softeng.fe.up.pt

GenT: A Tool for the Automatic Generation of Unit Tests from Algebraic Specifications using Alloy and SMT (http://gloss.di.fc.ul.pt/quest)

João Pascoal Faria, FEUP/INESC TEC (jpf@fe.up.pt)

(with Francisco Andrade, Francisco Silva, Tiago Campos, Ana Paiva, and Antónia Lopes)

MAPI, 3 December 2014

Agenda

- Introduction
  - Algebraic Specification of Abstract Data Types (ADTs)
  - Examples in ConGu
  - Automated Conformance Checking & Testing
- Base approach (GenT v1)
  - Overview
  - Example
- Advanced features (GenT v2)
  - Support for unbounded data types
  - Test generation for the equals method
  - Test generation for invalid inputs
  - Exclusion of unsatisfiable test goals
- Experimental Results
- Demo
- Conclusions & Future work

Introduction
Algebraic Specification of Abstract Data Types (ADTs)

Formal algebraic specification languages are well suited for specifying abstract data types (ADTs)

- Provide the highest possible level of **abstraction**: semantics of operations is specified through axioms, without saying anything about the internal data representation

- formal specs **remove ambiguity** usually present in natural language

- formal specs are **automatically analyzable** (as will be shown here)

Introduction
Example in ConGu

```
specification Stack[Element]
  sorts
    Stack[Element]
  constructors
    make: -->Stack[Element];
    push: Stack[Element] Element --> Stack[Element];
  observers
    peek: Stack[Element] -->? Element;
    pop: Stack[Element] -->? Stack[Element];
    empty: Stack[Element];
  domains
    S: Stack[Element];
    peek(S) if not empty(S);
    pop(S) if not empty(S);
  axioms
    S: Stack[Element];
    E: Element;
    peek(push(S,E))=E;
    pop(push(S,E))=S;
    empty(make());
    not empty(push(S,E));
end specification
```

Introduction
A more complex example

```
specification SortedSet[TotalOrder]
 sorts
   SortedSet[Orderable]
 constructors
   empty: --> SortedSet[Orderable];
   insert: SortedSet[Orderable] Orderable --> SortedSet[Orderable];
 observers
   isEmpty: SortedSet[Orderable];
   largest: SortedSet[Orderable] -->? Orderable;
   ...
 domains
   S: SortedSet[Orderable];
   largest(S) if not isEmpty(S);
 axioms
   E, F: Orderable;  S: SortedSet[Orderable];
   largest(insert(S, E)) = E if isEmpty(S);
   largest(insert(S, E)) = E if not isEmpty(S) and geq(E, largest(S));
   largest(insert(S, E)) = largest(S)  if not isEmpty(S) and not geq(E, largest(S));
   ...
end specification
```

```
specification TotalOrder
  sorts
    Orderable
  others
    geq: Orderable Orderable;
  axioms
    E, F, G: Orderable;
    E = F if geq(E, F) and geq(F ,E);
    geq(E, F) if E = F;
    geq(E, F) if not geq(F, E);
    geq(E, G) if geq(E ,F) and geq(F, G);
end specification
```

