Testing DSLs

How to test DSLs, their IDEs and Programs

Markus Voelter
independent/itemis
voelter@acm.org

www.voelter.de
voelterblog.blogspot.de
@markusvoelter
+Markus Voelter

A DSL is a **focussed**, **processable language** for describing a specific **concern** when building a system in a specific **domain**. The **abstractions** and **notations** used are natural/suitable for the **stakeholders** who specify that particular concern.

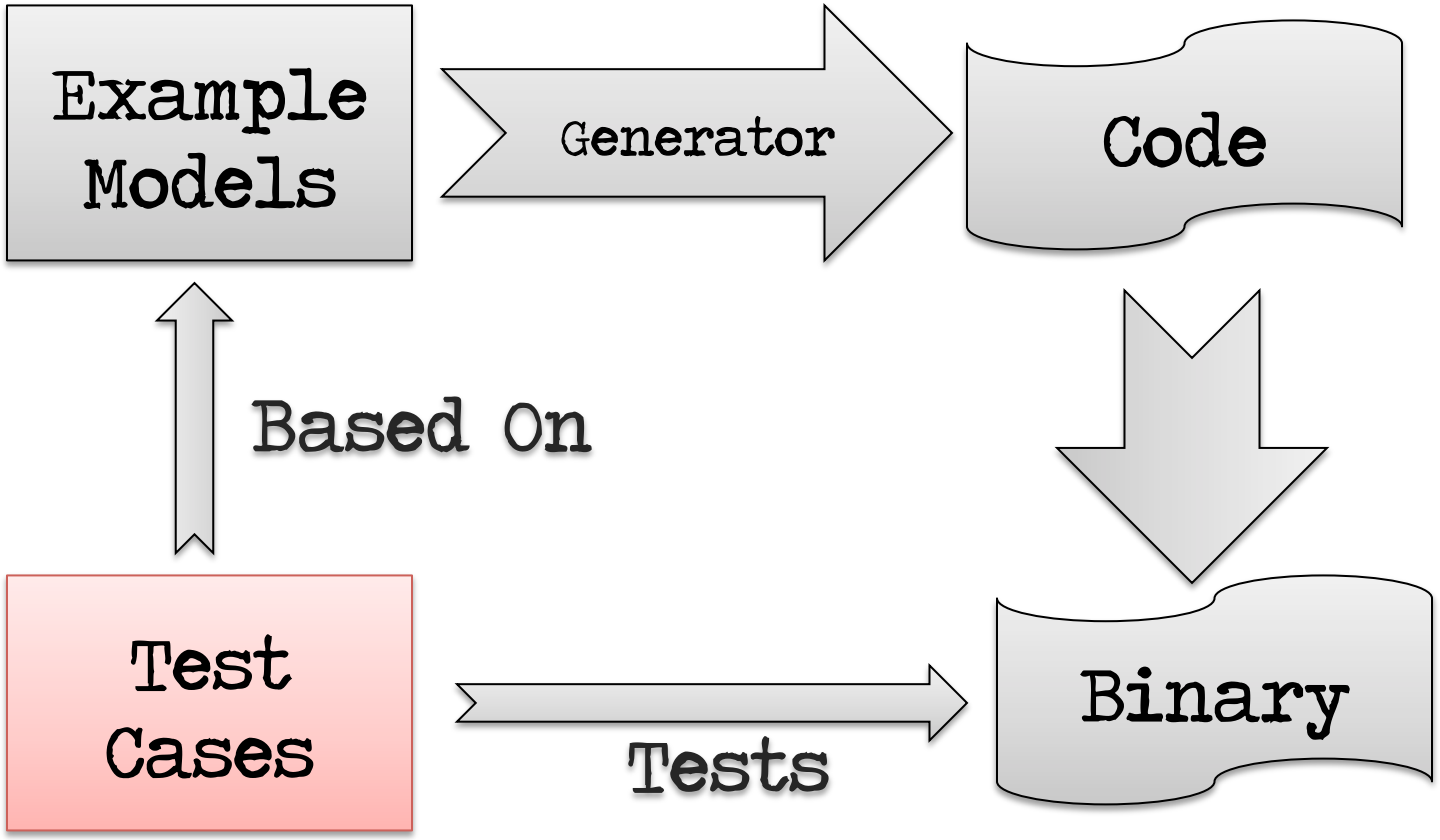
Limited Expressiveness.

Reduced Need For Tests.

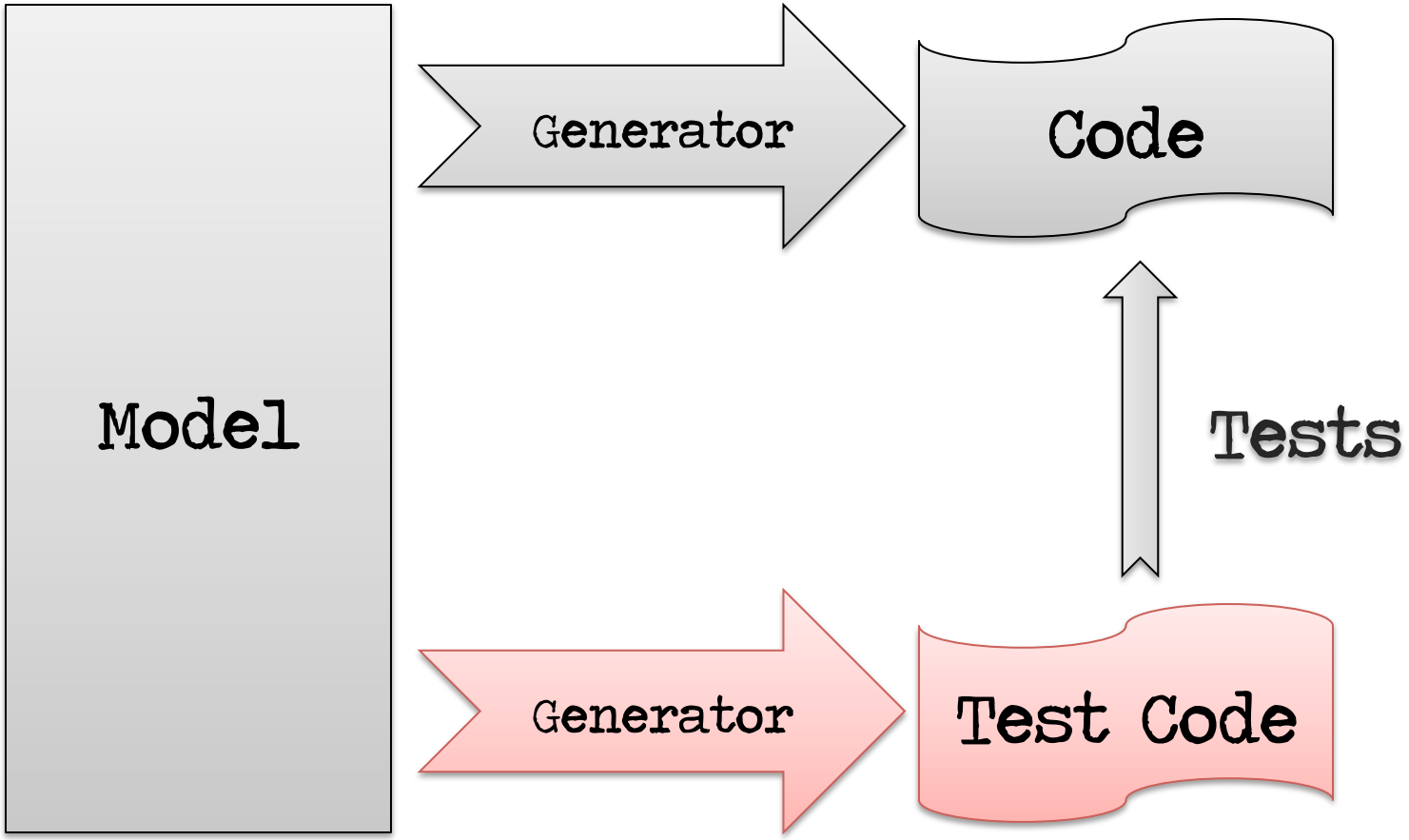
Constraint Checks.

A Form of Test.

