



Random Testing

Wieger Wesselink HG 6.77
j.w.wesselink@tue.nl




Random Testing

Improve trust in the correctness of the
mCRL2 toolset implementation

Why is testing important?

Model

```
act a,b;  
proc S = sum n: > a.X(n);  
proc X(n:Nat) = (a+b).X(n)  
                .a.X(Int2Nat(n-1));  
init S;
```



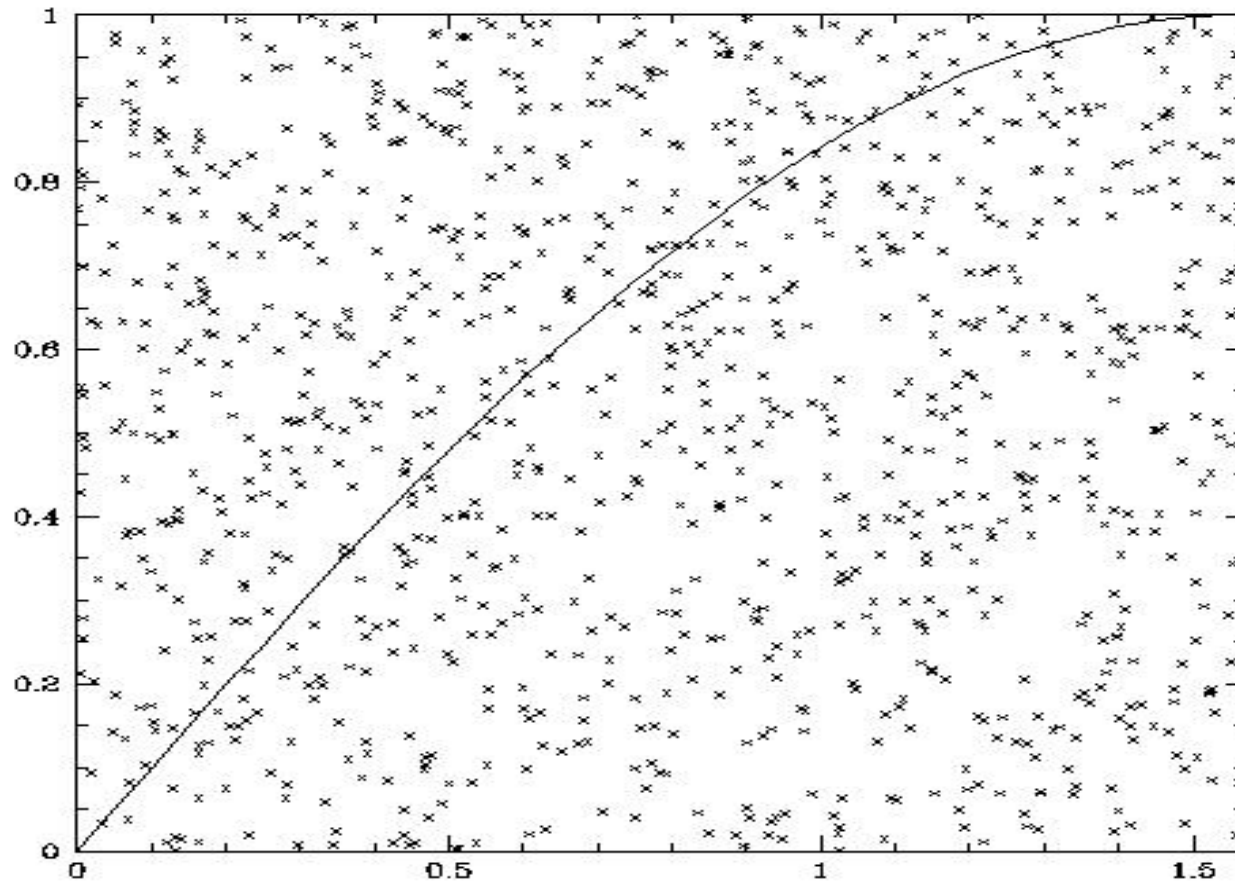
Property

```
<a>([a] false)
```

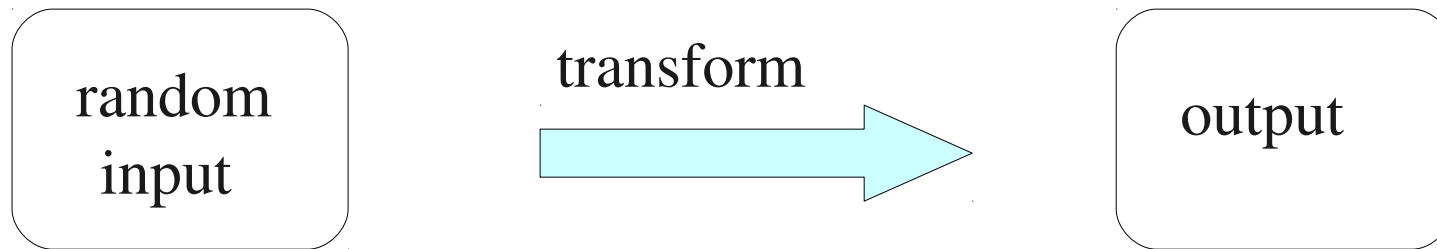
How do we currently test?

- Unit tests
- Regression tests
- System tests
- Performance tests
- **Random tests**

Random testing



Typical mCRL2 use case



Check a property that should be maintained

Aspects of Random Testing

- Grammar based testing
- Coverage
- Depth control (combinatorial explosion)
- Black-box testing
- Syntax-based or Semantics-based?

Practical Experiences

- Applied to Boolean Equation Systems
- About 5-10 bugs discovered
- Automatic counter example minimization
- Also useful during development

Example

```
if (pbes_equation(*i).symbol() == sigma)
{
    m_priorities[pbes_equation(*i).variable().name()] = priority;
}
else
{
    sigma = pbes_equation(*i).symbol();
    m_priorities[pbes_equation(*i).variable().name()] = ++priority;
}
```

Example

```
if (pbes_equation(*i).symbol() == sigma)
{
    m_priorities[pbes_equation(*i).variable().name()] = priority;
}
else
{
