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: Tutorial/Introduction to OCaml

If OCaml is familiar to you, you can skip this step. Otherwise, open the document "Tuto-Ocaml-en.pdf" (or there[1]). It will explain you most of the concepts and syntaxes of the language useful for this course. The goal isn't to spend the whole session on this tutorial, but rather for it to serve as a reference for the whole course.

Exercice 2: Introduction to Z3

Z3 is an SMT-solver from Microsoft Research. It allows to decide the satisfiability of first-order formulæ given as parameter. These formulæ may use booleans, integers, real numbers, arrays, abstract functions, bit vectors, etc.

In case a formula is satisfiable, it is possible to get back a valuation satisfying it. In case a formula is unsatisfiable, it can give a proof of unsatisfiability.

In this exercise session, we will focus on integers handling, and we'll have to get back a valuation (called model) – this part will be done for you. We won't use all features from Z3. We'll use it through its OCaml API, but you may look very quickly at the following tutorial on Z3 itself[2], mainly on sections ≪Basic Commands≫ (before ≪Using Scopes≫), Propositional Logic and Arithmetic.

It is completely ok not to look at it, and in the rest of this course, we'll only use the OCaml API; the next Exercise is thus more useful. You can however use this tutorial as a reference for what is specific to Z3.

Exercice 3: The OCaml API of Z3

We now focus on the OCaml API of Z3.

Its documentation is available here[3].

[1] `Tuto-Ocaml-en.html"`

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

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

1. Open `z3_ml_example.ml`. It contains use examples of Z3 through OCaml. Compile it with `make z3_ml_example.byte`, and execute it with `./z3_ml_example.byte`. Observe specifically how to construct boolean and arithmetic formulæ.

Exercice 4: Keen Keen is a logic game which you can find here[4]. Its goal is to fill a square grid of size $n \times n$ with integers from 1 to n such that every integer appear once on each row and each column. Moreover, the grid is divided between zone to which is associated an operator and a result: if the operation is an addition (resp. a multiplication), the sum (resp. product) of all numbers from the zone must yield the result. Substraction and division can only be used on zones of size 2, and one of the difference (resp. quotient) must yield the result. Notice that for divisions, the remainder must be zero.

Your task is to implement a solver for this game using Z3. All calls to Z3, and the game parsing are given to you, you only have to create the formula representing the constraint of the input game. Every cell will be represented by a variable that can hold an integer (its value).

To control your implementation, an executable with the solution, `keen-solution.d.byte` is given. Use it to compare the solution you obtain.

Questions 2 through 5 are there to guide you in the conception of your reduction (it is one). You don't have to treat them separately. Do not hesitate to print the formula you produce with `Z3.Expr.to_string`. On small examples, the formulæ should stay small.

1. Run `make doc` and browse the produced documentation for the module KeenSolver. You have to implement function `game_formula`.

2. Give a formula encoding the fact that all cell of a row i are different.

3. Give a formula encoding that the sum of cells of some zone is equal to some result (which you can take arbitrarily while you are conceiving it).

4. Give a formula encoding that one of the twe possible substractions between two cells is equal to some result.

5. Do the same for multiplication and division (keep in mind the remainder must be zero).

6. Implement `game_formula`, and test the result on provided examples. Warning, it can be slow, mainly for examples where multiplication and division are present (don't try to solve the 9×9 grid with multiplication, but you will notice that its multiplication-less variant terminates in reasonable time).

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