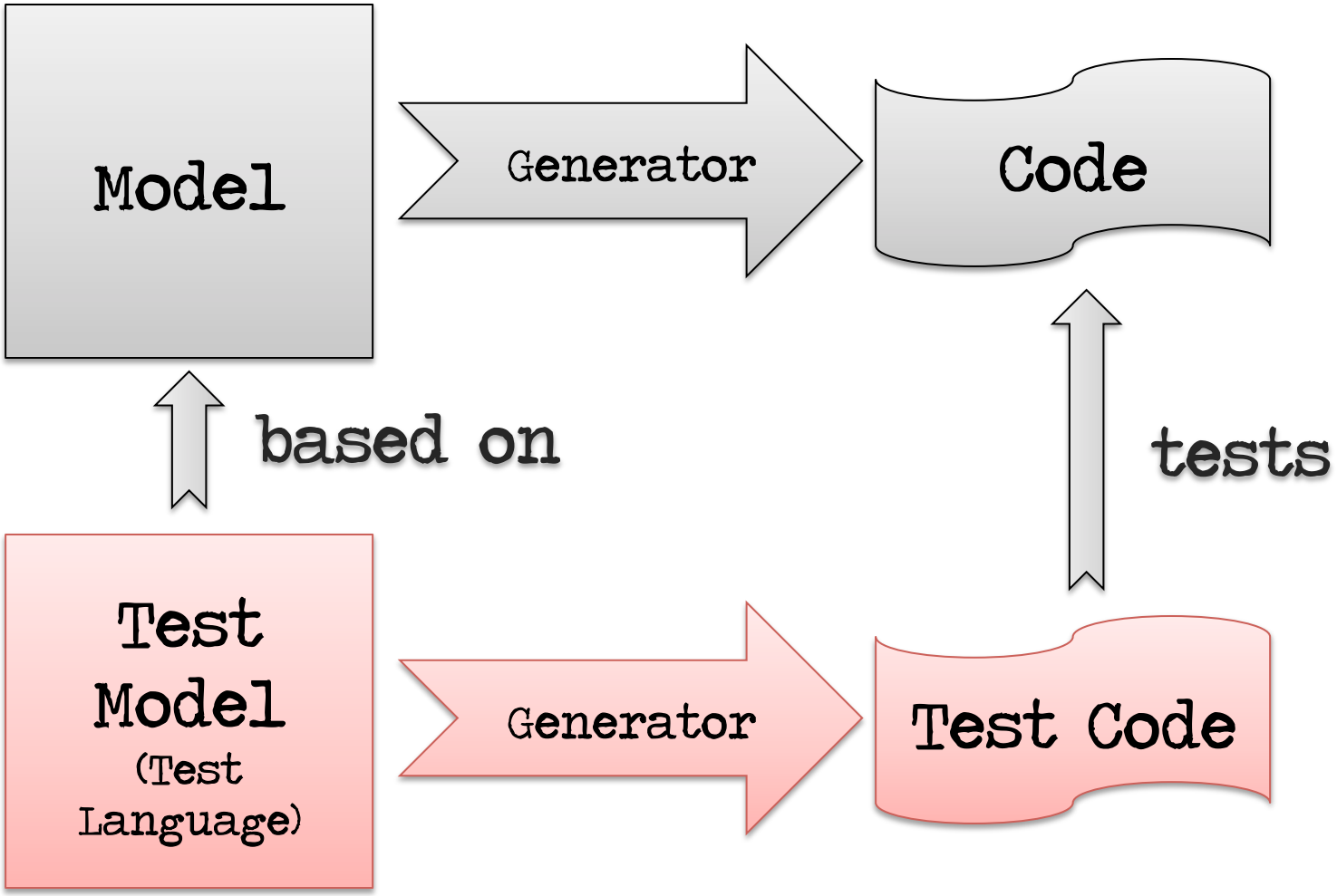
Test Semantics, not Structure!



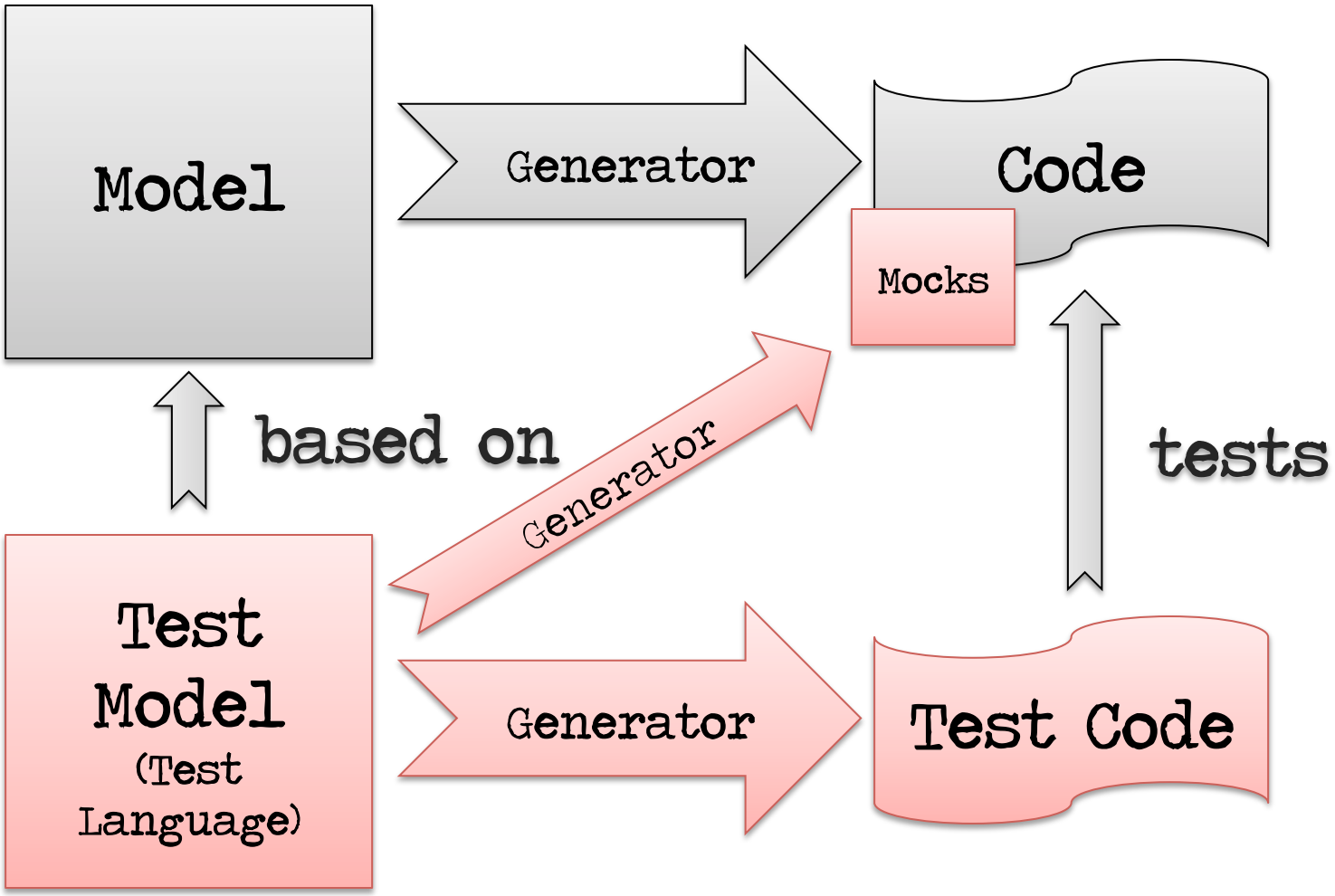
This tests only the generators!



