Software Verification

2024-2025

TP1: SMT

Vincent Penelle

Points abordés

- Rappel OCaml
- Z3
- API Z3 de OCaml
- Codage de solver d'un jeu (Keen)

Exercice 1: Rafraîchissement/Introduction à OCaml

Si OCaml vous est un langage familier, vous pouvez passer à la suite. Sinon, ouvrez le document ≪Tuto-OCaml.pdf≫ (ou ici[1]). Il vous présentera la plupart des concepts et syntaxes du langage dont nous aurons besoin durant le cours.

Exercice 2: Introduction à Z3

Z3 est un SMT-solveur de Microsoft Research. Il permet de décider la satisfiabilité de formules du premier ordre passées en paramètre. Ces formules peuvent parler de booléens, d'entiers, de réels, de tableaux, de fonctions abstraites, de bit-vectors, etc.

En cas de satisfiabilité, il permet de récupérer une affectation de variables satisfaisant la formule. En cas d'insatisfiabilité, il permet de récupérer une preuve d'indécidabilité.

Dans le cadre de ce TP, nous nous concentrerons sur la manipulation des entiers, et nous serons amenés à récupérer un modèle (cette partie vous sera donnée). Nous n'utiliserons pas l'intégralité de ses fonctionnalités. Nous l'utiliserons via son API OCaml, mais vous pouvez éventuellement jeter un œil au tutoriel suivant qui concerne Z3 lui-même[2], en regardant spécifiquement les sections ≪Basic Commands≫ (avant ≪Using Scopes≫), Propositional Logic et Arithmetic.

Il est tout à fait facultatif de regarder en détail ce tutoriel, par la suite, on ne passera que par l'API Ocaml, l'exercice suivant est donc plus utile. Vous pouvez cependant le garder comme référence de ce qui est spécifique à Z3.

Exercice 3: L'API OCaml de Z3 Nous nous tournons maintenant vers l'API de Z3 pour OCaml que nous utiliserons dans le reste du cours.

La documentation de l'API Z3 est accessible ici[3].

[1] `Tuto-Ocaml.html`

[2] `https://microsoft.github.io/z3guide/docs/logic/intro`

[3] `http://z3prover.github.io/api/html/ml/Z3.html`

1. Ouvrez le fichier `z3_ml_example.ml`. Il contient des exemples d'utilisation de Z3 via OCaml. Compilez-le avec la commande `make z3_ml_example.byte`, et lancez le résultat avec `./z3_ml_example.byte`. Observez notamment comment construire des formules booléennes et arithmétiques.

Exercice 4: Keen Keen est un jeu de logique que vous trouverez là[4]. Il consiste à remplir une grille carrée de coté n avec des entiers de 1 à n tels que chaque entier n'apparaisse qu'une fois sur chaque ligne et sur chaque colonne. De plus, la grille est divisée en zones auxquelles sont associées une opération et un résultat : si l'opération est une addition (resp. multiplication), alors la somme (resp. produit) des nombre de la zone donne le résultat. Pour la soustraction et la division, les zones font nécessairement 2 cases et l'une des deux différences (resp. quotients) possibles doit être le résultat. Notez que dans le cas d'une division, le reste doit être nul.

Vous allez réaliser un solveur pour ce jeu. Tous les appels à Z3 et le parsing du jeu sont déjà implémentés, il ne vous reste plus qu'à créer la formule qui encode les contraintes du jeu. Chaque case sera représentée par une variable entière (qui représente le contenu de la case). Les questions 2 à 5 de cet exercice sont là pour vous guidez dans la conception de votre réduction (car c'en est une).

1. Éxécutez la commande `make doc` et regardez la documentation produite pour KeenSolver. Vous aurez à implémenter la fonction `game_formula`.

2. Donnez une formule encodant le fait que toutes les cases de la ligne i sont différentes.

3. Donnez une formule qui assure que la somme des cases d'une zone est égale à un résultat (vous choisirez une zone et un résultat quelconque pour l'exemple).

4. Donnez une formule qui assure que l'une des différence possibles entre deux cases est égale à un résultat (vous choisirez une zone et un résultat quelconque pour l'exemple).

5. Faites de même pour la multiplication (d'une zone) et la division (de deux cases – attention au reste qui doit être nul).

6. Implémentez la fonction `game_formula`, et testez le résultat sur les exemples fournis (attention, certains peuvent être lents). Notamment, notez que les exemples sans multiplication sont résolus bien plus rapidement que ceux avec (n'essayez pas de résoudre la grille 9×9 avec multiplication!).

[4] `https://www.chiark.greenend.org.uk/~sgtatham/puzzles/js/keen.html`