

TP preuve mécanisée

Approches formelles pour la vérification de programmes, Master CNS

2020–21

Exercice 1 : SAT

On va d'abord se servir de l'outil `glucose` pour démontrer des problèmes en logique propositionnelle.

Le format d'entrée de `glucose` est DIMACS. Il s'agit d'une première ligne qui décrit le problème

`p cnf i j`

où i est le nombre de variables propositionnelles du problème et j le nombre de clauses ; puis d'une ligne pour chaque clause. Chaque variable propositionnelle est représentée par un entier entre 1 et i . La négation de k est représentée par $-k$. Une clause est écrite comme la succession de ses littéraux (variable propositionnelle ou négation de celle-ci) séparés par un espace, suivi d'un 0.

1. Utiliser `glucose` pour montrer que le problème $\{p \vee q; \neg p \vee q; p \vee \neg q; \neg p \vee \neg q\}$ est insatisfiable.

2. Utiliser `glucose` pour trouver un modèle pour l'ensemble de clauses $\{a \vee b \vee c; \neg a \vee \neg b; \neg b \vee \neg c; \neg c \vee \neg a\}$
Comment obtenir une autre solution ? (Indication : nier la solution obtenue.)

3. On rappelle le problème du club écossais :
Pour constituer un club on a énoncé le règlement suivant :

Article premier : Tout membre non écossais porte des chaussettes oranges.

Article second : Tout membre porte un kilt ou ne porte pas de chaussettes oranges.

Article troisième : Les membres mariés ne sortent pas le dimanche.

Article quatrième : Un membre sort le dimanche ssi il est écossais.

Article cinquième : Tout membre qui porte un kilt est écossais et marié.

Article sixième : Tout membre écossais porte un kilt.

Formaliser ce problème en logique propositionnelle, mettez en forme normale conjonctive, et utiliser `glucose` pour montrer que ce club n'a pas de membre.

4. On supprime l'article sixième. Utiliser `glucose` pour caractériser un membre possible du club.

Exercice 2 : SMT

On va maintenant utiliser des solveurs modulo théorie, en l'occurrence Z3, CVC4 et alt-ergo, pour démontrer des problèmes.

Exercice 2.1 : Symboles de fonction non interprétés

1. Étant donnés trois constantes a, b, c et un symbole de fonction unaire f , on veut montrer qu'à partir des trois hypothèses $b = d$, $f(b) = d$ et $f(d) = a$ on peut déduire $a = b$.

Dans un fichier en `.smt2`, on va d'abord dire qu'on travaille sans quantificateurs avec des symboles de fonction non interprétés.

```
(set-logic QF_UF)
```

On déclare ensuite une sorte pour les termes

```
(declare-sort term 0)
```

et les symboles de fonctions et constantes

```
(declare-const a term)
```

```
(declare-const b term)
```

```
(declare-const c term)
```

```
(declare-fun f (term) term)
```

On peut alors ajouter le problème en ajoutant les formules avec `assert`, comme par exemple

```
(assert (= (f b) c))
```

On niera la conclusion pour faire une preuve par réfutation.

On demande alors au solveur de vérifier la satisfiabilité :

```
(check-sat)
```

Écrire le fichier et tester avec Z3, CVC4 et alt-ergo.

2. On peut avoir des formules avec des connecteurs. On considère maintenant 5 constantes a, b, c, d, e . On suppose :

$$c = a \vee c = b$$

$$d = a \vee d = b$$

$$e = a \vee e = b$$

et on veut montrer $c = d \vee c = e \vee d = e$.

Écrire le fichier et tester avec Z3, CVC4 et alt-ergo.

Exercice 2.2 : Arithmétique

On se place maintenant dans la théorie de l'arithmétique linéaire (`QF_LIA`). On déclarera des constantes de type `Int`.

1. À l'aide d'un solveur SMT, montrer que le problème suivant n'a pas de solution entière :

$$\begin{aligned}x + y &= 1 \\x - y &= 2\end{aligned}$$

2. Montrer que le problème suivant à des solutions entières :

$$\begin{aligned}x + y &\leq 1 \\x - y &\geq 2\end{aligned}$$

On pourra afficher une solution en rajoutant la ligne :

`(get-value (x y))`

3. On veut maintenant montrer le résultat suivant :

$$n = qb + a \Rightarrow n = (q + 1)b + (a - b)$$

L'arithmétique linéaire ne suffit plus, il faut passer à l'arithmétique non linéaire

`(set-logic QF_NIA)`

Écrire le fichier correspondant et tester avec Z3, CVC4 et alt-ergo.

Exercice 3 : Logique du premier ordre

Pour pouvoir utiliser des axiomes quantifiés universellement, on doit passer à la logique du premier ordre. On utilise le prouveur E qui prend en entrée des fichiers au format TPTP à condition de l'appeler ainsi :

`$ eprover --tstp-format fichier.p`

On va démontrer le problème suivant : on considère deux relations binaires R et S (qui seront représentées par des symboles de prédicat). On suppose que R est irréflexive et transitive, et que S est la clôture réflexive de R . On veut montrer que S est antisymétrique. On aura donc les formules :

$$\begin{aligned}\forall x. \neg R(x, x) & && \text{(irréflexivité)} \\ \forall x y z. (R(x, y) \wedge R(y, z)) \Rightarrow R(x, z) & && \text{(transitivité)} \\ \forall x y. S(x, y) \Leftrightarrow (R(x, y) \vee x = y) & && \text{(clôture réflexive)} \\ \forall x y. (S(x, y) \wedge S(y, x)) \Rightarrow x = y & && \text{(antisymétrie)}\end{aligned}$$

Écrire le fichier TPTP correspondant et tester avec E.

Exercice 4 : Coq

Exercice 4.1 : Logique du premier ordre

On considère la signature (symboles de fonction et de prédicat) suivante :

Variable A : Prop.

Variable B : Prop.

Variable P : nat -> Prop.

Variable Q : nat -> Prop.

(nat est le type des entiers naturels.)

Montrer les théorèmes suivants (sans utiliser la tactique `tauto!`) :

1. $A \rightarrow B \rightarrow A$
2. $(\neg A \vee B) \rightarrow (A \rightarrow B)$
3. $(\text{forall } x, P\ x) \rightarrow (\text{exists } x, P\ x)$
4. $(\text{exists } x, P\ x \vee Q\ x) \leftrightarrow ((\text{exists } x, P\ x) \vee (\text{exists } x, Q\ x))$

NB : pour le dernier, on pourra utiliser la tactique `split` pour casser l'équivalence en deux implications.

Exercice 4.2 : Arithmétique

On va maintenant montrer des théorèmes en arithmétique entière. On va pour cela utiliser des bibliothèques Coq contenant les définitions des entiers relatifs, des théorèmes et des tactiques permettant de faire des preuves :

```
Require Import ZArith.
```

```
(* Definitions of relative integers Z *)
```

```
Require Import Psatz.
```

```
(* Tactics lia, nia *)
```

```
Open Scope Z_scope.
```

```
(* Consider +, *, etc. as operators on Z *)
```

Montrer les théorèmes suivants :

1. $\text{forall } x\ y, (x + 1) * (x + y) = x * x + (1 + y) * x + y$
2. $\text{forall } a\ b\ n, (a > b \wedge (\text{exists } q, n = b * q + a)) \rightarrow (\text{exists } q', n = b * q' + (a - b))$