

Devoir Maison

Michel Lévy

26 avril 2008

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 dans le cours :

- prover9 a pour donnée une liste de formules. Il transforme cette liste 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 a pour donnée une liste de formules. Lorsque cette liste est satisfaisable dans un domaine *fini*, il en construit un modèle.

Travail à faire

Lire le manuel (voir page 4) qui est joint à ce devoir, puis faire les exercices.

Fournir un compte-rendu suivant les indications données dans chaque exercice. Pour faciliter ce travail, il est préférable de faire ce compte-rendu avec nedit, en même temps que l'on utilise les logiciels prover9 et mace.

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 :

$\text{exists } x (p(x) \mid q(x)) \leftrightarrow \text{exists } x p(x) \mid \text{exists } x q(x)$

Prouver qu'elle est valide, en montrant grâce à prover9, que sa négation est insatisfaisable.

Dans le compte-rendu, reporter la forme clausale obtenue et la preuve. □

Exercice 2 (Débuter avec prover9 et mace4) Soit la formule :

$\text{exists } x (p(x) \ \& \ q(x)) \leftrightarrow \text{exists } x p(x) \ \& \ \text{exists } x q(x)$

Vérifier tout d'abord que prover9 n'arrive pas à déduire faux de sa négation, puis construire un modèle de cette négation grâce à mace4.

Dans le compte-rendu, reporter la forme clausale obtenue et le modèle construit. □

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 (Trouver des modèles, Prouver) Trouver un modèle (grâce à mace4) de l'ensemble de formules : $\text{all } x \text{ exists } y (x < y)$

$\text{all } x \text{ all } y (x < y \rightarrow \neg(y < x))$

$\text{all } x \text{ all } y (x < y \rightarrow (\text{exists } z (x < z \ \& \ z < y)))$

Appelons Γ cet ensemble de formules.

Donner dans le compte-rendu le modèle obtenu.

Montrer (grâce à prover9) que de ces formules, on peut déduire :

$\text{all } x \neg(x < x)$.

Donner dans le compte-rendu la preuve obtenue.

La propriété ci-dessus indique que la relation inférieure est irreflexive.

Montrer (grâce à mace4) que la relation inférieure n'est pas un ordre strict, autrement dit, étant donné qu'un ordre strict est une relation irreflexive et transitive, montrer, grâce à mace4, que de Γ , on ne peut pas déduire la transitivité de la relation inférieure.

Dans le compte-rendu, on indiquera comment vous avez utilisé mace4 pour répondre à cette dernière question. \square

Exercice 5 (Ensembles et Preuve) 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. Montrez, grâce à prover9 que cette définition implique que la relation sous-ensemble est transitive.

On formalisera la définition et la propriété ci-dessus, grâce à la «signature» suivante :

$\text{element}(x,y) = x$ est élément de y

$\text{sousens}(x,y) = x$ est sous-ensemble de y

Joindre au compte-rendu la formalisation du problème et la preuve obtenue. \square

Exercice 6 (Recurrence) Dans cet exercice, on explore l'arithmétique de Péano. Dans cette arithmétique, l'addition est définie par les deux formules :

(1) $\text{all } n (n + 0 = n)$

(2) $\text{all } n \text{ all } p (n + s(p) = s(n+p))$

Puisque cette théorie utilise l'égalité, il est indispensable d'utiliser prover9 dans le mode automatique (autrement dit sans la commande `set(raw)`).

1. Montrer grâce à prover9, que la formule

$s(s(0))+s(s(0))=s(s(s(s(0))))$ est conséquence des formules (1) et (2).

Autrement dit on peut faire des additions au moyen de preuve.

Joindre au compte-rendu la preuve produite par prover9.

2. Montrer, grâce à mace4, que des formules (1) et (2), on ne peut déduire aucune des formules

$\text{all } n (0+n=n)$

$\text{all } n \text{ all } p (s(p)+n=s(p+n))$

$\text{all } n \text{ all } p (n+p=p+n)$

Donnez le modèle, obtenu par mace4, rendant vraies les formules (1), (2) et fausses ces formules.

Quand on cherche une preuve «à la main» de la commutativité de la somme, on voit facilement qu'il faut déjà prouver l'analogue de la définition de la somme sur le premier argument, donc établir les lemmes :

(3) $\text{all } n (0+n=n)$

(4) $\text{all } n \text{ all } p (s(p) + n = s(p+n))$

Puisque ces deux formules, ne peuvent pas être déduites de la seule «définition» de la somme, il faut raisonner par récurrence et c'est ce que nous faisons ci-dessous.

