Projet LDP 2006/2007

« Calculatrice de formules booléennes quantifiées réalisé en OCaml »

Description du projet

Une simple syntaxe pour des formules booléennes quantifiées (http://en.wikipedia.org/wiki/Quantified_boolean_formula_problem) soit donnée par la grammaire suivante:

grammaire lexique hors mots clés (très standard) et les autres jetons indiqués dans le grammaire non-contextuelle ci-dessous :

```
alpha ::= [a-zA-Z0-9_]
whitespace ::= [ \n\t]*
ident ::= [a-z]alpha*
macroident ::= [A-Z]alpha*
```

grammaire non-contextuelle, mots clés et d'autre jetons en gras :

```
programme ::=    commande ; programme | EOF
commande ::=     macro macroident =  transformule |
                 macro macroident ( identlist ) = formule |
                 print formule |
                 solve formule |
                 printordereditеprogramme identlist :formule

transformule ::= elimquant formule | fnd formule | fnc formule |
                 fnite formule| simplify formule | formule

formule ::=        1 | 0
                 | formule || formule
                 | formule && formule
                 | ! formule
                 | ( formule ? formule : formule  )
                 | ( formule )
                 | ( A identlist : formule )
                 | ( E identlist : formule )
                 | ident
                 | macroindent (listedeformules)
                 | macroindent
listedeformules ::= formule , listedeformules | formule
identlist ::= ident , identlist | ident
```

Afin de résoudre les ambiguïtés du grammaire, on ajoute que les opérateurs « || » et « && » sont associatif à gauche, avec « && » prioritaire sur « || », et « ! » encore prioritaire. Les identificateurs représentent des variables booléennes, la quantification A (universelle) et E(existentielle) porte donc pour chaque variable sur les valeurs «1» et «0 ». Exemple d'une session d'évaluation d'un programme :

```
> macro X = b || ! b ;
> macro  S = ( A b : X );
```

```
> print S;
(A b : b || ! b)
> macro Imp(x,y) = !x || y;
> macro Equ(x,y) = Imp(x,y) && Imp(y,x);
> print Equ(X,1);
(!(b || !b) || 1) && (!1 || (b || !b))
> macro Xs = simplify Equ(X,1);
> print Xs;
 b || ! b
> macro Ss = elimquant S;
> print Ss;
(0 || ! 0) && (!1 || !1)
> macro Sss = simplify Ss;
> print Ss;
1
> solve (b ||! c);
b = 1, c=1
> solve ! Equ(X,1);
no solution!
> macro Y = Z || ! Z;
error : Z unbound
> print Equ(X);
error : wrong number of parameters for macro Equ
...
```

D'autres explications : «fnd» signifie la forme normale disjonctive, « fnc » la forme normale conjonctive, « fnite » est une forme normal qui n'utilise que des variables, « 1 », « 0 » et l'opérateur « (? :) » (if then else). « solve » calcule une attribution de valeurs de vérité aux variables libres de la formule, si une telle solution existe.

Tâches :

1. Définir deux types récursifs « formule » et « commande » pour une représentation abstraite (arbre) de la syntaxe indiquée.
2. Réaliser un module d'impression (transformation en strings) pour ces types. N'oubliez pas d'insérer suffisamment de parenthèses pour rendre ces représentations correctes (moins de parenthèses sont mieux, si la représentation reste correcte). Une indentation sensée serait un plus.
3. Créer une analyse syntaxique et lexique à l'aide de camllex et camlyacc pour le grammaire indiquée avec les symboles « commande » et « programme » comme symboles initiaux.
4. Réaliser des versions du programme en réalisant **dans l'ordre** les commandes :

   print, macro, elimquant, fnd, fnc, fnite, simplify, solve
5. BONUS : Réaliser la commande printiteprogramme : Cette dernière imprime à la place d'une formule un programme avec des macros de la forme « macro A = (x,B,C) » un dernier macro « F(...) =A» portant sur toutes les variables de la liste donnée en sorte que les macros respectent l'ordre de la liste de variables : si « y » figure avant « x » dans la liste des variables, alors dans une définition A =(x,B,C) la variable « y » de devait pas figurer directement ou indirectement dans les définitions de B et de C. De plus, le nombre de macros devait être minimal, c'est à dire que jamais deux macros devaient définir la même formule.

