(x) & exists x q(x)` est conséquence de `exists x (p(x) & q(x))`.
Justifier votre réponse, sans vous fatiguer, c'est-à-dire en utilisant le logiciel. □

Exercice 3 (Raisonner avec prover9) *On se propose de vérifier la correction du raisonnement suivant :*

1. *il y a un malade aimant tous les docteurs*
2. *aucun malade n'aime les charlatans*
3. *Donc aucun docteur n'est un charlatan*

Formaliser ce raisonnement. Dans les formules, on emploiera la «signature» suivante :
M(x) = x est malade
D(x) = x est un docteur
C(x) = x est un charlatan
A(x,y)= x aime y
Prouver qu'il est correct grâce à prover9 . Donner au compte-rendu la formalisation du raisonnement, la forme clausale calculée par prover9 ainsi que la preuve trouvée par prover9. □

Exercice 4 (Equivalence ou non équivalence) *Le professeur et un étudiant ont proposé deux traductions en formules de la phrase «Les chiens et les oiseaux sont des animaux domestiques» :*
`all x (C(x) | O(x) -> D(x))` *est la formulation du professeur*
`all x all y (C(x) & O(y) -> D(x) & D(y))` *est la formulation de l'élève*
Ces deux formulations ne sont pas équivalentes. Le vérifier à l'aide de `mace4` *et donner l'interprétation modèle q'une formule et pas de l'autre.*
Une des formules a pour conséquence l'autre, préciser, en utilisant `prover9`*, quelle formule a pour conséquence l'autre : indiquez la preuve obtenue.* □

Exercice 5 (Equivalence ou non équivalence) *Le professeur et un étudiant ont proposé deux traductions en formules de la phrase «Personne n'a appelé tous ceux qui ont réussi à l'examen» :*
`- exists x all y (P(y) -> Q(x,y))` *est la formulation du professeur*
`all y - exists x (P(y) -> Q(x,y))` *est la formulation de l'élève*
Ces deux formulations ne sont pas équivalentes. Le vérifier à l'aide de `mace4` *et donner l'interprétation modèle q'une formule et pas de l'autre.*
Une des formules a pour conséquence l'autre, préciser, en utilisant `prover9`*, quelle formule a pour conséquence l'autre : indiquez la preuve obtenue.* □

Exercice 6 (Trouver des modèles, Prouver) *Trouver un modèle (grâce à mace4) de l'ensemble de formules :*
```
all x exists y (x<y).
all x all y (x<y -> -(y<x)).
all x all y (x<y -> (exists z (x<z & z<y))).
```
Donner dans le compte-rendu le modèle obtenu, en le dessinant avec une flêche entre un élément ***a*** *et un élément* ***b*** *tels que* ***a < b****.*
Montrer (grâce à prover9) que de ces formules, on peut déduire :
`all x -(x <x).`
Donner dans le compte-rendu la preuve obtenue avec les options par défaut, puis en se restreignant aux règles du cours. On verra en 2.2.2 comment obtenir cette restriction. □

Exercice 7 (Ensembles et Preuve) *On dit que deux ensembles sont égaux si et seulement si tout élément de l'un est élément de l'autre. Cette définition de l'égalité est formalisée par l'hypothèse :*
`all x all y (x = y <-> all z (element(z,x) <-> element(z,y)))`
*On dit qu'**un** ensemble est sous-ensemble d'un **autre** ensemble si et seulement si tout élément de l'**un** est élément de l'**autre**. Formaliser cette définition en notant par la formule* `sousens(x,y)` *le fait que x est sous-ensemble de y.*

1. *Montrez, grâce à prover9, que ces deux définitions impliquent que la relation sousens est une relation d'ordre. Joindre au compte-rendu la formalisation du problème et les preuves obtenues.*
2. *Montrez, grâce à mace4, que la relation sousens n'est pas une relation d'équivalence. Indiquez le but fourni à mace4 et le contre-modèle (à deux élément 0 et 1) de ce but. Un de ces deux éléments est-il l'ensemble vide ?*

□

Exercice 8 (Ensembles et Preuve) *Cet exercice fait suite au précédent : on prend comme hypothèse la définition de l'égalité de deux ensembles. L'union des deux ensembles est définie par la formule :*
`all x all y all z (element(z,union(x,y)) <-> element(z,x) | element(z,y)).`
En suivant ce modèle, définir `intersection(x,y)` *l'intersection des deux ensembles* `x` *et* `y`*.*
De même définir `c(y)`*, le complément de l'ensemble* `y`*.*
On définit `ev`*, l'ensemble vide, comme étant l'ensemble qui n'a pas d'élément et* `d`*, comme le complémentaire de l'ensemble vide.*
Explicitez ces définitions montrer (grâce à `prover9` *que la théorie ci-dessus est une algèbre de boole, c'est à dire que* `ev` *est élément neutre de l'union, que l'union est idempotente, commutative et associative, que* `d` *est élément neutre de l'intersection, que l'intersection est idempotents, commutative et associative et que les lois de Morgan sont vérifiées.*
Dans le compte-rendu, on donnera les hypothèses et la liste des buts prouvés. □

