Travaux Pratiques

Outils Logiques (OL3)

25 septembre 2006

1 TP1 : Méthode de Davis-Putnam

1.1 Objectifs

Le but de ce TP est d'implanter en langage Java la procédure de Davis-Putnam : vous devez réaliser un programme qui prend en entrée une formule A en forme normale conjonctive, puis décide si cette formule est satisfiable et si c'est le cas, renvoie une interprétation qui satisfait A. Le programme que vous allez réaliser va lire les formules à traiter dans un fichier. Ce fichier respecte un format particulier : le format DIMACS. Le choix des structures de données à employer est de votre ressort. Il est fortement conseillé de bien réfléchir à l'intégralité de l'algorithme avant d'implémenter les classes et les méthodes dont vous aurez besoin.

1.2 Définitions

Rappelons quelques définitions :
- un littéral est une variable propositionnelle x ou sa négation $\neg x$;
- une clause est une disjonction de littéraux ;
- une clause est une *tautologie* si et seulement si elle contient une variable x et sa négation $\neg x$.
- une clause est dite *unitaire* si elle contient exactement un littéral,
- une formule en forme normale conjonctive est une conjonction de clauses,
- une affectation v est une fonction partielle des variables aux valeurs booléennes.

1.3 Format DIMACS (http ://www.satlib.org/Benchmarks/SAT/satformat.ps)

Par exemple, la formule :

$$(x_1 \vee x_3 \vee \neg x_4) \wedge (x_4) \wedge (x_2 \vee \neg x_3)$$

peut être codée par :

```
c Exemple fichier au format CNF
p cnf 4 3
1 3 -4 0
4 0
2 -3
```

- la ligne c est une ligne commentaire,

– la ligne p spécifie qu'il s'agit d'une formule en CNF avec 4 variables et 3 clauses,
– les lignes suivantes spécifient les clauses. Le littéral x_i est codé par i et le littéral $\neg x_i$ par $-i$ où $i \geq 1$ (et dans ce cas $i \leq 4$),
– les clauses peuvent être sur plusieurs lignes et elles sont séparées par 0.

Dans la page du cours, nous fournissons les fonctions **afficheDimacs** et **ecrisDimacs**, qui sont un exemple de lecture et d'écriture de fichiers DIMACS. Vous pourrez vous baser sur ces exemples pour réaliser l'interface de votre programme.

La réalisation de l'algorithme de Davis-Putnam exige la manipulation de formules, de clauses et d'affectations. Vous devrez donc définir les classes correspondantes ainsi que les méthodes dont vous aurez besoin. Voici quelques méthodes de base (il s'agit de simples suggestions, certaines méthodes pourront être omises ou ajoutées selon vos besoins).

Classe formule :
– un constructeur qui lit une formule CNF en format DIMACS et construit la formule correspondante,
– **vide** qui teste si la formule est vide,
– **affiche** qui écrit une formule dans un fichier au format DIMACS,
– **verifie** qui prend en argument une affectation et qui renvoie **true** si la formule est vraie dans cette affectation.

Classe clause :
– **appartient** qui prend en argument un littéral et qui renvoie **true** s'il apparaît dans la clause,
– **unitaire** qui renvoie **true** si la clause ne contient qu'un seul littéral,
– **vide** qui renvoie **true** si la clause est vide,
– **verifie** qui prend en argument une affectation, et qui renvoie **true** si la clause est vraie dans cette affectation.

Classe affectation :
– **fixe** qui prend en argument un littéral et un booléen et qui ajoute le littéral à l'affectation avec la valeur du booléen,
– **valeur** qui prend en argument un littéral et renvoie sa valeur dans l'affectation.

1.4 Davis-Putnam

La méthode de Davis-Putnam permet de décider si une formule en forme normale conjonctive est satisfiable. On représente une formule A en CNF comme un *ensemble* (éventuellement vide) de clauses $\{C_1, \ldots, C_n\}$ et une clause C comme un *ensemble* (éventuellement vide) de littéraux. Dans cette représentation, on définit la substitution $[b/x]A$ d'une valeur booléenne b dans A comme suit :

$$[b/x]A = \{[b/x]C \mid C \in A \text{ et } [b/x]C \neq 1\}$$

$$[b/x]C = \begin{cases} 1 & \text{si } (b = 1 \text{ et } x \in C) \text{ ou } (b = 0 \text{ et } \neg x \in C) \\ C \backslash \{\ell\} & \text{si } (b = 1 \text{ et } \ell = \neg x \in C) \text{ ou } (b = 0 \text{ et } \ell = x \in C) \\ C & \text{autrement} \end{cases}$$

La méthode de Davis-Putnam fonctionne comme suit. Au départ, A est une formule CNF :

– si A est vide, retourner true.
– si A contient la clause vide, retourner false.

– si A contient une clause C qui contient à la fois les littéraux x et $\neg x$, appeler la fonction davis-putnam sur la formule $A \setminus C$.
– si A contient une clause $\{x\}$ (resp. $\{\neg x\}$), appeler la fonction davis-putnam sur $[1/x]A$ (resp. $[0/x]A$).
– sinon, choisir une variable x dans A. Appliquer la procédure DP récursivement sur $[1/x]A$ et $[0/x]A$. Retourner true si l'un des résultats est true, retourner false sinon.