Separate test models and generated test code



Separate test models and generated test code

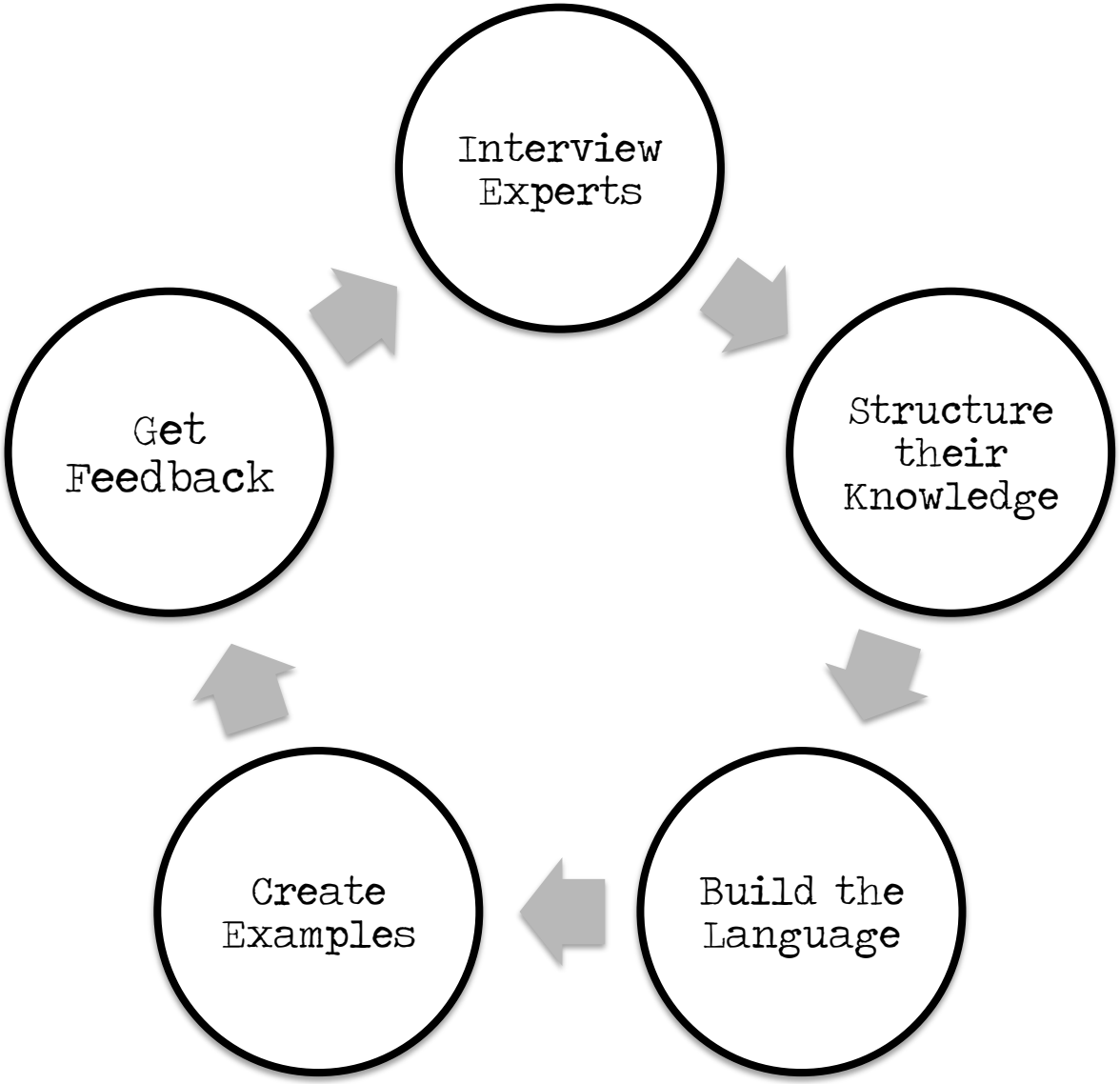


Testing the Language

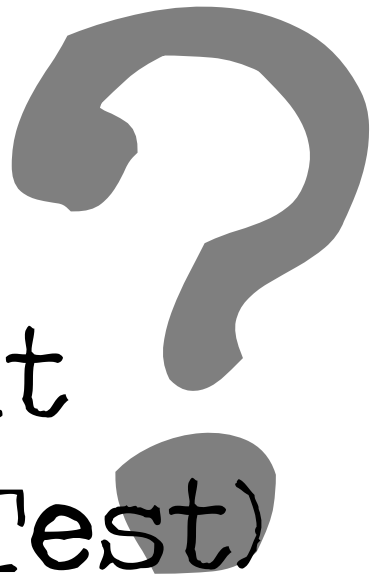
Testing Programs

Does the DSL cover the domain it is intended to cover?

Build Examples and discuss with Stakeholders.


Interview Experts
Structure their Knowledge
Build the Language
Create Examples
Get Feedback

Can the language definition process all of the relevant notation? (Parser Test)



Create and maintain example programs and keep trying to parse them

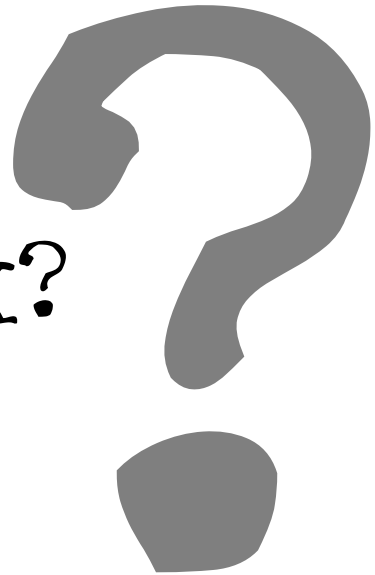
```
@RunWith(XtextRunner.class)
@InjectWith(Expr2DslInjectorProvider.class)
public class TestModels extends XtextTest {

    @Test
    public void testBasics() {
        testFileNoSerializer("1-basic.expr2");
    }

}
```

http://code.google.com/a/eclipselabs.org/p/xtext-utils/wiki/Unit_Testing

```java
@RunWith(XtextRunner2.class)
@InjectWith(DomainmodelInjectorProvider.class)
public class ParserAndLexerTest extends XtextTest {
	@Test
	public void id(){
		testTerminal("bar", /* token stream: */ "ID");
		testTerminal("bar3", /* token stream: */ "ID");
		testTerminal("_bar_", /* token stream: */ "ID");
		testTerminal("$bar$", /* token stream: */ "ID");

		testNotTerminal("3bar", /* unexpected */ "ID");
		testNotTerminal("#bar", /* unexpected */ "ID");

		// token streams with multiple token
		testTerminal("foo.bar", "ID", "'.'", "ID");
		testTerminal("foo.*", "ID", "'.'", "'*'");
	}

	@Test
	public void qualifiedName(){
		testParserRule("foo.bar", "QualifiedName");
		testParserRuleErrors("3foo.bar", "QualifiedName", "extraneous input '3'");
	}

	@Test
	public void qualifiedNameWithWildcard(){
		testParserRule("foo.*", "QualifiedNameWithWildCard");
	}
}
```

http://code.google.com/a/eclipselabs.org/p/xtext-utils/wiki/Unit_Testing

Do the scopes work?

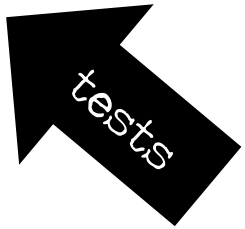


Do the constraints and typesystem work?

Are there any error messages associated with certain program elements?

Create positive and negative examples and check for correct error annotations.

```
tested model basic

var int a = 1;
var int b = 2;
var int c = 3;

calc int x = 1;
calc int y = a + c;
calc int z = a * a + plus(1,2);

assert a is 1
assert b is 2
assert x is 1
assert y is 4
assert z is 4

function int plus(int a, int b ) {
    return a+b;
}
```




```
@Test
public void testBasics() {
    testFileNoSerializer("1-basic.expr2");
    assertConstraints( issues.errorsOnly().sizeIs(0) );
}
```

```
model subtyping

calc int t1 = 2;        // works
calc int t2 = 2.2;      // error: float cannot be assigned to int
calc float t3 = 2.2;    // works
calc float t4 = 2;      // works; int can be assigned to float
calc int t5 = 2 + 2;    // works; type of plus is int
calc int t6 = 2 + 2.3; // error: common type of 2 and 2.3 is float
                                                    signed to int
```

incompatible type Int and Float (on a Formula)

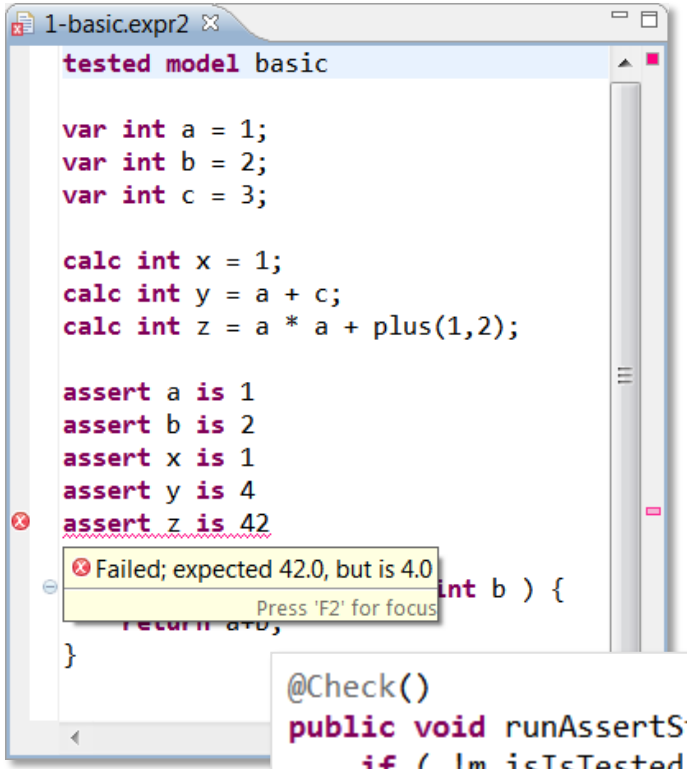
Press 'F2' for focus

tests

```
@Test
public void testTArray() throws Exception {
    testFileNoSerializer("4-array.expr2");
    assertConstraints( issues.errorsOnly().sizeIs(1) );
    assertConstraints( issues.forType(Formula.class).named("anotherOne2").
            theOneAndOnlyContains("incompatible") );
}
```