Introduction
Automated Conformance Checking

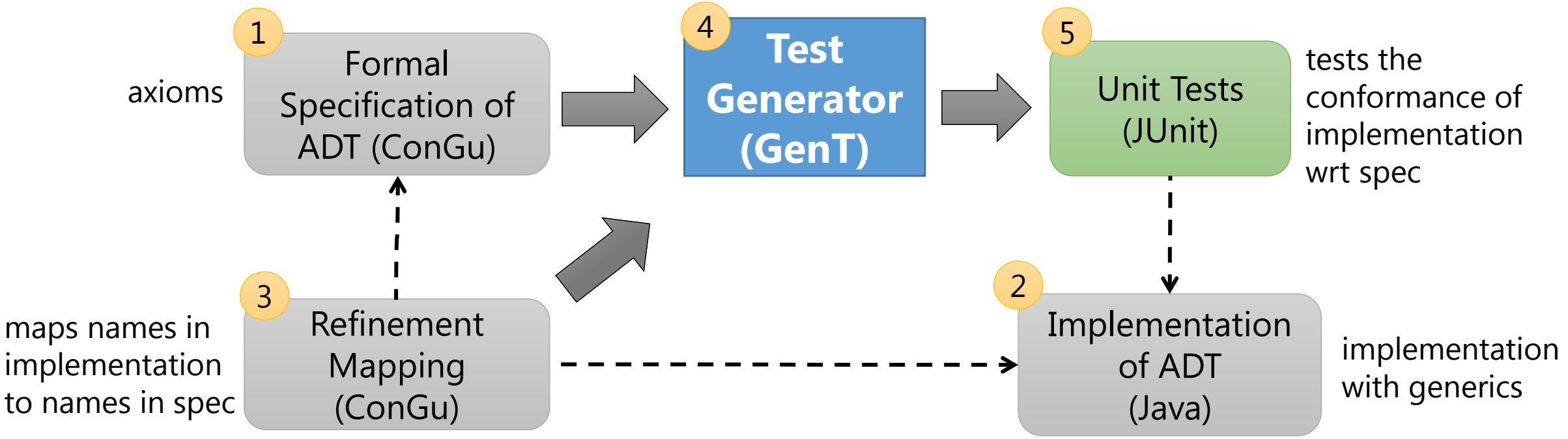
One can take advantage of formal specifications to **automatically check the conformance of implementations wrt the specification**, reducing the manual effort and increasing the confidence in the test process

Two complementary approaches were developed in the QUEST project:

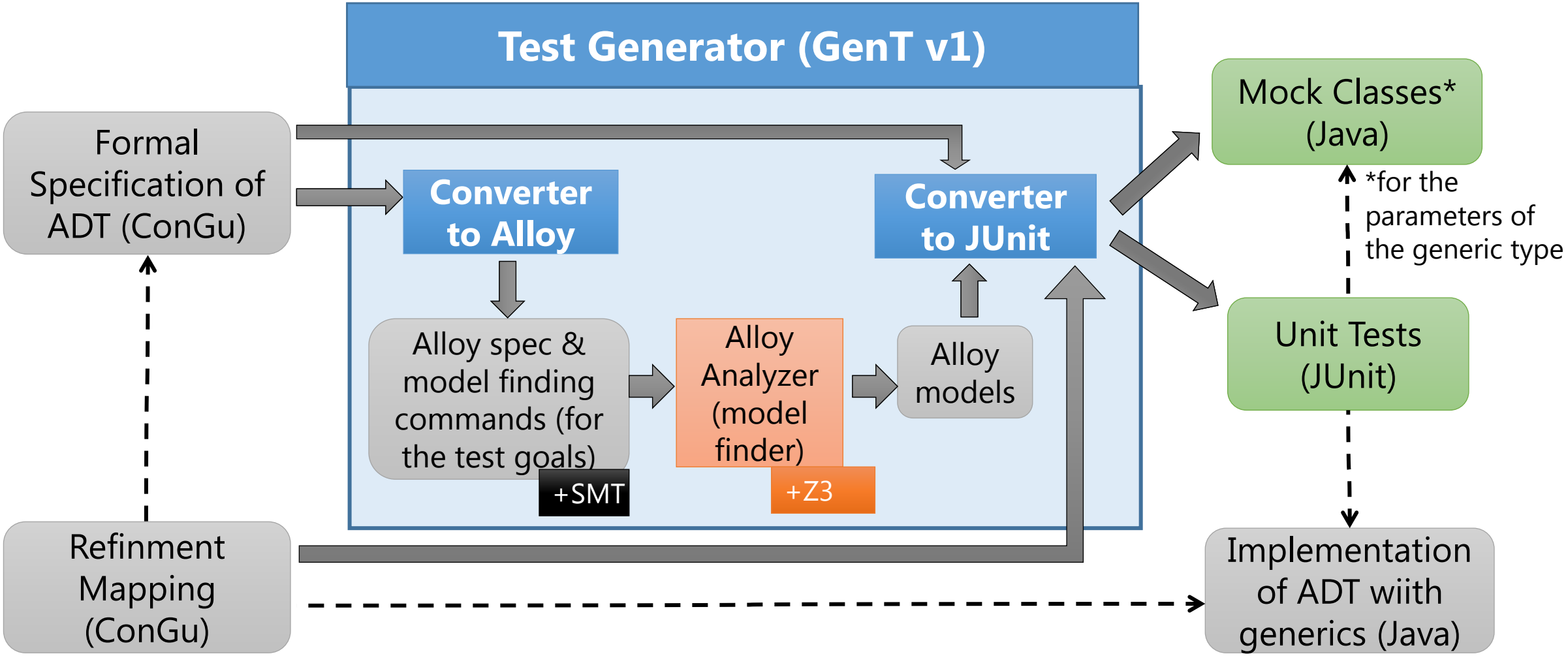
- generate **run-time assertions**, i.e., pre/post-conditions (FC/UL)
  - can be seen as built-in test oracles

