Démontrer automatiquement des petits théorèmes, mini-Prolog

David Monniaux

25 mars 2005

Le projet dispose d'une page de suivi à l'URL : `http://www.enseignement.polytechnique.fr/profs/informatique/David.Monniaux/prolog/` Consultez-la !

Qui, essayant de démontrer un théorème mathématique, n'a jamais eu recours à la technique d'essayer d'appliquer tous les théorèmes ayant la bonne conclusion ? Il s'agit ici de faire un mini-prouveur de théorèmes se basant sur ce principe, appliqué selon la méthode dite de *résolution*.

1 Définitions et exemples

1.1 Définitions

On dispose d'un certain nombre de *règles de déduction* ou *axiomes* de la forme :

$$\begin{aligned}\forall x_1 \forall x_2 \dots \forall x_n\ H_1\,(t_{11}(x_1,\dots,x_n),\dots,t_{1o_1}(x_1,\dots,x_n))\\ \wedge \dots \wedge H_m\,(t_{m1}(x_1,\dots,x_n),\dots,t_{mo_m}(x_1,\dots,x_n))\\ \Rightarrow C\,(t_1(x_1,\dots,x_n),\dots,t_o(x_1,\dots,x_n))\end{aligned} \tag{1}$$

Les H_i sont les *hypothèses* de la règle, C est la *conclusion*. Il n'est pas obligatoire que chacune des variables x_j interviennent dans toutes les hypothèses et dans la conclusion. L'ensemble des hypothèses peut être vide.

Les t_i sont des *termes*. Un terme désigne soit une variable, soit l'application d'un *symbole de fonction* à des termes. Nous ne chercherons pas à attacher un sens aux symboles de fonctions et aux termes, il suffira donc de les traiter comme des entités syntaxiques formelles. Une fonction avec zéro paramètres est une *constante*.

1.2 Exemple : monoïdes

La théorie des monoïdes (version simplifiée des groupes, sans inverse ; par exemple, le monoïde $(\mathbb{N},+)$ des entiers naturels) s'exprime par les axiomes :

$$\forall x\ x = x.e \tag{2}$$

$$\forall x\ x = e.x \tag{3}$$

$$\forall x_1 \forall y_1 \forall x_2 \forall y_2\ x_1 = x_2 \wedge y_1 = y_2 \Rightarrow x_1.y_1 = x_2.y_2 \tag{4}$$

$$\forall x \forall y \forall z\ (x.y).z = x.(y.z) \tag{5}$$

$$\forall x\ x = x \tag{6}$$

$$\forall x \forall y\ y = x \Rightarrow x = y \tag{7}$$

$$\forall x \forall y \forall z\ x = y \wedge y = z \Rightarrow x = z \tag{8}$$

e est l'élément neutre, . l'opération. Remarquons que les symboles $=$, e et . n'ont pas de sens particulier : il s'agit juste d'entités syntaxiques.

Pour une meilleure compréhension, pensons en termes d'entiers naturels. $2+(3+5)$ et 10 sont deux *termes* syntaxiquement différents. Toutefois, on peut définir une relation $=$ (l'égalité au sens mathématique sur les entiers représentés par ces termes) sur ces termes, qui est une relation d'équivalence compatible avec l'opération $+$: par exemple, puisque $2 + (3 + 5) = 10$, alors $(2 + (3 + 5)) + 6 = 10 + 6$.

Nous traduisons ces formules en forme préfixe, afin d'avoir une syntaxe proche du langage Prolog. $P_1 \wedge \ldots P_n \Rightarrow P$ est remplacé par `P :- P1, ..., Pn.`

```
egal(X, mul(X, e)).
egal(X, mul(e, X)).
egal(mul(X1, Y1), mul(X2, Y2)) :- egal(X1, X2), egal(Y1, Y2).
egal(mul(mul(X, Y), Z), mul(X, mul(Y, Z))).
egal(X, X).
egal(X, Y) :- egal(Y, X).
egal(X, Z) :- egal(X, Y), egal(Y, Z).
```

1.3 Exemple : relations familiales

Suivant la même syntaxe, quelques données sur les derniers rois de France Valois et le premier roi Bourbon (Henri IV).

```
homme(X) :- estLePereDe(X, Y).
femme(X) :- estLaMereDe(X, Y).

estParentDe(X, Y) :- estLePereDe(X, Y).
estParentDe(X, Y) :- estLaMereDe(X, Y).
estGrandMereDe(X, Z) :- estLaMereDe(X, Y), estParentDe(Y, Z).
estGrandPereDe(X, Z) :- estLePereDe(X, Y), estParentDe(Y, Z).
ontUnParentCommun(A, B) :- estParentDe(X, A), estParentDe(X, B).

aEteMarieA(X, Y) :- aEteLeMariDe(X, Y).
aEteMarieA(X, Y) :- aEteLeMariDe(Y, X).
homme(X) :- aEteLeMariDe(X, Y).
femme(X) :- aEteLeMariDe(Y, X).

estBeauFrereParAllianceDe(X, Z) :- aEteLeMariDe(X, Y), ontUnParentCommun(Y, Z).

estLePereDe(henri_II, francois_II).
estLePereDe(henri_II, charles_IX).
estLePereDe(henri_II, henri_III).
estLePereDe(henri_II, marguerite_de_Valois).
estLaMereDe(catherine_de_Medicis, francois_II).
estLaMereDe(catherine_de_Medicis, charles_IX).
estLaMereDe(catherine_de_Medicis, henri_III).
estLaMereDe(catherine_de_Medicis, marguerite_de_Valois).
homme(francois_II).
homme(charles_IX).
homme(henri_III).
femme(marguerite_de_Valois).

aEteLeMariDe(henri_IV, marguerite_de_Valois).
```