Exercice 9 (Recurrence) *Dans cet exercice, on étudie les listes et les opérations sur les listes. On va utiliser la notation des listes fourni dans le logiciel. Cette notation est la notation du langage Prolog, ainsi qu'il est précisé dans le manuel officiel du logiciel :*

Notation Ocaml	*Notation Prolog*	*Notation mace 4*
$[]$	$[]$	`$nil`
$[a;b;c]$	$[a,b,c]$	`$cons(a,$cons(b,$cons(c,$nil)))`
$pr :: fin$	$[pr : fin]$	`$cons(a,b)`

Dans cet exercice, on explore la concaténation des listes, notée @ comme en Ocaml, et le retournement des listes. La concaténation est définie par les hypothèses

```
(1) all l ([] @ l =l).
(2) all x all l all m ([x :l] @ m = [x :(l @ m)]).
```

1. *Montrer grâce à prover9, que la formule*
   `[1,2]@[3,4]=[1,2,3,4]` *est conséquence des formules ci-dessus.*
   Autrement dit on peut faire des concaténations au moyen de preuves.
   Joindre au compte-rendu la preuve produite par prover9.
2. *Montrer, grâce à mace4, que de la définition de la concaténation, on ne peut pas déduire les propriétés «triviales» suivantes :*
   `all l (l @ [] = l)`
   `all l all x all y l @ [x :y] = (l @ [x])@ yall l1 all l2 all l3 (l1 @ (l2 @ l3) = (l1 @ l2) @ l3)`
   Donnez l'interprétation , obtenue par `mace4`*, modèle de la définition de la concaténation et contre-modèle de* `all l (l @ [] = l)`*. Attention* `mace4` *utilise les notations des listes de* `mace4`*.*
   Puisque ces trois formules, ne peuvent pas être déduites de la seule «définition» de la concaténation, il faut raisonner par récurrence sur les listes et c'est ce que nous faisons ci-dessous.
3. *Preuve par récurrence de* `all l (l @ [] = l)`
   On définit la propriété P ci-dessous et la récurrence associée :
   `all l (P(l) <-> (l @ [] = l))`
   `P([]) & all l all x (P(l) -> P([x :l])) -> all l P(l)`
   Prouver à l'aide de prouver9 que de la définition de @, P et du principe de récurrence sur P, on peut déduire :
   `all l P(l)`
   Joindre au compte-rendu la preuve obtenue.
4. *Preuve par récurrence* `all l1 all l2 all l3 (l1 @ (l2 @ l3) = (l1 @ l2) @ l3)`
   Un peu de reflexion, c'est-à-dire une preuve informelle, montre que cette récurrence doit être faite sur `l1`*.*
   Aussi on ajoute la propriété Q ci-dessous et le principe de récurrence sur Q :
   `all l1 (Q(l1) <-> all l2 all l3 (l1 @ (l2 @ l3) = (l1 @ l2) @ l3)`

`Q([]) & all l all x (Q(l) -> Q([x :l])) -> all l Q(l)`
Prouver `all l Q(l)` *et joindre au compte-rendu la preuve obtenue.*

5. *Prouver par récurrence* `all l all x all y l @ [x :y] = (l @ [x])@ y`
6. *On étudie comme retourner les listes de deux façons différentes et on prouve que ces deux façons donnent le même résultat.*

   (a) *retournement directe d'une liste, on pose :*
   ```
   rev1([]) = []
   all x all y rev1([x :y])= rev1(y)@[x]
   ```

   (b) *retournement d'une liste par accumulation des éléments retournés, on pose :*
   ```
   all a rev2([],a) = a
   all x all y all a rev2([x :y],a) = rev2(y,[x :a])
   all x rev3(x)= rev2(x,[])
   ```

   Notre but est de prouver que `rev1 = rev3`*. Mais le prouveur ne sait pas comment faire, il faut que l'humain lui indique ce qu'il faut prouver par récurrence :*
   établir `rev2(x,y)= rev1(x)@ y` *par récurrence sur* `x`*.*
   De plus, en faisant une preuve informelle, on voit qu'il faut admettre (ou reprouver) les propriétés :
   ```
   all l (l @ [] = l)
   all l all x all y l @ [x :y] = (l @ [x])@ y
   ```
   Formaliser ce travail avec le logiciel et joindre la preuve (ou les preuves) permettant d'établir que :
   ```
   all x rev1(x) = rev3(x)
   ```

□

Conclusion

Le démonstrateur et le constructeur de modèles fonctionnent conjointement. Lorsque le démonstrateur n'arrive pas à déduire un but des hypothèses, le constructeur de modèle nous aide à contruire une interprétation finie, modèle des hypothèses et contre-modèle du but.

Les démonstrateurs automatiques servent, dans ce travail, d'assistants de preuves. Les résultats à prouver sont découpés en proprités simples, que le démonstrateur peut prouver. Il faut notamment lui expliciter les récurrences à faire.