- generate **test cases** (FE/UP) -> work presented here (GenT tool)!
  - test cases contain test inputs and expected results (oracles)

Introduction
Automated Conformance Testing



Base features (GenT v1)
Approach Overview



Base features (GenT v1)
Example (simplified & adapted)

Algebraic specification(ConGu)

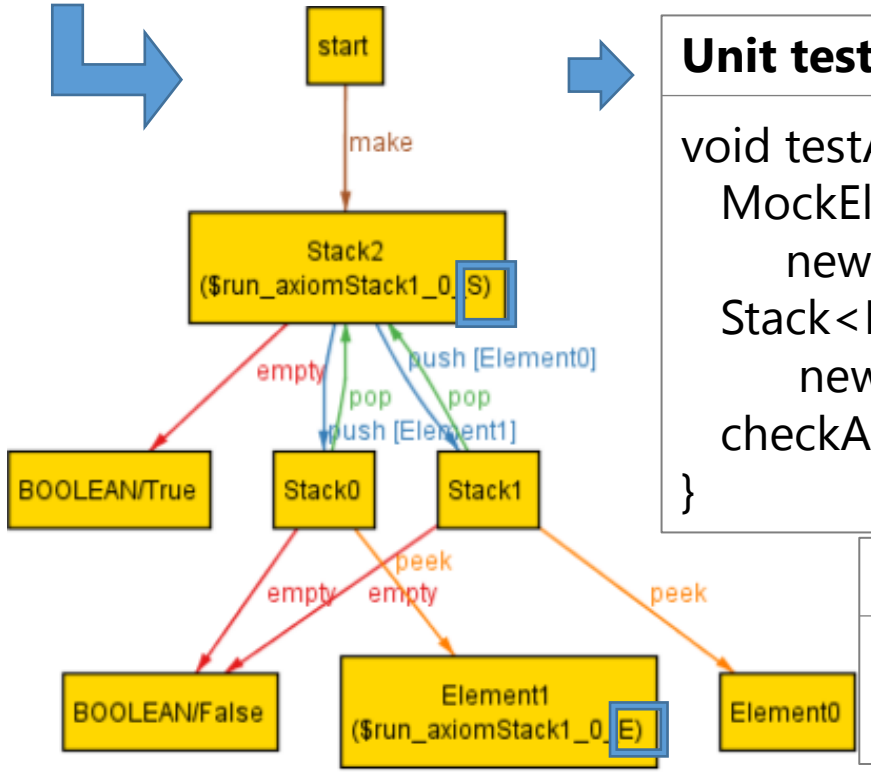
S:Stack; E:Element; **pop(push(S,E))=S**;

Alloy specification and model finding command

```
fact axiomStack1 {all S: Stack, E: Element |
        one S.push[E] implies S.push[E].pop = S }
run axiomStack1_0 {  some S:Stack, E:Element |
        one S.push[E]  and S.push[E].pop = S }
```

Parameterized test method (JUnit)

```
void checkAxiomStack1_0(
  Stack<Element> S, Element  E) {
      S2 = S.clone();
      S2.push(E);
      S2.pop();
      assertEquals(S, S2);
}
```