3. Preuve par récurrence de $\text{all } n (0+n=n)$

On ajoute la propriété P ci-dessous et la récurrence associée :

(5) $\text{all } n (P(n) \leftrightarrow (0+n=n))$

(6) $P(0) \ \& \ \text{all } n (P(n) \rightarrow P(s(n))) \rightarrow \text{all } n P(n)$

Prouver à l'aide de prover9 que des formules (1), (2), (5), (6) on peut déduire :

$\text{all } n P(n)$

Joindre au compte-rendu la preuve produite par prover9.

4. Preuve par récurrence $\text{all } n \text{ all } p (s(p)+n=s(p+n))$

Un peu de réflexion, montre que cette récurrence doit être faite sur n .

Aussi aux formules (1) et (2), on ajoute la propriété Q ci-dessous et la récurrence associée :

(7) $\text{all } n (Q(n) \leftrightarrow \text{all } p (s(p)+n = s(p+n)))$

(8) $Q(0) \ \& \ \text{all } n (Q(n) \rightarrow Q(s(n))) \rightarrow \text{all } n Q(n)$

Joindre au compte-rendu la preuve produite par `prover9` et essayer de voir comment sont utilisées les propriétés de l'égalité.

5. Utiliser la récurrence pour prouver $\text{all } n \ p (n+p=p+n)$

Aux formules (1) et (2), il faut ajouter tout d'abord la propriété R ci-dessous et la récurrence associée :

(9) $\text{all } n (R(n) \leftrightarrow (\text{all } p (n+p=p+n)))$

(10) $R(0) \ \& \ \text{all } n (R(n) \rightarrow R(s(n))) \rightarrow \text{all } n R(n)$

Mais cela ne suffit pas, il faut aussi donner au prouveur, soit les lemmes intermédiaires indiqués dans 2, soit le moyen de les prouver, ce que nous demandons de faire pour illustrer les capacités de `prover9`.

Autrement dit, il faut montrer que des formules (1), (2), (5), (6), (7), (8), (9), (10), on peut déduire

$\text{all } n R(n)$.

On ne joindra pas cette preuve au compte-rendu mais seulement la longueur de la preuve.

6. Eviter de reprover les "lemmes" : montrer que des formules (1), (2), (3), (4), (9), (10), on peut déduire

$\text{all } n R(n)$.

On joindra cette preuve au compte-rendu.

Il faut noter, à l'occasion de cet exemple, que `prover9` aide à faire les preuves, mais ne fait pas les preuves à votre place : pour prouver la commutativité de la somme, il a fallu deviner les propriétés intermédiaires à établir, et déterminer toutes les récurrences à faire.

□

Manuel prover9 et mace4

Ceci est un manuel *très abrégé* pour utiliser prover9 un prouveur par résolution et mace4 un programme permettant de construire des modèles.

Nous utiliserons ces deux programmes uniquement sur des exemples très simples. Mais le lecteur, curieux, trouvant ce manuel trop insuffisant, ou désireux d'installer ces logiciels sur sa propre machine, est invité à compléter son information en consultant le site : <http://www.cs.unm.edu/~mccune/mace4/>

1 Le prouveur prover9

On donne au prouveur prover9 une liste de formules. Le prouveur transforme cette liste de formules en une liste de clauses (la forme clausale des formules) puis il recherche une preuve par résolution de ce que la liste de clauses est contradictoire.

Soit le prouveur découvre une preuve qui commence par le mot PROOF, soit il échoue à trouver une preuve ce qui est marqué par le message SEARCH FAILED. Dans le deuxième cas, cela veut dire que :

- soit il n'y a pas de preuve, autrement dit pas de contradiction
- soit le prouveur a été incapable de trouver la preuve

Un bon moyen de vérifier qu'on est dans le premier cas, est de rechercher un modèle de la liste initiale de formules grâce à la commande mace4 appliquée à la même liste.

1.1 Syntaxe des formules

Attention, la syntaxe des formules peut différer légèrement de celle du cours. Une erreur de syntaxe est signalée par le message %%START ERROR%% suivi de la chaîne incorrecte, suivie du message %%END ERROR%%.

1.1.1 Variables et constantes

Dans une *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.

1.1.2 Symboles logiques

Les symboles logiques d'otter sont -, &, |, ->, <->, all, exists. La conjonction et la disjonction sont notées respectivement par &, |.

La priorité des connectives et relations est indiquée par la commande `op(precedence,type,symbol)`. L'ordre des précédences est l'inverse de celui des priorités.

Comme les déclarations par défaut des connectives logiques ne sont pas conformes à celles du cours, on modifie ces priorités en ajoutant avant d'employer ces connectives :

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

1.2 Activer prover9

prover9 lit ses données (qui sont des commandes et des formules) sur l'entrée standard et les écrit sur la sortie standard.