The method…	… returns a new IssueCollection that
forType(t)	contains only those issues that are attached to an instance of t
get(index)	are at position index in the IssueCollection
inLine(line)	are in line line in the model file
withStringFeatureValue (n, v)	whose feature named n has the value (tostring()) v
errorsOnly()	contains no warnings
named(n)	contains only those issues that are attached to an element with name property value n
forElement(t, n)	contains only those issues that are attached to elements have the na
under(t)	contains issues whos an ancestor
under(t, n)	contains issues whos an ancest named n

The method…	… asserts that
sizeIs(*s*)	the size of the current collection is *s*
oneOfThemContains(*t*)	the collection has any size, and one of the error messages contains the substring *t*
allOfThemContain(*t*)	the collection has any size, and all the error messages contains the substring *t*
theOneAndOnlyContains(*t*)	the collection is of size 1 and the message of the singe error contains the substring *t*



```
@Check()
public void runAssertStatements( Model m ) {
    if ( !m.isIsTested() ) return;
    try {
        MessageList errors = new ExprModelInterpreter().runModel(m, ts);
        for (MessageList.MessageItem o: errors.getMessages()) {
            error( o.message, o.element, null, -1, INTERPRETERFAILED );
        }
    } catch (InterpreterException e) {
        if ( e.getFailedObject() != null ) {
            error( e.getMessage(), e.getFailedObject(), null, -1, INTERPRETERFAILED );
        } else {
            error( e.getMessage(), m, null, -1, INTERPRETERFAILED );
            e.printStackTrace();
        }
    }
}
```

```
module Dummy from tests.ts.core.tests1 imports nothing {

  void dummy() {
    int8_t[2] andererArray = {10, 20};
    int8_t[3] intarr = {10, 20, 30};
    int8_t[2] two = {1, 2, 3};
    float[2] fa = {1, 2};
    int8_t[2] ia = {1.1, 2.1};
    int8_t* intp = andererArray;
    int8_t* intp2 = intarr;
    int8_t[ ] xx = {1, 2.1, "Hello"};
    int8_t x = xx[2];
    float ff = xx[2];
    int8_t fi = fa[2];

  } dummy (function)
}
```

Test case Arrays

nodes

```
( <check types module Dummy from tests.ts.core.tests1 imports nothing { > )

                void dummy() {
                  int8_t[2] andererArray = {10, 20};
                  int8_t[3] intarr = {10, 20, 30};
                  int8_t[2] two = <node {1, 2, 3} has error>;
                  float[2] fa = {1, 2};
                  int8_t[2] ia = <node {1.1, 2.1} has error>;
                  int8_t* intp = andererArray;
                  intp = <node fa has error>;
                  int8_t* intp2 = intarr;
                  int8_t[ ] xx = <node {1, 2.1, "Hello"} has error>;
                  int8_t x = xx[2];
                  float ff = xx[2];
                  int8_t fi = <node fa[2] has error>;

                } dummy (function)
              }
```

```
Test case Arrays
nodes
  ( [ <check types module Dummy from tests.ts.core.tests1 imports nothing { > ] )

                    exported test case arrayAccessPrioTest {
                      int8_t[2] array = {10, 20};
                      <arrayAccessPrioTest1  array[1]>;
                      <arrayAccessPrioTest2  array[1] * 2>;
                      int8_t i = 2;
                      <arrayAccessPrioTest3  array[1] * i++>;

                      <arrayAccessPrioTest4  -array[1]>;
                    } arrayAccessPrioTest(test case)
                  }

test methods
  test prioTestArrayAccess {
    assert true arrayAccessPrioTest1.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest2.isInstanceOf(MultiExpression);
    assert true arrayAccessPrioTest2.left.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest2.right.isInstanceOf(NumberLiteral);
    assert true arrayAccessPrioTest3.isInstanceOf(MultiExpression);
    assert true arrayAccessPrioTest3.left.isInstanceOf(ArrayAccessExpr);
    assert true arrayAccessPrioTest3.right.isInstanceOf(PostIncrementExpression);
    assert true arrayAccessPrioTest4.isInstanceOf(UnaryMinusExpression);
    assert true arrayAccessPrioTest4.expression.isInstanceOf(ArrayAccessExpr);
  }
```

Do the generators, transformations or interpreters work?

Run them; execute test cases against the running code.

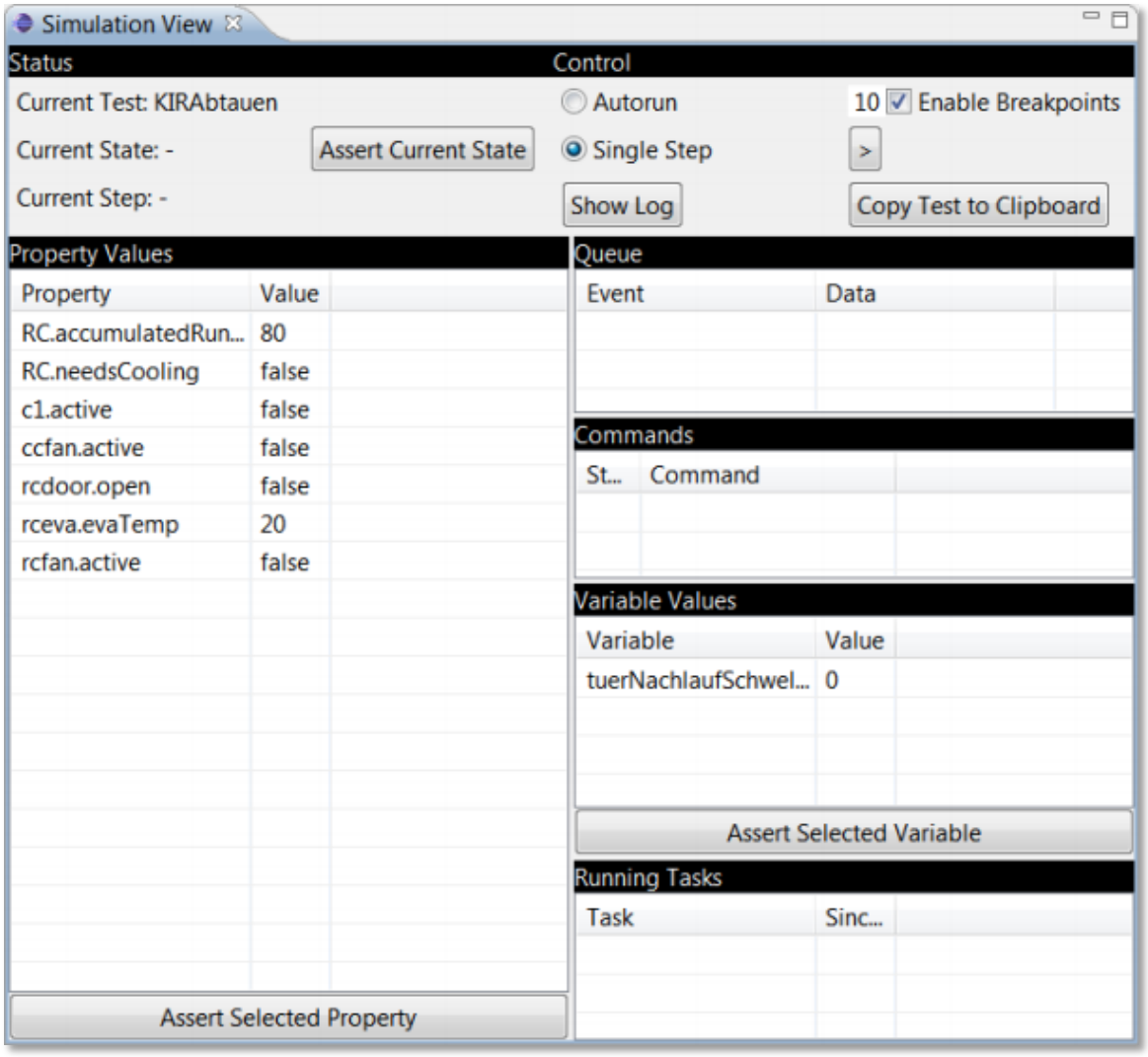
You can write test cases in the target language manually.

