Logic–based Software Testing and Verification

Emanuele De Angelis
Istituto di Analisi dei Sistemi ed Informatica
“Antonio Ruberti”
Consiglio Nazionale delle Ricerche

Software And Knowledge–based Systems

joint work with: Fabio Fioravanti\textsuperscript{1}, Adrián Palacios\textsuperscript{2}, Alberto Pettorossi\textsuperscript{3}, Maurizio Proietti\textsuperscript{4}

\textsuperscript{1}University of Chieti–Pescara “G. d’Annunzio”, Italy
\textsuperscript{2}AWS, USA
\textsuperscript{3}University of Rome “Tor Vergata”, Italy
\textsuperscript{4}IASI–CNR, Italy
What’s about this (short) talk?

Logic-based Software Testing and Verification

- Constrained Horn Clauses
- Property-based Testing
- Contract-based Verification
Example-based Testing

automation level 1 – test cases execution

A developer/test engineer designs a collection of examples of the program behaviour in the form of pairs < input, expected output > and compares the expected output against the actual output.

```c
int max(int a[], int n) {
    int i = 1;
    int m = a[0];

    for (i = 1; i < n-1; i++)
        if (a[i] > m)
            m = a[i];

    return m;
}
```

<table>
<thead>
<tr>
<th>Input</th>
<th>expected output</th>
<th>actual output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[60,1], 2</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>[-3,110,1], 3</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>[9], 1</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>[7,3,23,42], 4</td>
<td>42</td>
<td>23</td>
</tr>
</tbody>
</table>
```
Example-based Testing

automation level 1 – test cases execution

A developer/test engineer designs a collection of examples of the program behaviour in the form of pairs < input, expected output > and compares the expected output against the actual output.

<table>
<thead>
<tr>
<th>Input (a, n)</th>
<th>expected output</th>
<th>actual output</th>
</tr>
</thead>
<tbody>
<tr>
<td>([60,1], 2)</td>
<td>60</td>
<td>60</td>
</tr>
<tr>
<td>([-3,110,1], 3)</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>([9], 1)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>([7,3,23,42], 4)</td>
<td>42</td>
<td>23</td>
</tr>
</tbody>
</table>

Straightforward process, but does not give very high guarantees of program correctness.
Property-based Testing

automation level 2 – test cases generation & execution

instead of designing specific examples, a developer/test engineer provides a specification for program inputs and outputs

Given any array of integers \( a \) of size \( n \), \( \max(a, n) \) is the largest element of \( a \)

Automated generate & test approach

1. randomly generate an input that meets the specification
2. run the program with that input
3. test whether or not the output meets the specification
Property-based Testing

automation level 2 – test cases generation & execution

So far, so good – to generate simple test inputs (e.g., lists of integers)

Given any array of integers $a$ of size $n$, $\max(a, n)$ is the largest element of $a$

but what happens if the inputs have constraints on

- the content of data structure (e.g., sorted lists)
- the shape of the data structure (e.g., balanced trees)
- both (e.g., AVL trees)
Generating (complex) inputs from (complex) specifications

ordered(L) -> case L of
[A,B|T] -> A <= B andalso ordered([B|T]);
_ -> true
end.

avl(T) -> case T of
leaf -> true;
{node,L,V,R} -> B = height(L) - height(R) andalso
B >= -1 andalso B <= 1 andalso
ltt(L,V) andalso gtt(R,V) andalso
avl(L) andalso avl(R); 
_ -> false
end.

Random generation of inputs from the specification can be quite inefficient. State-of-the-art approaches use ad-hoc, hand-written, generators.

<<== ordered list
- constraints on the elements

<<== AVL tree
- binary search tree
- constraints on the elements
- height-balanced
- constraints on the shape
Property-based Testing
automation level 3 – generator of (test cases) generators

Constrained Horn Clauses (CHCs)
(a fragment of First Order Predicate Calculus)
tо formalize specifications

Operational Semantics of CHCs: Constraint Logic Programming (CLP)
CHC specifications >> runnable specifications
>> test cases generators

Smart Interpreter for CLP specifications: Constrain & Generate
(allows for a finer control of the test cases generation process)
we get an efficient generator of (test cases) generators
A glimpse of the generation process ...

```
ordered_list(L) :-

>typeof(L,list(integer)) ,

eval(apply('ordered',[var('L')]),[(L,L)],lit(atom,true)) .

? - ordered_list(L).
L = nil ;
L = cons(lit(int,X),nil), X in inf..sup ;
L = cons(lit(int,X),cons(lit(int,Y),nil)), Y #>= X ;
L = cons(lit(int,X),cons(lit(int,Y),cons(lit(int,Z),nil))), Y #>= X, Z #>= Y
```
Verification
Certifying Correctness

- **testing**, by exercising the program on specific inputs, can only **find bugs**

- **verification** aims at **proving** that there are no bugs for any input (even in infinite domains)
**Contract-based Verification**

Certifying Correctness

```
def rev(l: List[BigInt]): List[BigInt] = {
    require(is_asorted(l))
    l match {
        case Nil() => Nil[BigInt]()
        case Cons(x, xs) => snoc(rev(xs),x)
    } } ensuring { res => is_dsorted(res) }
```

- **precondition of rev**: is_asorted(l)
- **postcondition of rev**: is_dsorted(res)

---

**Specification S** + **Program P**

**SW WORLD**

**LOGIC WORLD**

- **CHCs Generator**
- **CHCs Transformer**
- **CHCs Solver**

- **SAT** $\Rightarrow$ P satisfies S
- **UNSAT** $\Rightarrow$ P violates S
- **UNK** (undecidable prob.)

**find a counterexample (i.e, a bug)**
to conclude ...

A few pointers

• Verifying Catamorphism-Based Contracts using Constrained Horn Clauses
  E. De Angelis, F. Fioravanti, A. Pettorossi, M. Proietti
  Theory and Practice of Logic Programming, 2022

• Analysis and Transformation of Constrained Horn Clauses for Program Verification
  E. De Angelis, F. Fioravanti, A. Pettorossi, J.P. Gallagher, M.V. Hermenegildo, M. Proietti
  Theory and Practice of Logic Programming, 2021

• Property-Based Test Case Generators for Free
  E. De Angelis, A. Palacios, F. Fioravanti, A. Pettorossi, M. Proietti
  LNCS 11823, Springer, 2019

Ongoing work

CHC testing & verification of rule-based XAI models: do not test the AI model (e.g., the NN), but the explainable model of the AI model (rule-based XAI model)