A very short introduction SMT, Symbolic execution, DO178

Franco Raimondi

Department of Computer Science
School of Science and Technology
Middlesex University
http://www.rmnd.net

A very short introduction SMT, Symbolic execution, DO178

Plan

- What is an SMT solver, with examples.
- What is symbolic execution.
- Certification: DO178 very quick overview.

What is an SMT solver

SMT = satisfiability modulo theories. An SMT problem is a *decision problem* for logical formulae expressed in a combination of theories (in the sense defined in our first week, revision of predicate logic).
Intuitively, *"An SMT instance is a generalization of a Boolean SAT instance in which various sets of variables are replaced by predicates"*. For instance:

$$((x + y) = 10) \wedge ((x + 2y) = 20)$$

Given x and y Int, is this SAT or unsat? What about this other one?

$$((x - y) = 10) \wedge ((x + 2y) = 2)$$

Let's install an SMT solver (Z3)

```
git clone https://github.com/Z3Prover/z3.git
cd z3
python scripts/mk_make.py --java
cd build/
make
sudo make install
```

(this enables Java bindings, see below)

A simple example

Taken from `http://smtlib.github.io/jSMTLIB/SMTLIBTutorial.pdf`, using SMT language (Lisp-like)

```
(set-option          :print-success          false)
(set-option          :produce-models          true)
(set-option          :interactive-mode          true)
(set-logic          QF_LIA)
(declare-fun    x  ()  Int)
(declare-fun    y  ()  Int)
(declare-fun    z  ()  Bool)
(declare-fun    w  ()  Bool)
(assert          (= (+ x (* 2 y)) 20))
(assert (= (- x y) 2))
(assert (and z w  (> (- x y) 1)))
(check-sat)
(get-value (x y z w))
```

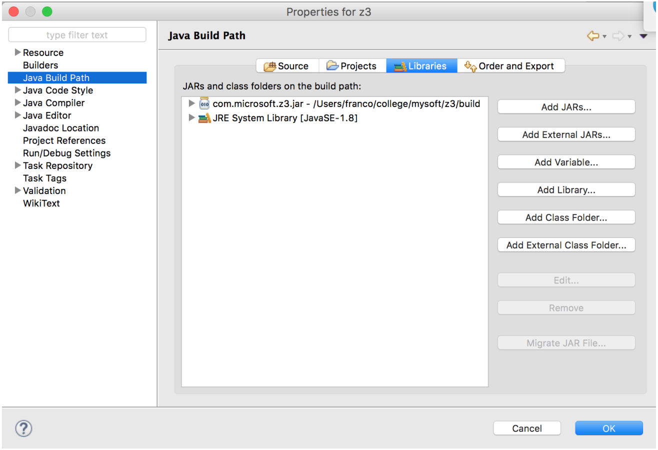
Run it with ./*z3filename* (in build/)

Use Z3 from Java

```java
import com.microsoft.z3.*;

public class SimpleTest {

public static void main (String[] args) {
 HashMap<String, String> cfg = new HashMap<String, String>();
 cfg.put("model", "true");
 Context ctx = new Context(cfg);

 IntExpr x = ctx.mkIntConst("x");
 IntExpr y = ctx.mkIntConst("y");
 IntExpr one = ctx.mkInt(1);
 IntExpr two = ctx.mkInt(2);

 System.out.println("model for: x < y + 1, x > 2");
 model = check(ctx, q, Status.SATISFIABLE);
 System.out.println("x = " + model.evaluate(x, false) +
    ", y =" + model.evaluate(y, false));
 // [...]
}
```

Eclipse set-up

Add .jar file generated with the `--java` option above:



Properties for z3

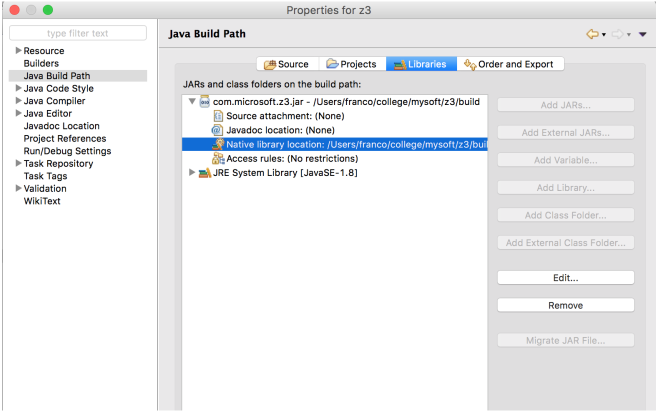
Java Build Path

JARs and class folders on the build path:

- com.microsoft.z3.jar - /Users/franco/college/mysoft/z3/build
- JRE System Library [JavaSE-1.8]

Eclipse set-up

Add native library, point to z3/build:



Properties for z3

Java Build Path

JARs and class folders on the build path:

- com.microsoft.z3.jar - /Users/franco/college/mysoft/z3/build
  - Source attachment: (None)
  - Javadoc location: (None)
  - Native library location: /Users/franco/college/mysoft/z3/buil
  - Access rules: (No restrictions)
- JRE System Library [JavaSE-1.8]

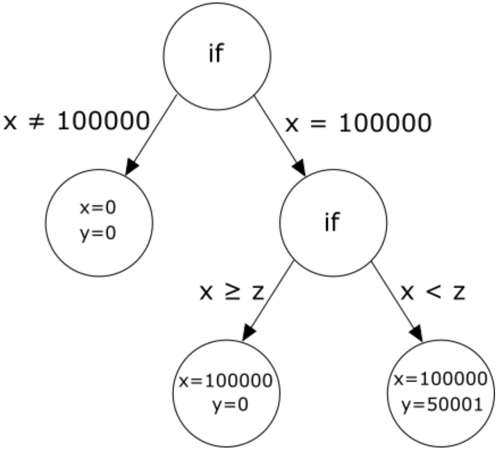
Concolic testing

Concolic = Con(crete) + (Symb)olic.
Consider the following example (source: WIkipedia):

```
void f(int x, int y) {
  int z = 2*y;
  if (x == 100000) {
    if (x < z) {
      // Some nasty error here
    }
  }
}
```

How to reach the nasty error?

Use an SMT solver to generate test cases



Each leaf is an assignment to variables for a possible test.

Testing?

We have moved from model checking to testing. The reason is that safety-critical software (airplanes, medical devices, autonomous cars, automated systems on trains, etc.) need to be *certified*.
Example: DO178, *Software Considerations in Airborne Systems and Equipment Certification*. The guideline convers all stages of software development, including *verification*. Verification is achieved through *testing*.
Different levels of failure conditions:

Level	Failure condition	Rate
A	Catastrophic	1.0E-9/hour
B	Hazardous	1.0E-7/hour
C	Major	1.0E-5/hour
D	Minor	1.0E-3/hour
E	No Effect	n/a

In summary (1)

- We have seen various tools: picosat, Spin, NuSMV, JPF, Z3.
- We have seen libraries: ltl2buchi, cudd
- Many many more tools exist: see `http://www.adacore.com/` for a tool that is used in industry to certify software. It includes SAT and SMT solvers both to prove properties and to generate test cases.

Conclusion

In the past lectures:

- Propositional logic and SAT solvers
- Predicate logic
- LTL: syntax, semantics, LTL2buchi
- CTL: syntax, semantics, labelling algorithm.
- CTL model checking using Ordered Binary Decision Diagrams (OBDDs).
- Tools: Spin for LTL, NuSMV for CTL.
- Other tools and libraries: picosat, Cudd, ltl2Buchi, JPF, Z3.