Using the SMT solver Z3

Z3 is a state-of-the art theorem prover from Microsoft Research. It can be used to check the satisfiability of logical formulas over one or more theories. It is a low level tool that is often used as a component in the context of other tools that require solving logical formulas.

How do I write scripts for Z3?

The Z3 input format is an extension of the one defined by the SMT-LIB 2.0 standard. A Z3 script is a sequence of commands. Z3 maintains a **stack** of user provided formulas and declarations. These are the **assertions** provided by the user.

- The command **declare-const** declares a constant of a given type.
- The command **declare-fun** declares a function.
- The command **assert** adds a formula into the Z3 internal stack.

The set of formulas in the Z3 stack is **satisfiable** if there is an interpretation (for the user declared constants and functions) that makes all asserted formulas true. When the command **check-sat** returns **sat**, the command **get-model** can be used to retrieve an interpretation that makes all formulas on the Z3 internal stack true.

Exercise 1. Propositional and Predicate Logic in Z3:

We can check propositional logic assertions using functions to represent the propositions x and y. E.g. copy and paste the following into the online interface to Z3 at http://www.rise4fun.com/z3 and press the play button. This checks to see if ¬(x ∧ y) ≡ (¬ x ∨ ¬ y):

```
(declare-fun x () Bool)
(declare-fun y () Bool)
(assert ( = (not(and x y)) (or (not x)(not y))))
(check-sat)
```

The command **check-sat** determines whether the current formulas on the Z3 stack are satisfiable or not. If the formulas are satisfiable, Z3 returns **sat**. If they are not satisfiable (i.e., they are **unsatisfiable**), Z3 returns **unsat**. Z3 may also return **unknown** when it can't determine whether a formula is satisfiable or not.

Using Z3, determine if the following formulae are true:

(a) ¬(x ∨ y) ≡ (¬ x ∧ ¬ y)
(b) (x ∧ y) ≡ ¬ (¬ x ∨ ¬ y)

Note 1
We can also use propositional logic using constants and we can name the theorems that we want to work with. E.g.

```
(declare-const a Bool)
(declare-const b Bool)
(define-fun demorgan () Bool
    (= (and a b) (not (or (not a) (not b)))))
(assert (not demorgan))
(check-sat)
```

When the command check-sat returns **sat**, the command **get-model** can be used to retrieve an interpretation that makes all formulas on the Z3 internal stack true.

Note 2:
We can also check predicate logic assertions in Z3 e.g. To check if
∀p:bool, f(p) => ¬∃q:Bool, ¬f(q)

```
(declare-fun f (Bool) Bool)
(assert
  (=>
    (forall ((p Bool)) (f p))
    (not(exists ((q Bool)) (not (f q))))))
  (check-sat)
  (get-model)
```

Exercise 2. Using Z3 for Arithmetic:

In the following script, the first asserted formula states that the constant `a` must be greater than 10. The second one states that the function `f` applied to `a` and `true` must return a value less than 100.

```
(declare-const a Int)
(declare-fun f (Int Bool) Int)
(assert (> a 10))
(assert (< (f a true) 100))
(check-sat)
```

Add the assertion "*(get-model)*" to the script above. Explain what you understand from the output.

Exercise 3. Solving Equations using Z3

The following script allows us to solve the two given equations, finding suitable values for x and y.

```
(declare-const x Int)
(declare-const y Int)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
```

(a) What does the output from this script tell us?
(b) Add (get-model) after each occurrence of (check-sat). Explain the output.
(c) Using Z3, can you find a solution for x and y in the following equations:

$$3x + 2y = 36$$

$$5x + 4y = 64$$

Exercise 4. Using Z3 as a calculator

Z3 also has support for division, integer division, modulo and remainder operators. E.g.

```
(declare-const a Int)
(declare-const r1 Int)
(declare-const r2 Int)
(declare-const r3 Int)
(assert (= a 10))
```

```
(assert (= r1 (div a 4))); integer division
(assert (= r2 (mod a 4))); mod
(assert (= r3 (rem a 4)));remainder
(declare-const b Real)
(declare-const c Real)
(assert (>= b (/ c 3.0)))
(assert (>= c 20.0))
(check-sat)
(get-model)
```

Using Z3 calculate the following and store the result in constants a1, a2 and a3:
(a) 16 mod 2
(b) 16 divided by 4
(c) The remainder after dividing 16 by 5

Exercise 4. Using the stack:

Z3 maintains a global stack of declarations and assertions. The command `push` creates a new scope by saving the current stack size. The command `pop` removes any assertion or declaration performed between it and the matching `push`. The `check-sat` and `get-assertions` commands always operate on the content of the global stack.

```
(declare-const x Int)
(declare-const y Int)
(declare-const z Int)
(push)
(assert (= (+ x y) 10))
(assert (= (+ x (* 2 y)) 20))
(check-sat)
(pop) ; remove the two assertions
(push)
(assert (= (+ (* 3 x) y) 10))
(assert (= (+ (* 2 x) (* 2 y)) 21))
(check-sat)
```

(a) What does the output from this script tell us?
(b) Add (get-model) after each occurrence of (check-sat). Can you explain the output?

Exercise 5. Using Z3 at the restaurant:

(a) Encode the following menu choices into Z3 and determine what a customer could buy using exactly $15.05

- Mixed fruit $2.15
- French Fries $2.75
- Side Salad $3.35
- Hot Wings $3.55
- Mozzarella Sticks $4:20
- Sampler Plate $5:80

(b) Is it possible to order everything on the menu?
(c) Does this problem have only one solution? If it does not, what are the other solutions?

For further assistance see the Z3 tutorial at http://www.rise4fun.com/Z3/tutorial/guide