1.4 Clauses existentielles

Notre formalise ne permet pas de représenter directement des propriétés du type $\forall x\ \exists y\ P(x, y)$. Il existe cependant une transformation simple, appelée *skolémisation*, qui permet de le faire. La propriété ci-dessus est équivalente à l'existence d'une fonction f telle que $\forall x\ P(x, f(x))$.

2 Résolution propositionnelle

Plaçons d'abord dans le cas où l'on ne fait pas intervenir de variables ni de termes : les règles de déduction sont alors de simples implications de la forme $P_1 \wedge \cdots \wedge P_n \Rightarrow P$.

Étant donné un tel ensemble de règles, une démonstration d'une propriété P est un *arbre de preuve* de la forme :

$$\dfrac{\dfrac{\dfrac{\vdots}{P_1 1}\,A_1 1 \quad \ldots \quad \dfrac{\vdots}{P_1 n_1}\,A_1 n_1}{P_1}\,A_1 \qquad \ldots \qquad \dfrac{\dfrac{\vdots}{P_m 1}\,A_m 1 \quad \ldots \quad \dfrac{\vdots}{P_m n_m}\,A_m n_m}{P_n}\,A_m}{P}\,A \tag{9}$$

Chaque règle

$$\dfrac{P_1 \quad \ldots \quad P_n}{P}\,A$$

se lit : P s'obtient par application de la règle A sur les hypothèses P_1 à P_n.

Pour rechercher un tel arbre de preuve, on peut procéder ainsi : pour chaque règle $\dfrac{P_1 \quad \ldots \quad P_n}{P}\,A$, on essaye de relancer récursivement le procédé pour démontrer les hypothèses P_1 à P_n.

On peut également voir cela de la façon suivante : on considère le graphe orienté G dont les sommets sont des arbres de preuve éventuellement incomplets (i.e. on n'a pas démontré toutes les branches de l'arbres), et dont les arêtes $\pi_1 \to \pi_2$ sont présentes si l'arbre π_2 est égal à l'arbre π_1 dans lequel on a remplacé une proposition P_0 non démontrée par l'application d'une règle A de conclusion P_0 et d'hypothèses P_1 à P_n : autrement dit, on remplace une sous-branche de la forme

$$\begin{array}{c} ??? \\ \vdots \\ P_0 \end{array}$$

par un sous-arbre de la forme

$$\dfrac{\begin{array}{c} ??? \\ \vdots \\ P_1 \end{array} \quad \ldots \quad \begin{array}{c} ??? \\ \vdots \\ P_n \end{array}}{P_0}$$

.

Le problème de savoir s'il existe une démonstration de P se résume donc à savoir s'il existe un chemin de l'arbre de preuve incomplet

$$\begin{array}{c} \vdots \\ P \end{array}$$

à un arbre de preuve complet (qui constitue donc une démonstration de P).

Si l'ensemble des règles applicables est infini, ce graphe peut être infini. Les méthodes de parcours en profondeur et en largeur peuvent être utilisées pour trouver une solution : la méthode en profondeur peut être plus facile à implémenter, mais elle peut partir dans des branches de raisonnement stérile et ne pas découvrir une démonstration ; le parcours en largeur doit découvrir toutes les démonstrations possibles. Notons que dans les deux cas, s'il n'existe pas de preuve alors l'exploration peut ne pas terminer.

3 Unification

Nous utilisons en fait des règles quantifiées universellement, avec des termes en argument. Une règle R de la forme $\forall x_1 \dots \forall x_n P_1(x_1, \dots, x_n) \wedge \dots \wedge P_m(x_1, \dots, x_n) \Rightarrow P(x_1, \dots, x_n)$ représente en fait une infinité de règles sans variables : R dans laquelle on a remplacé $x_1, \dots, x_n$ par des termes quelconques. Cela justifie notre discussion de la recherche de preuves avec un nombre infini de règles.

La recherche d'une règle dont la conclusion correspond à la propriété P choisie est maintenant plus complexe. Par exemple, si l'on a une règle R de la forme $Q(x, y, z) \Rightarrow P(x, S(y), T(z))$ et qu'on cherche à démontrer $\exists a \exists b \exists c\ P(S(a), b, c)$, on pourra appliquer R avec $x = S(a), b = S(y), c = T(z)$; on devra donc ensuite démontrer l'hypothèse Q en tenant compte de ces choix.