Or you can express test cases on the model level

```
parameter t_abtaustart: int
parameter t_abtaudauer: int
parameter T_abtauEnde: int

var tuerNachlaufSchwelle: int = 0

start:
    entry { state noCooling }

state noCooling:
    check ( (RC->needsCooling) && (cc.c1->stehz
        state rccooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        perform rcfanabschalttask after 10 {
            set RC.rcfan->active = false
        }
    }


state rccooling:
    entry { set RC.rcfan->active = true }
    check ( !(RC->needsCooling)  ) {
        state noCooling
    }
    on isDown ( RC.rcdoor->open ) {
        set RC.rcfan->active = true
        set RC.rclight->active = false
        set tuerNachlaufSchwelle = currStep + 30
    }
    exit {
        perform rcfanabschalttask after max( 5, tuerNachlaufSchwelle-currStep ) {
            set RC.rcfan->active = false
        }
    }
```

```
prolog {
    set RC->accumulatedRuntime = 80
}

step 10
assert-currentstate-is noCooling

mock: set RC->accumulatedRuntime = 110
step


mock: set RC.rceva->evaTemp = 10
assert-currentstate-is abtauen
assert-value cc.c1->active is false
mock: set RC->accumulatedRuntime = 0
step 5
assert-currentstate-is abtauen
assert-value cc.c1->active is false
step 15
assert-currentstate-is noCooling
```

Interpreting the tests directly in the IDE reduced turn around time.


Simulation View
Status
Control
Current Test: KIRAbtauen
Autorun
10
Enable Breakpoints
Current State: -
Assert Current State
Single Step
>
Current Step: -
Show Log
Copy Test to Clipboard
Property Values
Property
Value
RC.accumulatedRun...
80
RC.needsCooling
false
c1.active
false
ccfan.active
false
rcdoor.open
false
rceva.evaTemp
20
rcfan.active
false
Assert Selected Property
Queue
Event
Data
Commands
St...
Command
Variable Values
Variable
Value
tuerNachlaufSchwel...
0
Assert Selected Variable
Running Tasks
Task
Sinc...

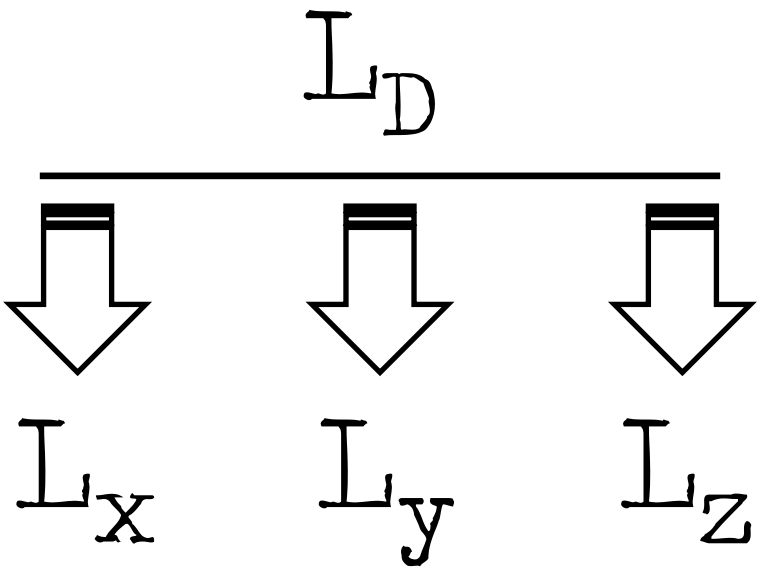
Name	Documentation	Tags	Valid time	Transaction time	Fixture	Product	Element	Expected value	Actual value
Accrued right at retireme			2006-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	761.0402	761.0402
Accrued Right last final pay			2004-1-1	2007-9-24	Jan De Jong	Old Age Pension	Accrued right	705.0589	705.0589
premiun last year			2006-1-1	2007-9-24	Jan De Jong	Old Age Pension	Premium old age pension	329.0625	329.0625
Accrued right at retireme 2)			2006-12-31	2007-9-24	Piet Van Dijk	Old Age Pension	Accrued right	740.94	724.7658
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	73.661	73.661
			1985-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	3.7534	3.7534
			1987-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	7750	7750
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Accrued Right in service period	387.7449	387.7449
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Years of service in service period	10.8082	10.8082
			1998-12-31	2007-9-24	Jan De Jong	Old Age Pension	Pension base average FP	8250	8250

```
exported test case logicalOperators {
  assert(0) (true || true) == true;
  assert(1) (true || false) == true;
  assert(2) (false || false) == false;
  assert(3) (true && true) == true;
  assert(4) (true && false) == false;
  assert(5) (false && false) == false;
  assert(6) (true && false || true) == true;
  assert(7) (true && false || false) == false;
} logicalOperators(test case)
```

```
module functionPointerStuff from test.ex.core.pointers imports nothing {

  var (int8_t, int8_t)=>(int8_t) globalRef;

  typedef (int8_t, int8_t)=>(int8_t) as twoIntArgsReturningInt;

  int8_t add(int8_t x, int8_t y) {
    return x + y;
  } add (function)

  exported test case testFC {
    globalRef = :add;
    int8_t res = globalRef(1, 2);
    assert(0) res == 3;

    (int8_t, int8_t)=>(int8_t) localRef;
    localRef = :add;
    int8_t localres = localRef(10, 22);
    assert(1) localres == 32;
    assert(2) -anotherHOF(:add) == -9;
    assert(3) higherOrderFunction(globalRef) == 3;
    assert(4) higherOrderFunction(:add) == 3;
    assert(5) anotherHOF(:add) == 9;

  } testFC(test case)

  int8_t higherOrderFunction((int8_t, int8_t)=>(int8_t) fun) {
    return fun(1, 2);
  } higherOrderFunction (function)

  int8_t anotherHOF(twoIntArgsReturningInt fun) {
    return fun(4, 5);
  } anotherHOF (function)
}
```

Multiple Mappings

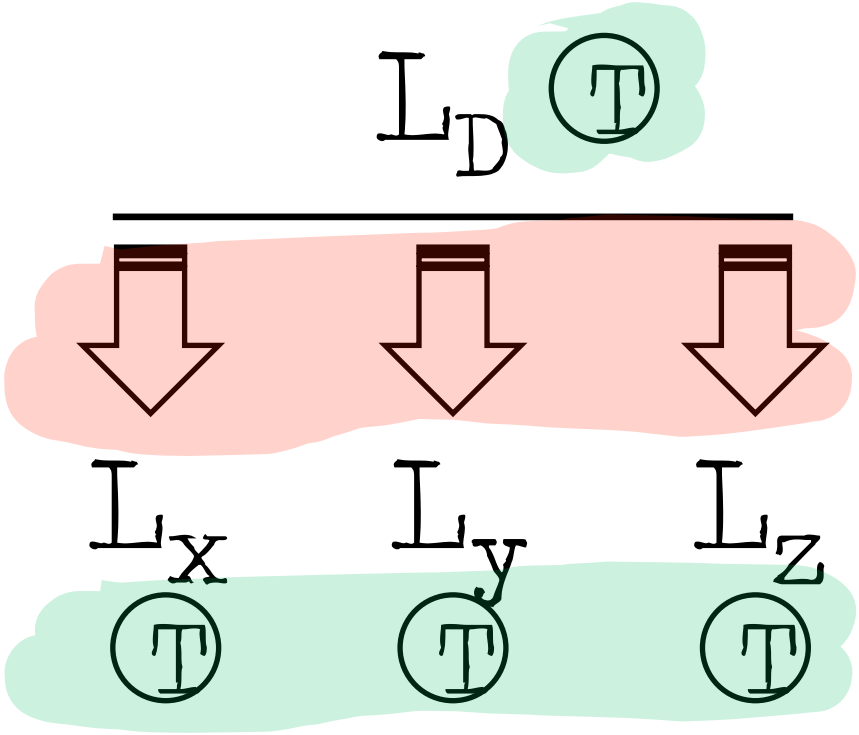
… at the same time



Similar Semantics?