Unit test metod (JUnit)

```
void testAxiomStack1_0( ) {
   MockElement E =
        new  MockElement();
   Stack<MockElement> S =
        new Stack<MockElement>();
   checkAxiomStack1_0(S, E);
}
```

Mock class (Java)

```
public class MockElement
{ … }
```

Advanced features (GenT v2)
Support for unbounded data types

- Problem: A direct translation to Alloy of a data type such as an unbounded Stack is not satisfiable by any finite model

```
sig Stack  {
        push: Element -> one Stack,
        peek: lone Element,
        pop: lone Stack,
        empty: one BOOLEAN/Bool
}
```

There is no no finite model that satisfies this spec (except with zero instances of Element)

- Solution: Relax the definideness constraint for constructors such as *push*, using *lone* instead of *one*

```
sig Stack  {
        push: Element -> lone Stack,
        peek: lone Element,
        pop: lone Stack,
        empty: one BOOLEAN/Bool
}
```

- The approach is safe because the axiom verification code is the same as before

Advanced features (GenT v2)
Test generation for the *equals* method

- Problem: GenT v1 didn't generate tests specifically for the *equals* method, upon which the majority of test cases depend on.

```
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;

        Stack<E> other = (Stack<E>) obj;
        if(other.stack.size() != this.stack.size())
            return false;
        for(int i = 0; i!=this.stack.size(); i++){
            if (!this.stack.get(i).equals(other.stack.get(i)))
                return false;
        }
        return true;
    }
}
```

- Solution:
  - Automatically inject (in Alloy) the axioms of *equals*
    - reflexivity
    - symmetry
    - transitivity
  - Follow the normal test generation process from these axioms
- Results:
  - Improved test coverage

Advanced features (GenT v2)
Test generation for invalid inputs

Problem: GenT v1 didn't generate tests for invalid inputs

```
public static class PopEmptyStackException extends Exception
    public PopEmptyStackException(){
        super("Throws Exception Pop");
    }
}
public static class PeekEmptyStackException extends Exception
    public PeekEmptyStackException(){
        super("Throws Exception Peek");
    }
}

public void push(E elem){
    stack.addFirst(elem);
}

public void pop() throws PopEmptyStackException {
    if (stack.isEmpty())
        throw new PopEmptyStackException();
    stack.remove();
}

public boolean isEmpty(){
    return (stack.size() == 0);
}

public E peek() throws PeekEmptyStackException {
    if (stack.isEmpty())
        throw new PeekEmptyStackException();
    return stack.getFirst();
}
```

Solution: Generate run commands for values outside the domain (Alloy)

```
fact domainStack0 {
  all S1 : Stack | S1.empty != BOOLEAN/True implies one S1.peek else no S1.peek
}

run domainStack0_0 {
  some S1 : Stack | not ( S1.empty != BOOLEAN/True )
} for 6 but exactly 4 Stack, exactly 2 Elem
```

Generate a parameterized test method that checks if an exception is thrown for values outside the domain

```
/***** Axiom domainStack0 *****/
private void domainStack0Tester(CoreVarFactory<Stack<Object>> S1_Factory,
        String testId) throws Exception {
   Stack<Object> S10 = S1_Factory.create();

   if((S10.isEmpty() != true)) {
      Stack<Object> S11 = S1_Factory.create();
      assertTrue( defined(() -> {S11.peek();}));
   } else {
      Stack<Object> S12 = S1_Factory.create();
      assertTrue( ! defined(() -> {S12.peek();}));
   }
}
```

Advanced features (GenT v2)
Exclusion of unsatisfiable test goals

Problem: Some generated test goals may be unsatisfiable.

Axiom :
E: Orderable; S: SortedSet;
largest(insert(S,E)) = E **if not** isEmpty(S) **and** geq(E, largest(S));

Test goals (minterms of FDNF):

largest(insert(S,E))=E **and not** isEmpty(S) **and** geq(E, largest(S));

largest(insert(S,E))=E **and** isEmpty(S) **and** geq(E, largest(S));

…

Solution: If Alloy Analyzer does not find any model for a test goal (in bounded search), we use a SMT theorem prover (unbounded search)

```
assert axiomSortedSet5_1 {
   some E : Orderable, S : SortedSet | one  S.insert[E].largest and
( ( S.isEmpty != BOOLEAN/True and E.geq[S.largest] != BOOLEAN/True )
  and S.insert[E].largest = E )
}

check axiomSortedSet5_1
```

Alloy spec (modif.)