Vous devrez définir des méthodes pour chacune de ces opérations. La dernière, en particulier, doit être traitée avec attention : si la première affectation choisie échoue, il faut pouvoir revenir à l'état courant pour tester la deuxième ; une forme de sauvegarde ou de duplication sera donc nécessaire.

Exercice 1.1 *1- Programmez une méthode estSatisfiable qui décide si la formule est satisfiable en utilisant la procédure de Davis-Putnam*
2- Modifiez la fonction estSatisfiable pour que si la formule est satisfiable, elle affiche une affectation v qui satisfait A.

1.5 Test

Il s'agit maintenant de tester la correction et l'efficacité de votre programme.
– Il est facile de vérifier si une formule A est satisfiable par une affectation v.

Exercice 1.2 *Programmez une méthode permettant ce test.*

– Il est plus compliqué de vérifier qu'une formule n'est pas satisfiable. Une possibilité est de générer de façon aléatoire un certain nombre d'affectations et de vérifier qu'elles ne satisfont pas la formule.

Exercice 1.3 *Programmez une méthode qui réalise ce test sur une centaine d'affectations prises au hasard.*

– Sur quelles formules tester votre programme ? Il est pratique de disposer d'un générateur de formules. Par exemple, on peut programmer une fonction $G(n, m, p)$ qui génère une formule avec n clauses et m variables avec la propriété que :
  – le littéral x_j est présent dans la clause C_i avec probabilité $p/2$;
  – le littéral $\neg x_j$ est présent dans la clause C_i avec probabilité $p/2$;
  – les littéraux x_j et $\neg x_j$ ne sont jamais présents ensemble dans une clause C_i (et donc ils sont absents avec probabilité $(1 - p)$).

Exercice 1.4 *Programmez une procédure qui prend les paramètres (n, m, p, k), génère k formules en utilisant la fonction $G(n, m, p)$, applique la procédure DP pour déterminer la satisfiabilité et applique les méthodes développées dans les exercices 1.2 et 1.3 pour vérifier le résultat.*

1.6 Heuristique

Le choix d'une variable x dans la dernière étape peut avoir beaucoup d'influence sur la rapidité de la procédure. Une heuristique possible est de choisir x de sorte que le nombre de clauses dans lesquelles x apparaît multiplié par le nombre de clauses dans lesquelles $\neg x$

apparaît est maximal, et de tester $DP([1/x]A)$ d'abord s'il y a plus de clauses contenant x que de clauses contenant $\neg x$, et $DP([0/x]A)$ sinon.

Exercice 1.5 *1. Implanter cette stratégie dans la fonction DP. Soit DPH la fonction obtenue.*

2. Modifiez le code de la fonction DP et de la fonction DPH pour qu'elles retournent le nombre de fois que le pas de 'choix' $DP([0/x]A)$ or $DP([1/x]A)$ est exécuté.

3. Programmez une procédure qui prend les paramètres (n, m, p, k), génère k formules en utilisant la fonction $G(n, m, p)$, applique les procédures DP et DPH aux formules, vérifie qu'elles produisent le même résultat (satisfiable ou pas satisifiable) et pour chaque formule imprime le nombre de fois que le pas de 'choix' est exécuté par DP et DPH.

1.7 Le principe du pigeonnier (*pigeon principle*)

On dispose de m pigeons et de n nids. Le problème $P(m, n)$ a une solution si :
- Chaque pigeon a un nid.
- Chaque nid contient au plus un pigeon.

Il est évident que le problème n'a de solution que si m est inférieur ou égal à n, mais la vérification de ce fait par la méthode de Davis-Putnam peut s'avérer très coûteuse.

On écrit le problème du pigeonnier en CNF de la façon suivante :
- On introduit les variables $o_{i,j}$ pour $i \in [1..m], j \in [1..n]$. On interprète $o_{i,j}$ par *le pigeon i occupe le nid j.*
- On introduit la CNF
$$\bigwedge_{i=1,\ldots,m} \Big(\bigvee_{j=1,\ldots,n} o_{i,j} \Big)$$
qui exprime le fait que chaque pigeon doit se trouver dans un nid.
- On introduit la CNF
$$\bigwedge_{j=1,\ldots,n,\ i,k=1,\ldots,m, i<k} (\neg o_{i,j} \vee \neg o_{k,j})$$
qui exprime le fait que dans chaque nid il y a au plus un pigeon.
- On prend la conjonction des CNF (qui est encore une CNF !).

Exercice 1.6 *1. Programmez une fonction* **pigeon** *qui prend en argument deux entiers (m, n), et qui produit un fichier DIMACS contenant la formule CNF correspondant au problème $P(m, n)$.*

2. Testez vos programmes DP et DPH pour voir jusqu'à quelle valeur de m il parviennent à résoudre les problèmes $P(m, m)$ et $P(m + 1, m)$.