Multiple Mappings

… at the same time

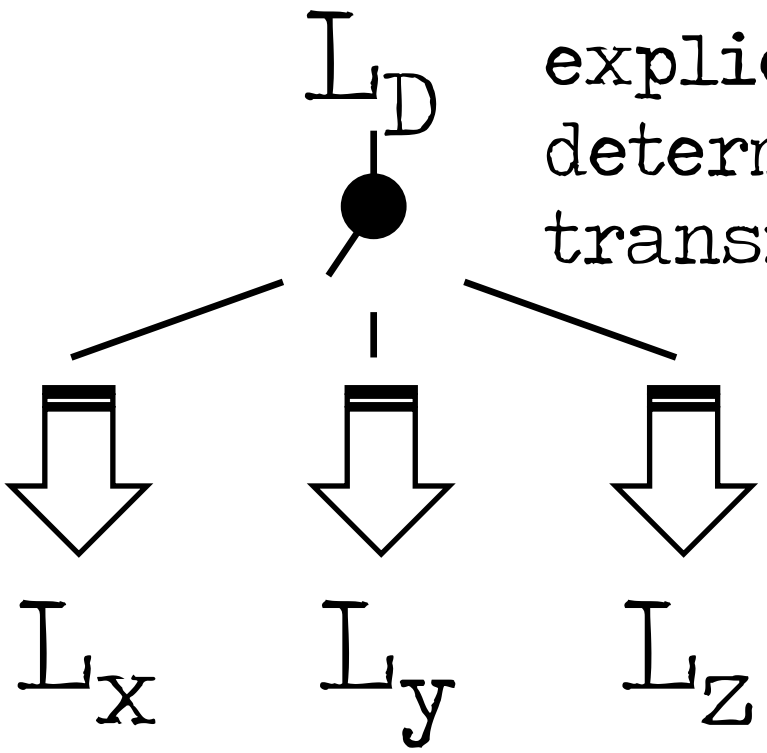


Similar Semantics?

all green!

Multiple Mappings

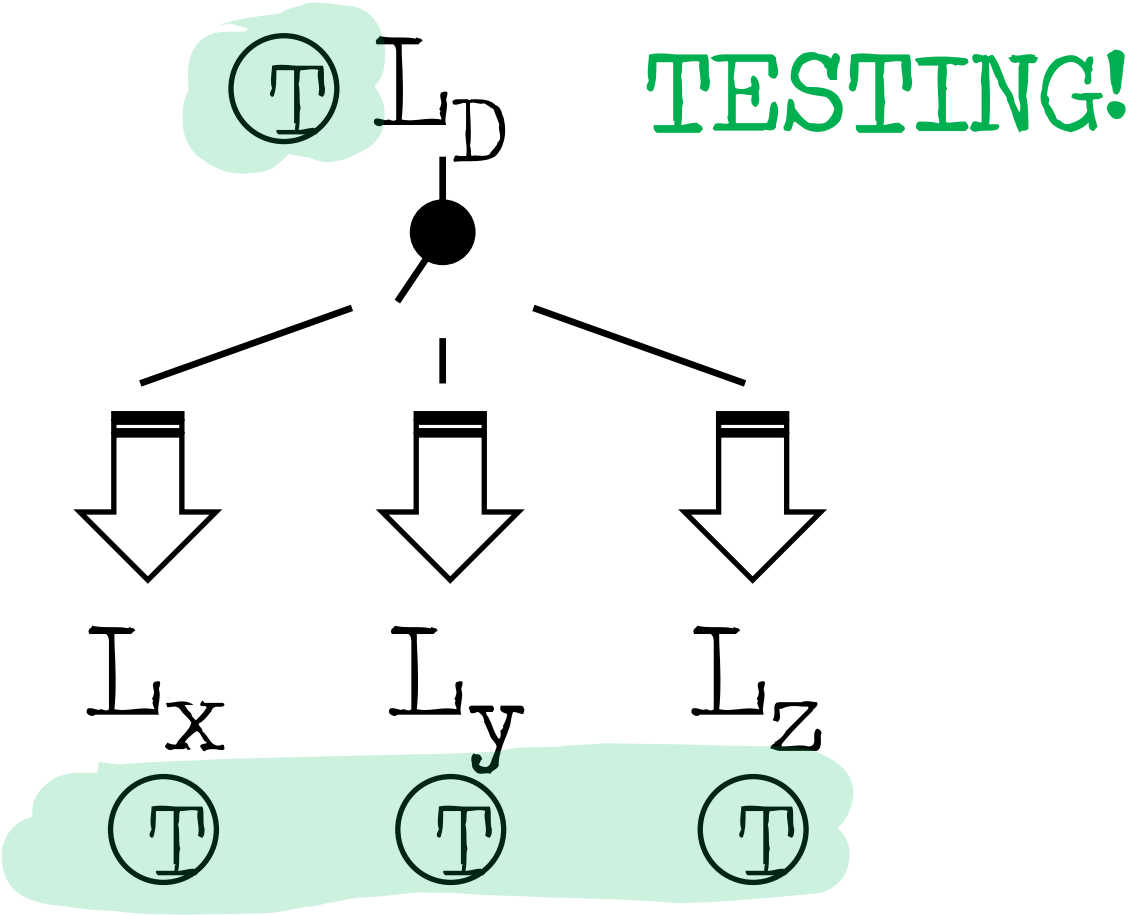
… alternatively, selectably



Extend L_D to include explicit data that determines transformation

Multiple Mappings

… alternatively, selectably



Generators:
What do I do if I cannot execute the programs?

Generators:
What do I do if I cannot execute the programs?

Then (and only then!) use structure analysis

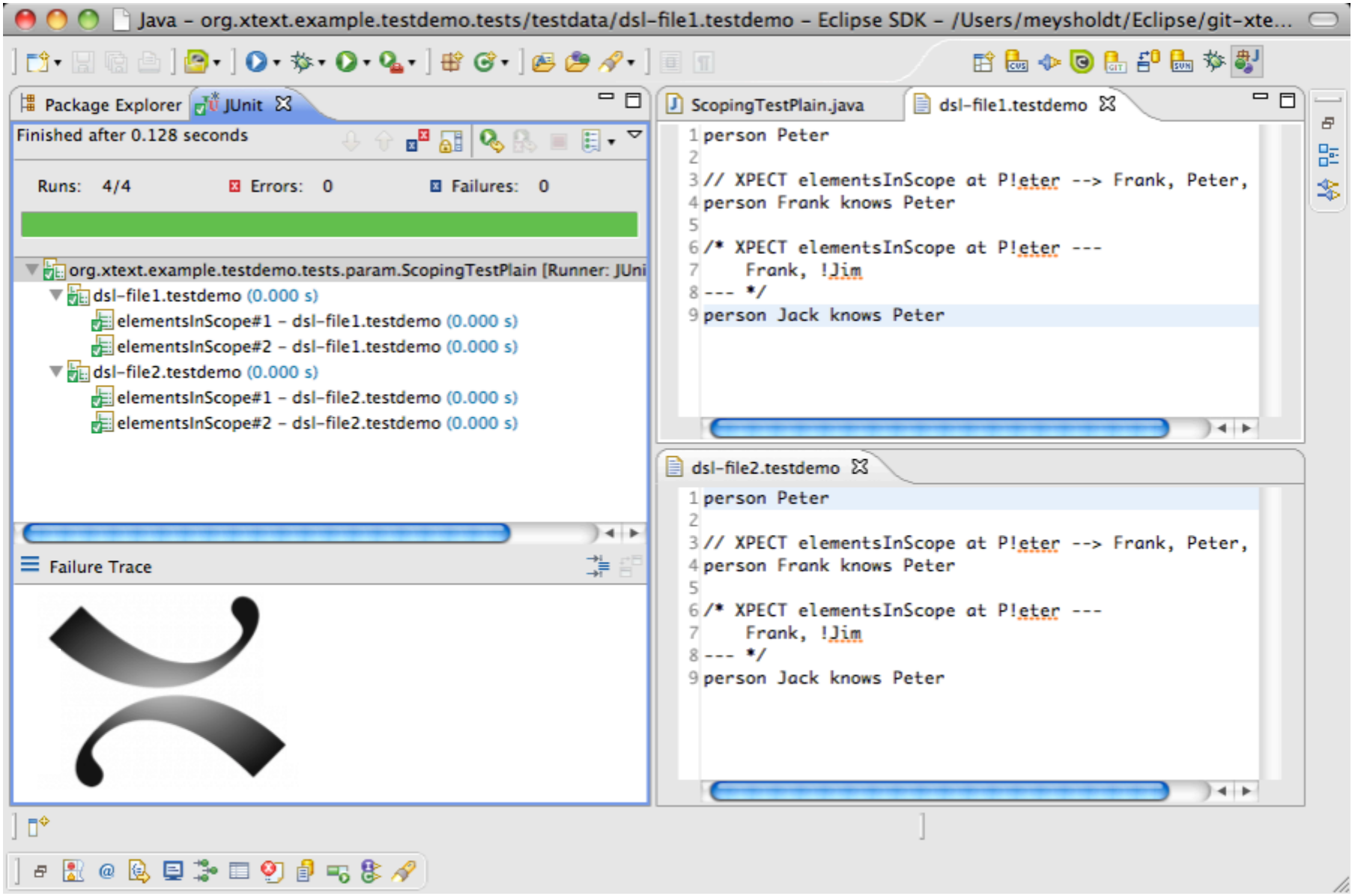
Does the actual editor perform its services correctly?