AlloyPE → *Translates Alloy to SMT*

```
(assert
 (exists ( (E Orderable) (S SortedSet) )
  (and
   (exists ( (o Orderable) (s SortedSet) )
    (and
     (insert S E s)
     (largest s o)
    )
```

SMT2 spec

Z3 → *Fully auto. theorem prover*

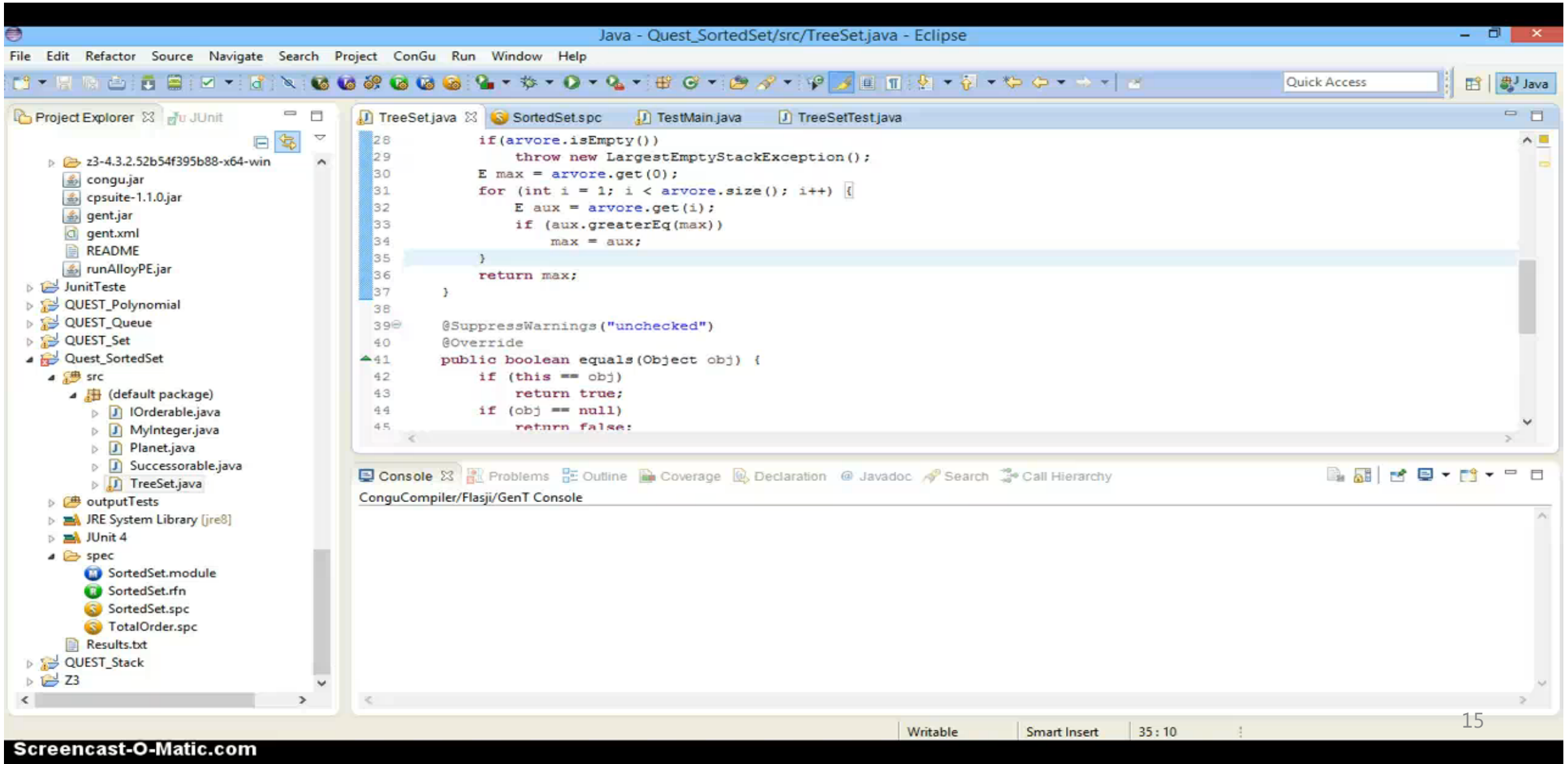
satisfiable / *not satisfiable*

Experimental Results

Item	Stack	SortedSet
Number of axioms	7	12
Number of axiom cases / run commands generated	13	36
With models found by Alloy Anayzer	13	29
Without models found by Alloy Analyzer	0	7 (1)
Time spent by Alloy Analyzer + Z3 in finding models	9 sec.	129 sec.
Number of JUnit test cases generated	13	29
Number of test cases failed	0	0
Coverage of the Java implementation	78%	75%

(1) All determined unsatisfiable with Z3 and consequently discarded.

Demo


Java - Quest_SortedSet/src/TreeSet.java - Eclipse
File Edit Refactor Source Navigate Search Project ConGu Run Window Help
Quick Access
Java
Project Explorer
JUnit
z3-4.3.2.52b54f395b88-x64-win
congu.jar
cpsuite-1.1.0.jar
gent.jar
gent.xml
README
runAlloyPE.jar
JunitTeste
QUEST_Polynomial
QUEST_Queue
QUEST_Set
Quest_SortedSet
src
(default package)
IOrderable.java
MyInteger.java
Planet.java
Successorable.java
TreeSet.java
outputTests
JRE System Library [jre8]
JUnit 4
spec
SortedSet.module
SortedSet.rfn
SortedSet.spc
TotalOrder.spc
Results.txt
QUEST_Stack
Z3
TreeSet.java
SortedSet.spc
TestMain.java
TreeSetTest.java
28 if(arvore.isEmpty())
29 throw new LargestEmptyStackException();
30 E max = arvore.get(0);
31 for (int i = 1; i < arvore.size(); i++) {
32 E aux = arvore.get(i);
33 if (aux.greaterEq(max))
34 max = aux;
35 }
