TP8 – Lemme de Schwartz-Zippel

Projet de programmation M1

24 Novembre 2015

Une opération assez utile en informatique est de tester l'égalité de deux polynômes à plusieurs variables P_1 et P_2. Cela est équivalent à tester si le polynôme $P_1 - P_2$ vaut le polynôme nul. Par exemple, est-ce que $(X_1 + X_1)^2 - 2X_1X_2 - X_1^2 + X_2^2 = 0$? Dans ce cas, il semble assez simple de résoudre ce problème : on développe le polynôme et l'on vérifie s'il est nul. Cela ne marche cependant pas si bien, si l'on prend le polynôme $(\sum_{i=1}^{n} X_i)^n$ qui a un nombre de terme exponentiel en n. Le problème dans toute sa généralité est appelé *Polynomial Identity Testing* (PIT) et est un problème difficile pour lequel on ne connaît pas d'algorithme exact efficace. Le lemme de Schwartz-Zippel est une approche probabiliste du problème. Supposons qu'on est un polynôme P de degré au plus d a une variable. On sait qu'il a au plus d racines. Si on l'évalue sur une valeur v aléatoire choisie uniformément dans un ensemble fini S, la probabilité pour que $P(v) = 0$ mais que $P \neq 0$ est au plus $d/|S|$. En faisant cette opération un certain nombre de fois, on peut construire un algorithme probabiliste pour PIT.

On peut prouver par induction que le lemme de Schwartz-Zippel fonctionne toujours avec plusieurs variables :

Théorème 1. *Soit $\mathbb{F}$ un corps et soit P un polynôme de $\mathbb{F}[X_1, \ldots, X_n]$ de degré au plus d, non-nul. Soit S un sous-ensemble fini de $\mathbb{F}$ et $v_1, \ldots, v_n \in S$ choisies uniformément. Alors*

$$\mathbb{P}[P(v_1, \ldots, v_n) = 0] \leq d/|S|.$$

1 Polynômes multivariés

On supposera que le polynôme que l'on veut étudier est décrit dans un fichier. La première ligne du fichier contiendra un entier N qui désigne le nombre de ligne qui va suivre. Ensuite, chaque ligne sera de la forme :

- `VAR i` pour indiquer que vous parler de la variable X_i
- `CONST k` pour indiquer que vous parler de la constante k (entière)
- `PLUS i j` pour indiquer que vous additionnez les polynômes des lignes i et j
- `TIMES i j` pour indiquer que vous multipliez les polynômes des lignes i et j

La dernière ligne du fichier sera le polynôme représenté par ce fichier. Par exemple, le polynôme $(X_0 + X_1) \times 2$ pourra être écrit :

```
5
VAR 1
VAR 0
CONST 2
PLUS 0 1
TIMES 3 2
```

Télécharger le fichier `poly.c` sur ma page (si vous n'avez pas accès à internet, j'ai le fichier avec moi sur une clé). Il contient la définition d'une structure C pour représenter les polynômes comme des circuits. Il contient aussi deux fonctions : une fonction d'évaluation d'un polynôme et une fonction pour créer une nouvelle porte dans le circuit qui vous sera utile pour la suite. Lisez le code et comprenez-le.

Exercice 1. On va commencer par transformer un fichier en un `polynome`, comme défini dans `poly.c`.

1. Écrire une fonction `int read_int(FILE *f)` qui lit des caractères dans le fichier tant que ce sont des chiffres. La fonction renvoie ensuite le nombre qui vient d'être lu. Par exemple, si le fichier `f` est ainsi : `j'ai .103 ans` (où le . est le curseur dans le fichier), votre fonction devra renvoyer 103 et positionner le curseur juste avant le 'a'. On rappelle que la fonction `char fgetc(FILE *f)` lit un caractère de `f`, le renvoie et déplace le curseur vers la droite.
2. Écrire une fonction `int read_operation(FILE *f)` qui trouve quelle opération parmi `VAR, CONST, TIMES, PLUS` se trouve sous le curseur du fichier. Par exemple, si le fichier `f` est ainsi (le . désigne le curseur) : `.PLUS 0 2`, votre fonction doit renvoyer la constante PLUS (qui vaut 0, voir `poly.c`, ligne 4). La fonction renverra -1 si le mot ne correspond à aucun des cas précédents.
3. Écrire une fonction `polynomial *file_to_poly(FILE *f)` qui lit le fichier `f` et reconstruit en mémoire (utilisez bien la fonction `create_node` pour l'allocation dynamique) un polynôme de type `polynomial` correspondant au polynôme décrit par le fichier.

2 Polynomial Identity Testing

Exercice 2. Notre but est de savoir si le polynôme est identiquement nul ou pas.

1. Écrire une fonction `int nvar(polynomial *p)` qui trouve l'indice de la plus grande variables apparaissant dans `p`. Par exemple si $P = x_1 + x_3x_2$, votre fonction doit renvoyer 3.
2. Écrire une fonction `int pit(polynomial *p, int min, int max, int k, int t[])` qui évalue le polynôme `p` sur des valeurs tirées uniformément entre `min` et `max`. L'opération est répétée `k` fois. Si une de ces évaluations ne donne pas 0 alors `p` n'est pas identiquement nul et votre fonction doit renvoyer 0 et écrire dans t le point où `p` ne s'annule pas. Si toutes ces évaluations renvoient 0 alors votre fonction doit renvoyer 1 (le polynôme est probablement identiquement nul).
3. Soit P fixé de degré au plus d. Évaluer la probabilité que votre fonction `pit` se trompe.