This is highly tool specific…

Default × | TestCreatingNewVariable × | InitializingNewVariable ×

```
Editor test case InitializingNewVariable
description: no description
before: Variables:
          <cell var <no name> : <no alias>>
result: Variables:
          var myVariale : boolean
code:
  type "myVariable"
  press keys <any>+<VK_TAB>;
  press keys <ctrl>+<VK_SPACE>;
  type "boo"
  press keys <any>+<VK_ENTER>;
```

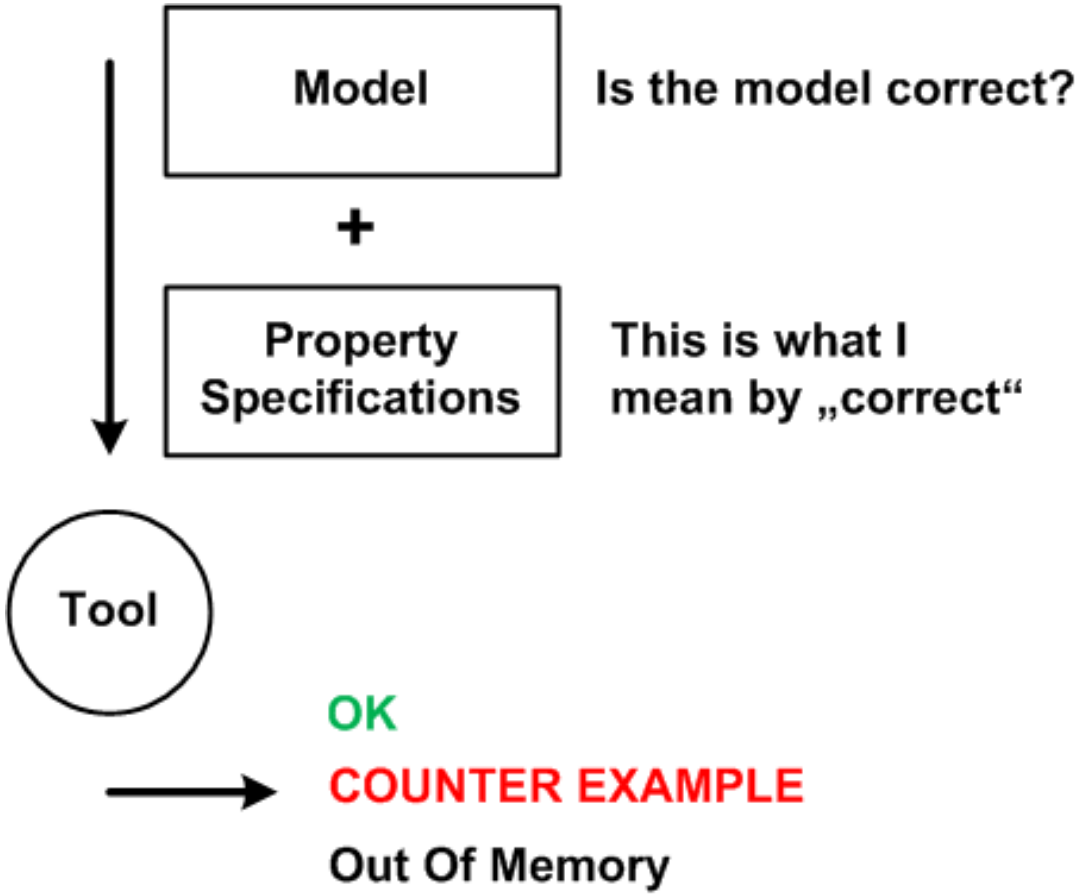
Moritz Eysholdt‘s **Xpect**



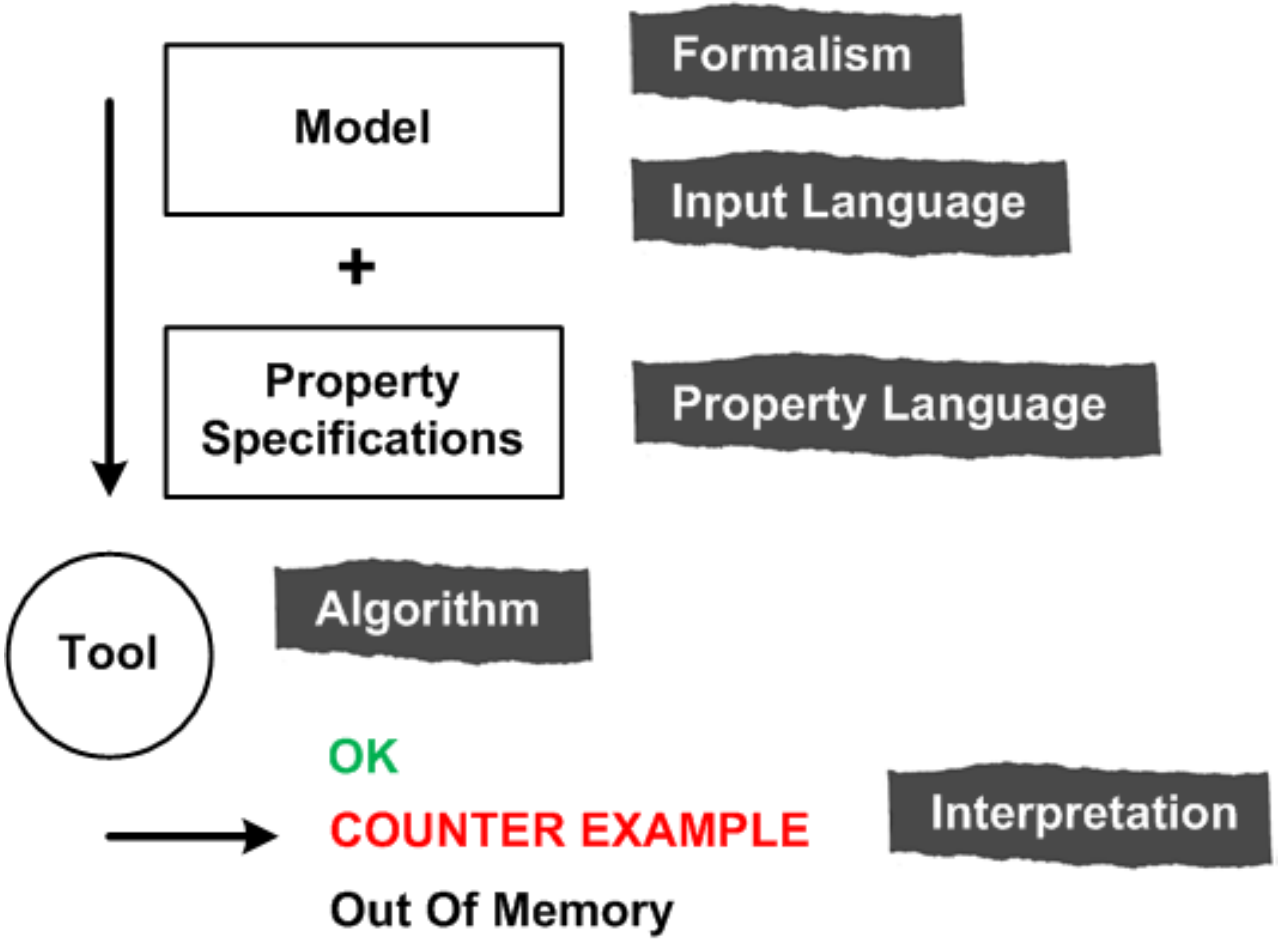
Can I do proofs, and not just "trial and error" testing?

Yes. Formal Methods.