Sur les systèmes Unix, la commande prover9 est presque toujours exécutée en redirigeant l'entrée standard depuis le fichier donnee et parfois la sortie standard vers le fichier resultat :

```
prover9 < donnee
prover9 < donnee > resultat
```

Pour montrer que Socrate est mortel est une conséquence de ce que tout homme est mortel et que Socrate est un homme, on écrit un fichier `socrate.in` comportant les lignes suivantes :

```
op(750,prefix,"-").
op(795,infix_right,"->").
op(805,infix_right,"<->").
set(raw).
set(binary_resolution).
set(factor).
formulas(sos).
% Tous les hommes sont mortels
all x (H(x) -> M(x)).
% Socrate est un homme
H(socrate).
% Socrate n'est pas mortel
-M(socrate).
end_of_list.
```

Entre les commandes `formulas(sos)` et `end_of_list` il y a une liste de formules. Chaque commande et chaque formule soumise à `prover9` est suivie par un point.

On peut (et on doit) mettre des commentaires après le caractère `%`, comme sur l'exemple du fichier `socrate.in`.

1. Avec la commande `prover9 < socrate.in`, les résultats de la commande sont affichés après la commande.
2. Avec la commande `prover9 < socrate.in > socrate.out`, les résultats sont écrits vers le fichier `socrate.out`

Au moment où ce manuel est écrit, le prouveur `prover9` et le constructeur de modèles `mace4` sont disponibles sur la machine `parentir` du DLST.

On peut, en cas de besoin, arrêter le fonctionnement de `prover9` par la frappe de `Ctrl-C`.

1.3 Diriger prover9

On peut diriger le fonctionnement du prouveur au moyen de nombreuses commandes. Si vous souhaitez que le prouveur n'utilise, sauf exception, que les règles de la résolution vue en cours, il faut, comme sur l'exemple, inclure en tête de votre fichier les commandes suivantes :

```
set(raw).
set(binary_resolution).
set(factor).
```

On note que, dans cette configuration, le prouveur n'a presque aucune règle pour traiter l'égalité. Aussi, on demande d'ajouter ces commandes si et seulement si les formules ne comportent pas d'égalité.

2 Le constructeur de modèle mace4

2.1 Rôle de mace4

Le constructeur de modèle `mace4` recherche un modèle fini d'une liste de formules, en commençant avec un domaine à deux éléments.

Dans la suite, la commande `mace4` est le plus souvent exécutée sous l'une des deux formes :

```
mace4 -c -p 1 < donnee
mace4 -c -p 1 < donnee > resultat
```

Mettez toujours l'option `-c`.

L'option `-p 1` impose un format lisible pour les modèles. Observez ce qui se passe si vous l'enlevez.

Le format du fichier `donnee` est le même que celui adopté pour `prover9`. Mais `mace4` ignore les commandes destinées

au prouveur.

L'intérêt de partager le même format entre `prover9` et `mace4` est le suivant : si `prover9` n'a pas réussi à prouver qu'une liste de formules était contradictoire, c'est peut-être que cette liste avait un modèle, et `mace4` vous vient en aide pour trouver ce modèle.

Soit le fichier `skolem.in` :

```
op(750,prefix,"-").
op(795,infix_right,"->").
op(805,infix_right,"<->").
set(raw).
set(binary_resolution).
set(factor).
formulas(sos).
all x exists y P(x,y).
- all x P(x,f(x)).
end_of_list.
```

La commande `mace4 -c -p 1 < skolem.in` trouve un modèle à 2 éléments des deux formules.

Avant de chercher un modèle, `mace4` transforme les formules en clauses, ce qui introduit deux nouveaux symboles, dont `mace4` indique aussi la valeur.

2.2 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 interprète le symbole `=` comme l'identité sur les entiers du domaine et le symbole `!=` comme la non identité sur les entiers.

Soit le fichier `egal.in` ci-dessous, dont nous avons omis les commandes `set` et `op` inutiles dans le petit exemple suivant :

```
formulas(sos).
c!=0 & c!=1 & c!=2.
end_of_list.
```

La commande `mace4 -c -p 1 < egal.in`, termine avec une erreur : en effet `mace4` commence par rechercher un modèle à deux éléments 0, 1 et ne peut pas donner de valeur à 2 sur ce domaine.

Avec la commande `mace4 -c -p 1 -n 3 < egal.in`, le logiciel recherche un modèle avec *exactement* trois éléments et n'en trouve pas. Avec la commande `mace4 -c -p 1 -n 4 < egal.in`, le logiciel recherche un modèle avec *exactement* quatre éléments et trouve le modèle `c=3`.