Projet 2 L3IF — Troisième rendu, apprentissage de clauses

à rendre pour le **jeudi 14/04/2016 à 23h59**

envoyez une archive *qui compile* à `olga.kupriianova@ens-lyon.fr`, `antoine.plet@ens-lyon.fr` et `daniel.hirschkoff@ens-lyon.fr`

Déclinaisons du menu suivant les binômes

Vous recevrez par mail le menu qu'il vous est demandé de traiter pour le rendu 3. À titre indicatif, la plupart des binômes aura à traiter (totalement ou partiellement) l'apprentissage des clauses.

1 Heuristiques et structuration du code

Il vous est demandé de traiter les parties 3.2 et 3.4 du rendu 2.

Si vous codez en Caml, vous devrez apprendre l'usage des modules et des signatures, et structurer votre code de manière à ce que chaque heuristique pour les paris corresponde à un module différent, qui est passé en paramètre à un foncteur.

2 Apprentissage de clauses

Le traitement de l'apprentissage de clauses est décomposé en deux étapes. Nous vous recommandons d'organiser votre travail suivant cette décomposition : commencez par vous assurer que la première phase est correcte avant de passer à la seconde. Comme toujours, il vaut mieux rendre un programme qui fait bien la première phrase et ne traite pas la seconde plutôt qu'un programme qui fait tout mal.

2.1 Première phase : explorer un conflit

On lance le programme: pour cela, il faut taper `./resol -cl-interac fichier.cnf`

Le programme s'arrête au premier conflit, et propose un mode interactif. L'utilisateur peut alors taper

- "`g`" pour engendrer un fichier au format `dot/graphviz` décrivant le graphe des conflits. Le graphe devra mettre en évidence les nœuds qui sont dans le niveau de décision courant (en bleu), ainsi que ceux qui font partie de l'ensemble "UIP" choisi (en violet, sauf le nœud UIP qui sera en jaune), afin que la clause nouvellement engendrée soit facilement lisible. Ne dessinez pas tout le graphe des conflits, ne représentez que les les nœuds bleus/violets/jaune et leurs causes.
- "`c`" pour continuer jusqu'au prochain conflit (et la saisie au clavier recommence);
- "`t`" pour aller jusqu'au bout de l'exécution sans s'arrêter.

NB : tant que vous ne traitez pas la seconde phase décrite ci-dessous, il n'y a pas d'apprentissage de clause.

Pouvoir débrancher le graphe. Veillez à ce que les calculs nécessaires au dessin du graphe ne soient pas faits systématiquement à partir du moment où l'on a choisi l'option `-cl`. Ceci afin de ne pas pénaliser les performances du solveur avec l'option `-cl` (voir seconde phase ci-dessous).

Pour le rendu. Expliquez clairement comment vous avez adapté le programme des rendus précédents (modification des structures de données, code additionnel) afin d'avoir ce qu'il faut pour faire l'apprentissage de clauses et dessiner le graphe de conflits.

Exemple(s) pédagogique(s). 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" (tout cela étant expliqué dans le fichier README).

2.2 Seconde phase : apprentissage de clauses

À chaque conflit, calculez la clause que l'on ajoute, ainsi que le niveau auquel il convient de faire un backtrack.

Gardez à l'esprit que le backtrack peut possiblement se faire plus "haut" que le dernier pari ayant été effectué: si le dernier pari est au niveau 12, et si tous les littéraux de la clause que l'on rajoute sauf un sont de niveau au plus 8, on saute au niveau 8 (intuitivement, la valeur de l'unique litéral de niveau 12 aurait pu être déduite dès le niveau 8).

Dans le mode interactif, le programme affiche la clause qui est ajoutée (sous la forme habituelle, avec un zéro à la fin), et, à la ligne suivante, le niveau où il va backtracker (affichez simplement un entier).

Proposez aussi la possibilité de lancer `./resol -cl fichier.cnf`
pour lancer le solveur avec apprentissage de clauses, mais sans mode interactif.

2.3 Autres ajouts — binômes avancés

2.3.1 Expériences

Les parties 2.3.2 et 2.3.3 ont pour effet d'introduire de nouvelles versions de votre solveur. Au-delà de la correction du solveur (qui est bien entendu une propriété indispensable), reprenez le rendu précédent de manière à faire une évaluation expérimentale des améliorations apportées par ces nouvelles options, en les combinant ou pas avec les options existantes.

2.3.2 CL et WL

Commencez par développer CL (*clause learning*) sans WL (*watched literals*), puis, lorsque cela marche, passez à CL avec WL.

2.3.3 Deux heuristiques en plus

Pour les paris.

VSIDS (*Variable State Independent Decaying Sum*).

L'idée est d'estimer l'*activité* d'une variable. On associe à chaque variable un *score*, qui augmente à chaque fois qu'elle apparaît dans une clause apprise (variante : à chaque fois qu'elle est manipulée dans l'analyse de conflit). On divise régulièrement par une constante (par exemple 2), pour privilégier l'activité récente. Lorsque l'on doit faire un pari, on choisit le littéral non affecté dont l'activité est la plus grande.

option : `-vsids`

Oubli.

Associez un score, que vous définirez vous-mêmes, aux clauses apprises (pas aux clauses du problème initial), de façon à estimer leur *activité*. L'activité croît lorsqu'une clause intervient dans l'analyse d'un conflit. Vous pouvez également raffiner cette heuristique en privilégiant les clauses plus courtes.

option : `-forget`

2.3.4 Bonus

Viser le premier UIP

Pendant que l'on construit la clause que l'on apprend, lorsqu'il y a k litéraux qui ont été *déduits* au niveau courant avec $k \geq 2$, faire la résolution avec celui qui a été affecté en dernier. Voire faire la résolution avec les $k-1$ les plus récemment affectés, simultanément.

Ceci pour éviter de "rater" le premier UIP, en remontant le graphe des conflits de manière maladroite.

Prouver la non satisfiabilité Lorsque le problème est satisfiable, le solveur peut afficher une affectation des variables qui satisfait la formule donnée en entrée.

Dans le cas contraire, on peut dériver une contradiction en utilisant la résolution.

En appelant le programme de la manière suivante

```
./resol -explainunsat
```

le solveur fournira, lorsque le problème est insatisfiable :

- un sous-ensemble des clauses du problème initial suffisant à dériver une contradiction ;
- une preuve par résolution de la contradiction. Celle-ci pourra être affichée suivant un format que vous définirez, ou alors sous forme d'arbre de dérivation en LaTeX.

3 Et à la fin...

...cf. rendus précédents.