Indications

La seule documentation dont vous avez besoin est le manuel de référence de Ocaml, en ligne à l'adresse http://caml.inria.fr/pub/docs/manual-ocaml . En particulier vous deviez apprendre à vous servir du compilateur. Pour ceci, vous allez trouver « OCamlMakefile » utile, qui se trouve sur le web ici : http://ocaml.info/home/ocaml_sources.html#toc15 .

Analyse syntaxique. Il s'agit de transformer des chaînes de caractères en expressions représentés par des arbres syntaxiques. Utilisez pour ceci les outils **ocamllex** et **ocamlyacc**., qui sont décrit dans le manuel de référence Ocaml. Le fonctionnement est très similaire à lex et yacc, avec la différence que les productions produisent des valeurs Ocaml d'une façon fonctionnelle.

Fonctions auxiliaires. Afin de réaliser votre projet, vous déviez réaliser un bon nombre de transformations (récursives) de formules, notamment de **substitution** pour l'élimination des quantificateurs. Bien planifier les phases de ces transformations.

Environnement de macros. Vous pouvez constater que l'environnement de macros évolue pendant l'exécution d'un programme. Il est suggéré de réalisé ceci avec des aspects impératifs du langage, par exemple une référence vers une liste de définitions.

Gestion d'erreurs. Il est suggérer de définir des exceptions pour la gestion des erreurs.

BONUS. Cette tâche un peu difficile n'est pas nécessaire pour la moyenne mais elle peut considérablement améliorer votre évaluation. Elle suggère l'utilisation d'aspects impératifs (enregistrements mutables), mais d'autres solutions existent. Il est utile de regarder http://en.wikipedia.org/wiki/Binary_decision_diagram

Modalités de travail.

On travaille impérativement par groupe (entre 2 et 3 étudiants obligatoirement). Inclure les noms des participants au groupe dans le fichier README et dans chaque fichier source, ainsi qu'une description de la contribution de chaque participant à la réalisation du pro jet. La structuration du projet sera documenté dans le fichier README et sera mise en ouvre à l'aide de fichiers indépendants

(modules, possiblement avec interface fichier .mli).

Modalités de soumission.

Préparer un répertoire contenant le code source, le fichier README, le fichier Makefile, et rien d'autre. Prendre contacte avec le responsable de votre groupe de TP (plus de détailles à venir). Date limite de soumission : à communiquer.

Modalités de soutenance.

Dans l'évaluation de votre projet, une importance particulière sera donnée aux critères suivants :

Documentation. Votre projet contiendra un fichier README contenant :

- La description des programmes adressé à un utilisateur non-expert.
- Une description concise des programmes (adressés aux enseignants) décrivant les choix d'implémentation, les structures de données utilisés, les résultats accomplis par rapport à chaque tache décrite ci-dessus.

Qualité du code.

Le code source sera bien commenté (commenter d'abord les fichiers interface .mli), les noms des variables, des fonctions, et des types seront bien choisis.

Organisation des programmes en unités logiques. Chaque unité logique sera contenue dans un fichier séparé nom.ml (avec nom est bien choisi) ayant un fichier en interface nom.mli par défaut. La liste des unités logiques (donc la liste du code source) apparaîtra dans le fichier README, la compilation sera gérée par la commande make à l'aide d'un fichier Makefile possiblement créé à partir de OCamlMakefile.

Fonctionnement de votre code. Le code source sera compilé au moment de l'évaluation. Votre présence n'est pas nécessaire pour l'évaluation (toute explication éventuelle doit être contenue dans le fichier README). Le fonctionnement de votre code sera testé par rapport aux résultats déclarés accomplis dans le fichier README.

Remarque important

Il est impératif de soumettre le projet avant la date limite. Tout projet rendu après cette date ne sera pas considéré. Il ne sera pas possible rendre le projet pendant ou après la session d'hiver. De même, il ne sera pas possible rendre une version amélioré du projet pendant ou après la session d'hiver.