Few exercises in Z3

This PDF consists of a collection of short exercises directly in Z3 which I did in my computational logic class. It is not an introduction to Z3 or to first order logic.

Before looking at the assignments I briefly mention two important things about theorem provers.

Proof by refutation : How do we prove a statement in the form of implication in software like Z3? Let's say we have assumptions $A_1, ..., A_n$ and we want to prove B. We tell the software to assume $A_1, ..., A_n, \neg B$ are true and let it prove a *contradiction*. This means that one of the assumptions $A_1, ..., A_n, \neg B$ must be false. If we assume $A_1, ..., A_n$ to be true, then we get that $\neg B$ has to be false. In other words, we proved that $A_1, ..., A_n \Rightarrow B$.

Skolem variables : Existential quantification is a tricky thing. When dealing with formulas involving existential quantifier, the software performs a so called *skolemization*. We get rid of the quantifier and replace the bounded variable with a fresh constant. For example the formula $\exists x : x > 1$ for $x \in \mathbb{R}$ is encoded in Z3 as

```
(declare-const c Real)
(assert (> c 1))
```

Exercise 1 Model the statement $(A \cup B \subseteq C \wedge B \setminus A \neq \varnothing) \Rightarrow A \subset C$ in first order logic and solve it.

We can express the idea of a set by a function which outputs a proposition. Let's say we have a set A and an element x. By $A(x)$ we mean that x is an element of the set A. Equipped with a way to encode sets in first order logic, we start by encoding the formula $A \cup B \subseteq C$ as

$$\forall x : (A(x) \vee B(x)) \Rightarrow C(x).$$

Now we move to the statement that the set $B \setminus A$ is not an empty set. If something is not an empty set then it has at least one element. We write the formula and immediately skolemize it

$$\exists x : B(x) \wedge \neg A(x) \quad \rightarrow \quad B(d) \wedge \neg A(d).$$

The last statement says that the set A is a *proper* subset of C. This means that A is a subset of C but A is not equal to C. This means that there is at least one element in the set C which is not in the set A. The first part of the conjunction says that A is a subset of C.

$$(\forall x : A(x) \Rightarrow C(x)) \wedge (\exists x : C(x) \wedge \neg A(x)).$$

The previous formula is the conjecture which needs to be negated. Thus the input to the solver is the following formula which is again skolemized

$$(\exists x : A(x) \wedge \neg C(x)) \vee (\forall x : A(x) \vee \neg C(x)) \quad \rightarrow \quad (A(e) \wedge \neg C(e)) \vee (\forall x : A(x) \vee \neg C(x)).$$

Z3 requires for all the variables to be of some sort. We declare an uninteresting sort `T` and say that the constants `d` and `e` are of this sort `T`.We also declare functions `A,B` and `C` representing the sets.

```
(declare-sort T)
(declare-const d T)
(declare-const e T)
(declare-fun A (T) Bool)
(declare-fun B (T) Bool)
(declare-fun C (T) Bool)
```

We then declare the three formulas using the `assert` command and then run the program. The output is `unsat`. This means the set of formulas is unsatisfiable or in other words, we proved that the implication holds.

```
(assert (forall ((x T)) (=> (or (A x) (B x)) (C x))))
(assert (and (B d) (not (A d))))
(assert
  (or
    (and (A e) (not (C e)))
    (forall ((x T)) (or (A x) (not (C x))))
  )
)
(check-sat)
```

Exercise 2 Find all solutions to the functional equation $f(x+y) = xf(y) + yf(x)$ where $x, y \in \mathbb{R}$.

First we inspect what happens for zero. We have $f(0+0) = 0f(0) + 0f(0)$. This means that $f(0) = 0$. What if we set y to zero? Then we get $f(x+0) = xf(0) + 0f(x)$. We know that $f(0) = 0$ which means $f(x) = 0$ for all $x \in \mathbb{R}$. This seems to be the only solution but how do we prove it? Suppose there exists a solution that is not zero everywhere. This means there exists a point c such that $f(c) \neq 0$. But then $f(c+0) = cf(0) + 0f(c)$ which we know has to be zero. Thus the functional equation admits only the trivial solution.

We can ask Z3 to find *some* solution of the functional equation. Z3 outputs `sat` meaning the functional equation is satisfiable by some function and also offer such function `f ((x!0 Real)) Real 0.0`. This means that `f` is a real function and outputs zero for all x.

```
(declare-fun f (Real) Real)
(assert
  (forall ((x Real) (y Real))
     (! (= (f (+ x y)) (+ (* x (f y)) (* y (f x))))
        :pattern ((f x) (g y))
     )
  )
)
(check-sat)
(get-model)
```

How can we prove that this is the only solution using Z3? We prove it the same way as we prove it on paper. We introduce a skolem constant `c` and say write the assumption that the function representing solution is not equal to zero at this point. The problem is that Z3 runs out of time for the next code and answers `unknown`.

```
(declare-fun f (Real) Real)
(declare-const c Real)
(assert
  (forall ((x Real) (y Real))
    (= (f (+ x y)) (+ (* x (f y)) (* y (f x))))
  )
)
(assert (not (= 0 (f c))))
(check-sat)
```

Z3 was not able to come up with a right instantiation for the `forall` clause. This is because formulas with quantifiers are *hard*. One of the tricks Z3 uses is called pattern based matching. It means that it instantiate the formula involving `forall` with constants found in other clauses and tries to use these instances to prove the statement. We disable the automatic process and tell Z3 to look for specific patterns.

```
(set-option :smt.auto-config false) ; disable automatic self configuration
(set-option :smt.mbqi false) ; disable model-based quantifier instantiation
```

We need to persuade Z3 that it should instantiate the `forall` clause with $x = c$ and $y = 0$ as we did. The problem is that the constants must be part of some function. This is true for c as we have $f(c)$ in the last clause. For 0 we introduce a dummy function `g` where `g` represents a predicate and write down $g(0)$. We also specify the pattern which Z3 should use using `:pattern`. For the following code Z3 swiftly outputs `unsat`. We proved that the trivial solution is the only one.

```
(declare-fun f (Real) Real)
(declare-fun g (Real) Bool)
(declare-const c Real)
(assert
  (forall ((x Real) (y Real))
     (! (= (f (+ x y)) (+ (* x (f y)) (* y (f x))))
        :pattern ((f x) (g y))
     )
  )
)
(assert (g 0))
(assert (not (= 0 (f c))))
(check-sat)
```