Projet 2 L3IF — Troisième rendu

à faire en binôme, à rendre pour le 16/4/2013 à 23h59
envoyez une archive qui compile à `benjamin.girault@ens-lyon.fr` et `daniel.hirschkoff@ens-lyon.fr`

Dans ce troisième rendu, on raffine encore l'algorithme DPLL, en ajoutant l'apprentissage de clauses.

1 *Clause learning*

Enrichissez votre programme avec l'apprentissage de clauses, en modifiant la façon dont le backtrack est fait. Autrement dit, plutôt que de systématiquement revenir sur le dernier pari qui a été fait, analysez la raison du conflit et ajoutez une clause à chaque backtrack, en retournant en arrière vers un point judicieux (pas nécessairement le dernier pari).

Il faudra que les améliorations que vous aviez codées pour les rendus précédents continuent de marcher avec cet enrichissement, ce qui pourra nécessiter de reprendre en partie l'organisation de votre code.

2 Expliquer les conflits

Votre programme devra donner la possibilité de "documenter" l'ajout de clauses (ce qui pourra d'ailleurs s'avérer utile en cours de développement, pour trouver des bugs), et ceci suivant deux déclinaisons:

1. Il faudra d'une part que votre programme propose un minimum d'interaction pour pouvoir visualiser le résultat de l'analyse lors des backtracks. Plus précisément:

   - on lance le programme, il s'arrête au premier conflit, et il affiche la clause qui est ajoutée lors de ce conflit;
   - on peut alors taper
     - "`g`" pour engendrer un fichier `dot` décrivant le graphe des conflits;
     - "`r`" pour afficher la preuve par résolution qui aboutit à la clause qui est ajoutée lors du conflit courant;
     - "`c`" pour continuer jusqu'au prochain conflit (et la saisie au clavier recommence);
     - "`s 12`" pour continuer et s'arrêter 12 conflits plus loin;
     - "`t`" pour terminer l'exécution sans s'arrêter.

   Pour la commande `g`, il faudra engendrer un fichier au format `dot/graphviz`. Voir ici, et le petit exemple disponible depuis la page www du cours. Le graphe devra mettre en évidence les nœuds du graphe qui sont dans le niveau de décision courant, ainsi que ceux qui font partie de l'ensemble "UIP" choisi, afin que la clause nouvellement engendrée soit facilement visible.

   Pour la commande `r`, il faudra engendrer un fichier LaTeX (un exemple se trouve sur la page du cours).

   Il faudra également mettre un "interrupteur" (*switch*) permettant de désactiver le mode interactif, qui peut être fastidieux à manipuler. Cela peut se faire par exemple par l'intermédiaire d'une option de votre exécutable.

2. D'autre part, à l'issue d'une exécution, un fichier de documentation sera produit (comme ci-dessus, mettez un "interrupteur" afin de pouvoir désactiver la génération de ce fichier de documentation). Ce fichier permettra de reconstruire l'information suivante pour l'ensemble des backtracks qui ont été effectués lors de l'exécution:

   - quel était le graphe des conflits lors de ce backtrack;
   - quelle clause a été ajoutée.

À vous de concevoir un format approprié (si possible, pas trop verbeux) pour le stockage de cette information.

Écrivez un programme à part, capable de lire ces "fichiers de documentation", et d'afficher les mêmes informations que ci-dessus, pour un numéro de backtrack passé en argument (on tapera typiquement quelque chose comme "`histoire documentation.txt 7`" pour examiner le 7ème backtrack).

Il est *très recommandé* que ces deux modes d'exploration des conflits (en cours d'exécution / après coup) s'appuient sur du code en commun.

Travail à faire, suivant les binômes. Pour tout le monde: dans votre rendu, proposez au moins un exemple servant de support à un "guide d'utilisation" de votre système d'exploration des conflits: on lance le programme, on s'arrête sur tel ou tel backtrack, et on constate que la clause ajoutée est "blabla".

Si vous n'êtes pas semi-avancés ou avancés, vous pouvez choisir entre traiter l'option `g` ou l'option `r`, dans les deux déclinaisons du programme (version interactive, fichier de documentation). Si vous êtes semi-avancés ou avancés, faites tout.

3 Expériences

Reprenez les moulinettes du rendu numéro 2 afin de concevoir des expériences mettant en évidence

- les améliorations de performances apportées par les approches ajoutées lors de ce rendu;
- une étude de la combinaison entre apprentissage de clauses et les autres heuristiques que vous avez implémentées précédemment (watched literals, Rand, MOMS, DLIS).

4 Une "check list" pour votre rendu

- Une archive envoyée **à l'heure** aux deux encadrants, avec un nom significatif.
- C'est bien la dernière version du programme que vous nous envoyez.
- Ça respecte les consignes du rendu.
- Ça compile (sur votre ordinateur *et* sur les machines des salles libre-service).
- Ce fut testé.
- Un fichier README contient les explications
  - sur l'utilisation de vos programmes (comment les exécuter, comment récupérer leurs sorties);
  - sur les choix d'implémentation importants, et sur l'organisation du code;
    en particulier, expliquez comment vous avez dû modifier votre code pour prendre en compte
    - le fait que l'on peut ajouter des clauses en cours de route;
    - la nécessité de garder une trace de la raison pour laquelle la valeur d'un litéral a été déduite;
  - sur la répartition (indicative, nous comprenons bien qu'au bout d'un certain temps vous travaillez à deux sur les mêmes fichiers) du travail dans les fichiers (qui a travaillé le plus sur quels fichiers).
  - sur d'autres remarques que vous pouvez faire.
- Le code est structuré de manière lisible, et commenté.