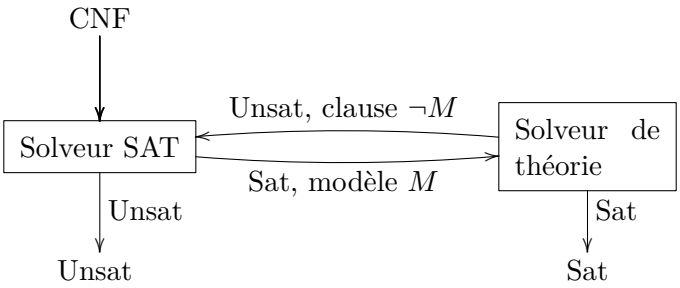
TP déduction automatique n° 2

Programmation raisonnée, ENSIIE

Semestre 5, 2024–25

Le but de ce TP est d'écrire un solveur SMT *offline*, c'est-à-dire que le solveur SAT et le solveur de théorie seront considérés comme des boîtes noires. On rappelle le schéma de base d'un solveur SMT :



On utilisera `glucose` comme solveur SAT, et l'outil de programmation linéaire `glpsol` de la bibliothèque GLPK comme solveur pour la théorie de l'arithmétique linéaire entière.

On trouvera dans `/pub/FISE_PRRU35/` une archive `smtoff.tar.gz` qui contient un début de code pour le prouveur SMT, dans laquelle sont présents :

- un `Makefile`
- des fichiers `lexlia.mll` et `parselia.mly` pour lire des fichiers de problèmes contenant de l'arithmétique linéaire. Ces fichiers contiennent une clause par ligne, une clause étant une liste de littéraux séparés par `|`, un littéral étant soit une égalité `=`, soit une différence `!=`, soit une inégalité `<`, `<=`, `>` ou `>=` entre expressions arithmétiques linéaires (c'est-à-dire, pas de produit de variables).
- des exemples de fichiers de problèmes `sat.p` et `unsat.p`
- un fichier `main.ml` à compléter qui pour l'instant se contente de lire un fichier sur l'entrée standard et d'afficher comment il est lu sur la sortie standard.
- un fichier `smt.ml` qui contient
  - la définition des littéraux, des clauses, des problèmes et des modèles en arithmétique linéaire. On remarquera en particulier qu'on n'a plus que deux types de littéraux, $\leq$ et $>$. En effet, on peut se ramener aux autres cas en utilisant

les transformations :

$$
\begin{aligned}
a < b &\rightsquigarrow a \leq b - 1 \\
a \geq b &\rightsquigarrow a > b - 1 \\
a = b &\rightsquigarrow a \leq b \ \wedge \ a > b - 1 \\
a \neq b &\rightsquigarrow a \leq b - 1 \ \vee \ a > b
\end{aligned}
$$

— la définition des problèmes et des modèles en logique propositionnelle ;

— une fonction `call_glucose` qui prend en argument en problème en logique propositionnelle et qui lance `glucose` dessus. La fonction retourne un modèle en logique propositionnelle si le problème est satisfiable, ou lève l'exception `Unsat` sinon ;

— une fonction `call_glpsol` qui prend en argument un modèle en arithmétique linéaire et qui lance `glpsol` dessus. La fonction retourne `true` si le problème est satisfiable, `false` sinon.

1. Compiler à l'aide de la commande `make` (Insister si une erreur s'affiche.) Tester le programme `smt` produit sur les exemples.
2. Écrire une fonction `model_to_clause` qui transforme un modèle (en arithmétique linéaire) en la clause qui est sa négation. Par exemple, pour le modèle $x \leq 2; x + y > 3; y \leq 1$ on retournera la clause $x > 2 \vee x + y \leq 3 \vee y > 1$.
3. Écrire une fonction `arith_to_sat` qui prend un problème d'arithmétique linéaire et qui le transforme en problème en logique propositionnelle. L'idée est de voire p `<=` q comme un littéral positif (donc un entier positif au format DIMACS) dont la négation (l'opposé) est p `>` q. Il faut donc mettre en bijection l'ensemble des littéraux arithmétiques `<=` avec l'ensemble des variables propositionnelles représentées par des entiers positifs. On pourra pour cela utiliser les tables de hachage `poly_htbl` et `ylop_htbl` qui permettent respectivement d'associer à une expression arithmétique un entier et à un entier une expression arithmétique.
4. Écrire une fonction `sat_to_arith` qui prend un modèle en logique propositionnelle et qui retourne le modèle en arithmétique linéaire correspondant. Ici aussi, il sera utile d'employer `ylop_htbl`.
5. Écrire une fonction `dpllt` qui prend en argument un problème en arithmétique linéaire et qui le résout en utilisant le schéma indiqué ci-dessus. La fonction retournera `true` si le problème est satisfiable, `false` sinon.
6. Modifier `main.ml` pour appeler `dpllt` et pour afficher suivant les cas `Sat` ou `Unsat`.