36 return max;
37 }
38
39 @SuppressWarnings("unchecked")
40 @Override
41 public boolean equals(Object obj) {
42 if (this == obj)
43 return true;
44 if (obj == null)
45 return false;
Console
Problems
Outline
Coverage
Declaration
Javadoc
Search
Call Hierarchy
ConguCompiler/Flasji/GenT Console
Writable
Smart Insert
35 : 10
Screencast-O-Matic.com

Conclusions

The tool is able to automatically generate JUnit tests to check the conformance of implementations of abstract data types against their formal algebraic specification.

Future work

Update the ConGu Eclipse plug-in

Conduct further experiments

- with further Java libraries
- with students in classes

References and further reading

Specification-driven Unit Test Generation for Java Generic Classes, Francisco R. de Andrade, João P. Faria, Antónia Lopes, Ana C. R. Paiva, in Integrated Formal Methods, pp.296-311, 2012

Test Generation from Bounded Algebraic Specifications using Alloy, Francisco Rebello de Andrade, João Pascoal Faria, Ana C. R. Paiva, in ICSOFT 2011, pp.-, 2011

GenT: Ferramenta para Geração Automática de Testes Unitários a partir de Especificações Algébricas usando Alloy e SMT, Tiago Campos, Francisco Silva, INForum 2014, 2014

Ferramenta de Geração Automática de Testes Unitários a partir de Especificações Algébricas usando Alloy e SMT, Tiago Faria Campos, Master in Informatics and Computing Engineering, FEUP, 2014

Geração de Testes a partir de Especificações Algébricas de Tipos Genéricos usando Alloy, Francisco R. de Andrade, João P. Faria, Ana C. R. Paiva, Antónia Lopes, in INForum 2011, pp.-, 2011

Questions?

Thank you!