    sigma = pbes_equation(*i).symbol();
    m_priorities[pbes_equation(*i).variable().name()] = priority++;
}
```

Random Test

```
from path import *
from random_pbes_generator import *
from mcrl2_tools import *

def test_pbes_solvers(p, filename):
    txtfile = filename + '.txt'
    path(txtfile).write_text('%s' % p)
    pbesfile = filename + '.pbes'
    run_txt2pbes(txtfile, pbesfile)
    answer1 = run_pbes2bool(pbesfile)
    answer2 = run_pbespgsolve(pbesfile)
    print filename, answer1, answer2
    if answer1 == None or answer2 == None:
        return True
    return answer1 == answer2

def main():
    options = parse_command_line()
    try:
        equation_count = 5
        atom_count = 4
        propvar_count = 3
        use_quantifiers = True

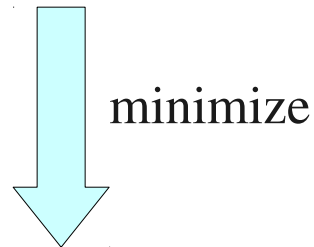
        for i in range(10000):
            filename = 'pbес_solvers'
            p = make_pbес(equation_count, atom_count, propvar_count, use_quantifiers)
            if not test_pbес_solvers(p, filename):
                m = CounterExampleMinimizer(p, lambda x: test_pbес_solvers(x, filename + '_minimize'), 'pbес_solvers')
                m.minimize()
                raise Exception('Test %s.txt failed' % filename)
    finally:
        if not options.keep_files:
            remove_temporary_files()

if __name__ == '__main__':
    main()
```

