Projet 2 L3IF — Premier rendu

à faire en binôme, à rendre pour le 01/03/2015 à 23h59
envoyez une archive qui compile à `julien.bensmail@ens-lyon.fr`, `vincent.lanore@ens-lyon.fr` et `daniel.hirschkoff@ens-lyon.fr`

1 Ce qui est attendu de votre programme

Entrée, sortie. La même chose que pour le DM : une formule en forme normale conjonctive, avec le même format que pour le DM en entrée, même manière d'invoquer le programme, l'information sur le problème SAT correspondant en sortie.

Algorithme(s). Tout le monde devra coder une version "de base" de l'algorithme DPLL vu en cours.

On se contentera d'une approche naïve pour la sélection de la prochaine variable sur laquelle miser (optez pour le plus simple, du style "la prochaine qui est inconnue", en commençant par la supposer vraie).

> **En plus.** Les binômes "avancés" devront *également* coder la technique des litéraux surveillés (*watched literals*).
> Il faudra avoir deux versions effectives du code, avec et sans watched literals. L'exécutable `resol` fera tourner DPLL en version naïve, l'exécutable `resol-wl` correspondra à DPLL + watched literals.
> Pour savoir si vous êtes dans un binôme "avancé", reportez-vous à la liste des binômes disponibles à partir de la page du cours.

2 Que prendre en compte

Modularité. Découpez votre code en plusieurs fichiers (la compilation se faisant à l'aide d'un Makefile).

Outre un fichier principal contenant le cœur de l'algorithme DPLL, il faudra aussi gérer de manière "propre" l'accès aux diverses structures de données (litéraux, clauses).

Autant que faire se peut, il faudra que vous conceviez la structure générale de votre code en ayant en tête les extensions mentionnées sur les transparents du cours (voir le transparent 19).

Efficacité. Tâchez de faire un compromis raisonnable entre vos capacités en programmation et l'efficacité dans la manipulation des données. Expliquez vos choix d'implémentation dans le fichier README, et indiquez les endroits où vous êtes conscients du fait que l'efficacité pourrait être améliorée.

Pré-traitement de l'entrée. On souhaite détecter les clauses tautologiques (contenant $x_i \vee \overline{x_i}$), et les clauses unitaires. Les fonctions de propagation peuvent par ailleurs être utilisées pour le prétraitement de l'entrée. Vous pouvez également songer à des traitements plus poussés : détecter les variables apparaissant avec une seule polarité, collecter des informations sur les clauses et les variables, etc.

Traçabilité élémentaire. Munissez votre programme de quelques fonctions d'affichage pas nécessairement très sophistiquées pour pouvoir examiner ce qui se passe en cours d'exécution. Le but est de pouvoir afficher l'état du système alors que l'algorithme tourne, et ce n'est pas très grave si ce n'est pas ultra lisible. Cela pourra vous servir en particulier à développer et débugger votre programme. Expliquez dans votre rendu comment utiliser la fonctionnalité d'affichage.

Tests. Testez votre programme *au moins* sur les fichiers de tests du DM.

Pour les binômes "avancés", une comparaison sommaire entre les performances avec et sans *watched literals* est demandée.

3 Une "check list" pour votre rendu

- Une archive envoyée **à l'heure** aux trois encadrants, avec un nom de fichier significatif.
- Ça respecte les consignes du rendu.
- Le code est structuré de manière lisible, et commenté.
- Ça compile et fonctionne sur les machines des salles libre service de l'ENS.
- Ce fut testé (il y a un sous-répertoire contenant des fichiers de test que vous avez utilisés).
- Un fichier README contenant au moins les points suivants:
  1. Une description de la structuration du code, et des choix d'implémentation importants (en particulier, structures de données, avec éventuellement des remarques sur des améliorations que vous envisagez d'apporter).
  2. Quelques mots sur la répartition du travail pour ce rendu (qui a fait quoi).
  3. Des commentaires sur les performances que vous avez pu observer en testant votre solveur.

     Pour les binômes avancés, une comparaison entre formules où les littéraux surveillés améliorent les performances et d'autres où ça n'est pas le cas.