Le processus par lequel on rapproche deux termes, ici $P(x, S(y), T(z))$ et $P(S(a), b, c)$, en cherchant sous quelles conditions minimales sur les variables ces termes peuvent être syntaxiquement égaux s'appelle *unification*. Formellement parlant, un *unificateur* de deux termes t_1 et t_2 est une substitution associant un terme (avec variables) à chaque variable ; on note $\sigma.t$ le résultat de la substitution dans t selon les règles de σ. Ce que l'on recherche, c'est un unificateur général, de telle sorte que tout autre unificateur qui convienne soit un cas particulier de cet unificateur général, c'est-à-dire qu'il se décompose en la composée $\sigma_1 \circ \sigma_2$ de deux substitutions.

L'algorithme d'unification est simple. Prenons par exemple l'unification des deux termes $P(x, S(y), T(z))$ et $P(S(a), b, c)$. Les symboles de tête coïncident (mais on n'aurait pas pu, par exemple, unifier $P(x, y)$ et $Q(x, y)$) ; on peut donc continuer. On va unifier x et $S(a)$; on met donc à jour σ en rajoutant la substitution $x \mapsto S(a)$. On passe ensuite à l'unification de $S(y)$ et b ; on obtient alors $b \mapsto S(y)$.

Il convient cependant de faire attention au phénomène dit de *capture des variables*. Prenons par exemple une règle $R : \forall x\ P(x) \Rightarrow Q(x)$; nous cherchons à démontrer $\exists x\ Q(f(x))$. Une approche naïve serait d'unifier $Q(x)$ et $Q(f(x))$, et cela nous donne $x \mapsto f(x)$, ce qui est bizarre et même indésirable (voir plus loin). Notons que la règle R est strictement équivalente à la règle $\forall x'\ P(x') \Rightarrow Q(x')$ par renommage des variables.[1] Si on emploie cette règle, on obtient $x' \mapsto f(x)$, ce qui est raisonnable. Il convient donc, avant d'essayer d'appliquer une règle, de renommer les variables de cette règle en des noms frais (non déjà utilisés).

Un autre problème est celui du *test d'occurrence* (*occur-check*). Nous ne désirons pas avoir des solutions comme $x \mapsto f(x)$, c'est-à-dire des solutions où une variable intervient dans sa propre définition. Prenons un exemple simple pour voir en quoi ceci est indésirable : supposons que nous ayons une règle $\forall x\ P(S(x)) \Rightarrow P(x)$, une formalisation d'une propriété P sur les entiers naturels telle que $\forall n \in \mathbb{N}\ P(n+1) \Rightarrow P(n)$. Une solution de la forme $x = S(x)$ donnerait $n = n+1$, une solution absurde. En fait, si notre seule hypothèse est $\forall n \in \mathbb{N}\ P(n+1) \Rightarrow P(n)$, nous ne pouvons pas démontrer $P(0)$. En conséquence, il devra y avoir un test pour que la définition d'une variable dans une substitution ne fasse pas intervenir cette même variable.

4 Syntaxe et utilisation

Nous avons proposé ci-dessus des exemples écrits dans une syntaxe proche de celle du langage Prolog. Ces exemples sont une succession de trois types de commandes :

– des *règles* de la forme `P :- P1, P2,...,Pn.`, représentant $\forall x_1 \dots \forall x_n P_1 \wedge \dots P_n \Rightarrow P$; on distingue les variables des constantes par le fait que les variables prennent une majuscule initiale, tandis que les constantes n'en prennent pas ;

[1] On appelle parfois ce renommage α-conversion.

– des *axiomes* de la forme `P.` ; il s'agit juste d'une commodité de notation pour `P :- .` ;
– des requêtes de la forme `?- P.`

En réponse à une requête P :

– s'il n'y a pas de variable dans la requête P, le système cherche au moins une preuve de P ; s'il en trouve une, il affiche `yes` ; s'il n'en trouve pas, il affiche `no` ; il peut très bien ne pas terminer s'il n'existe pas de preuve (ou si sa recherche est trop longue) ;
– s'il y a au moins une variable, le système va afficher l'ensemble des solutions trouvées.

Ainsi, sur l'exemple familial ci-dessus, la requête

```
?- estBeauFrereParAllianceDe(henri_IV, X).
```

va produire les solutions :

```
{
  X -> francois_II()
}
{
  X -> charles_IX()
}
{
  X -> henri_III()
}
{
  X -> marguerite_de_Valois()
}
```

Remarquons qu'il y a une faiblesse dans la définition de frère et sœur : une personne est considérée comme son propre frère (ou sa propre sœur). L'ajout et l'utilisation d'une propriété `different` et d'axiomes `different(`x`, `y`)` pour toute paire de personnes différentes est laissé comme exercice au lecteur.

Il n'est pas demandé de faire un algorithme optimisé. Il existe cependant un grand nombre d'optimisations possibles : simplification des unificateurs ; si pour démontrer une propriété P, on demande de démontrer la même propriété ou une propriété essentiellement isomorphe, alors on arrête dans cette voie ; etc. Remarquons que certaines règles peuvent provoquer de tels problèmes : il s'agit notamment des règles d'associativité et de commutativité.