Counter example

```
pbes mu X0(c: Bool, n: Nat) = X0(true, 1) || (forall v: Nat. val(v < 3) && (exists w: Nat. val(w < 3) ||
((val(!w > 0)) && X1(1)) || val(n > 1)) || val(!v < 3) || val(v == n) || X3(true));
  mu X1(n: Nat) = (forall w: Nat. val(!w < 3) && (X4(w < 2) || (exists t: Nat. val(!t < 3) || X2))
|| (val(w > 0) && val(w < 2)) || X1(n + 1)) || (exists w: Nat. val(w < 3) || val(false) || val(w > 0));
  nu X2 = val(!false) || ((forall t: Nat. val(t < 3) && X4(true)) && val(true)) && (X3(true) ||
X2) || val(true) || val(true);
  mu X3(b: Bool) = ((X0(b, 1) || val(b)) || val(false)) && val(true) && val(b) && X1(1) || X4(false);
  mu X4(c: Bool) = ((val(!c) || X1(1)) && (forall v: Nat. val(!v < 3) && (forall u: Nat. val(u < 3) &&
X4(u > 1)) && val(!v > 1))) && X0(false, 0) && val(!c) && (forall v: Nat. val(v < 3) && val(!v > 1));

init X0(true, 0);
```



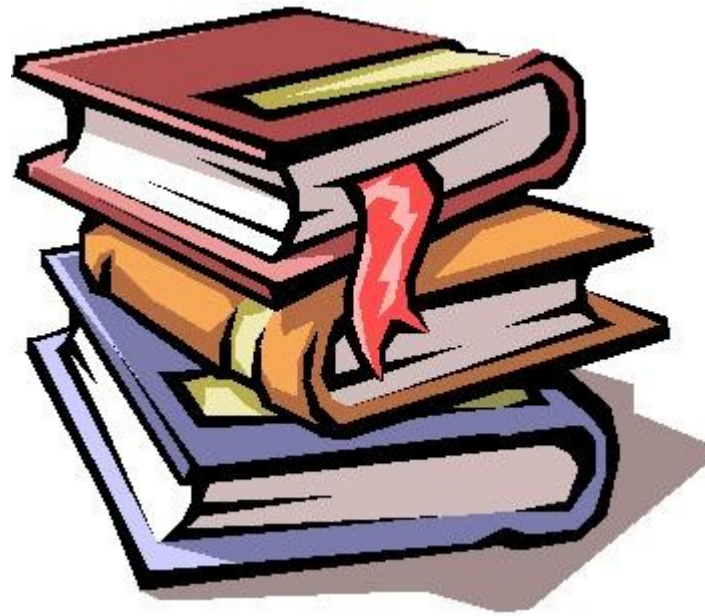
```
pbes mu X0(c: Bool, n: Nat) = X0(true, 1) || false;
  mu X1(n: Nat) = true;
  nu X2 = true;
  mu X3(b: Bool) = true;
  mu X4(c: Bool) = true;

init X0(true, 0);
```

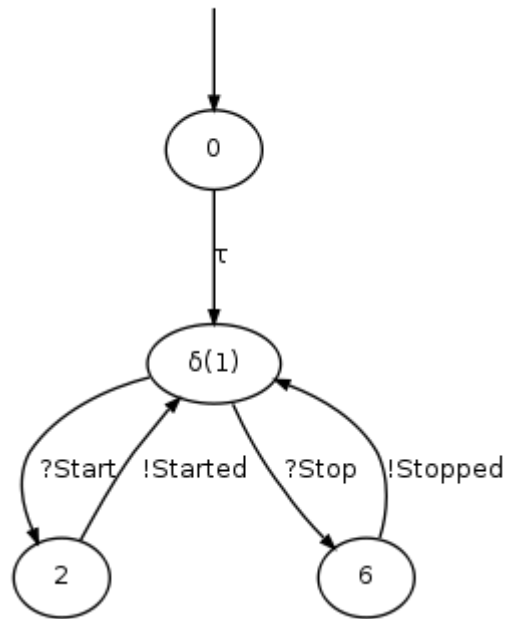
Assignment

Experiment with random testing
in the mCRL2 toolset

1) Literature study



2) Labeled Transition Systems



Apply the theory of random testing to LTSs

3) Practical experiments