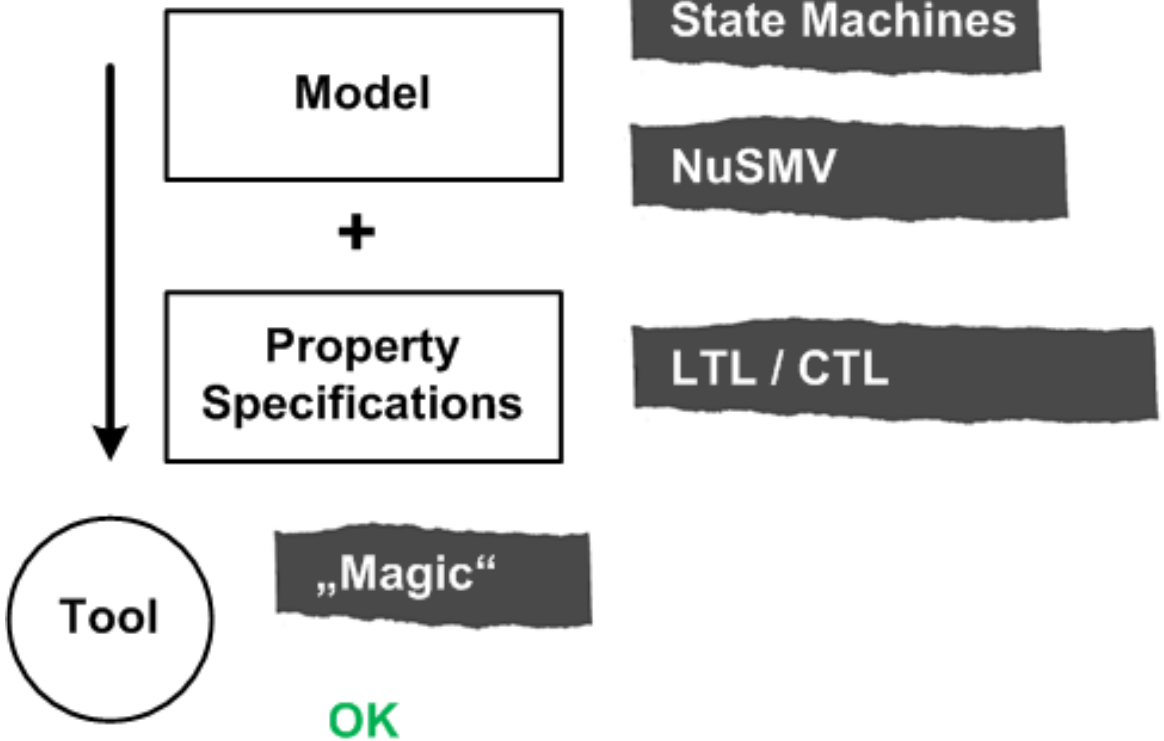
Model Checking



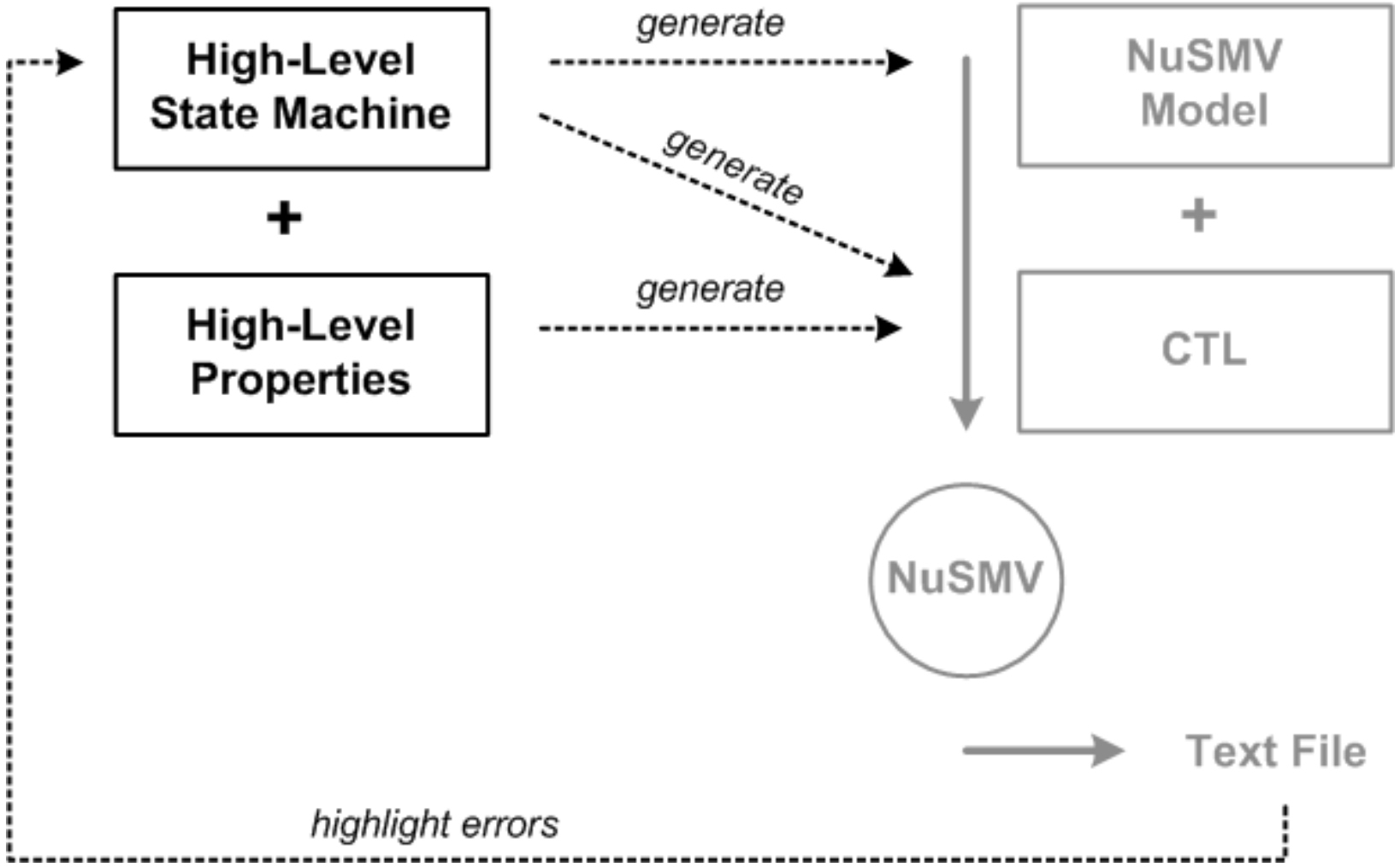
Challenges


Model
+
Property Specifications
Formalism
Input Language
Property Language
Tool
Algorithm
OK
COUNTER EXAMPLE
Out Of Memory
Interpretation

State Based


Model
+
Property Specifications
Tool
State Machines
NuSMV
LTL / CTL
„Magic“
OK
COUNTER EXAMPLE
Out Of Memory

mbeddr Approach



mbeddr Approach

easier to use

hopefully used more

full power: write CTL/ LTL if you want to

Model Checking

```
module TrafficLights from trafficLights imports nothing

  verifiable
  statemachine TrafficLights {
    in timePassed() <no binding>
       pedestrianButtonPressed() <no binding>
    out << ... >>
    vars int[0..2] pedLights = 0
         int[0..2] carLights = 0
    states (initial = bothRed)
      state bothRed {
        on timePassed [ ] -> carsGreen {
          carLights = 2;
          pedLights = 0;
        }
      }
      state carsGreen {
        on timePassed [ ] -> pedGreen {
          carLights = 0;
          pedLights = 2;
        }
      }
      state pedGreen {
        on timePassed [ ] -> carsGreen {
          carLights = 0;
          pedLights = 2;
        }
      }
  } end statemachine
}
```

	Status	Trace Size
State 'bothRed' can be reached	SUCCESS	
State 'carsGreen' can be reached	SUCCESS	
State 'pedGreen' can be reached	SUCCESS	
Variable 'pedLights' is always between its d...	SUCCESS	
Variable 'carLights' is always between its d...	SUCCESS	
State 'bothRed' has deterministic transitions	SUCCESS	
State 'carsGreen' has deterministic transitions	SUCCESS	
State 'pedGreen' has deterministic transitions	SUCCESS	
Transition 0 of state 'bothRed' is not dead	SUCCESS	
Transition 0 of state 'carsGreen' is not dead	SUCCESS	
Transition 0 of state 'pedGreen' is not dead	SUCCESS	
condition 'carLights==2&&pedLights==2' is...	SUCCESS	

Node	Value

Inspector

com.mbeddr.ext.statemachine.nusmv.structure.VerificationAttribute — Open Concept Declaration

```
verification conditions
  never carLights == 2 && pedLights == 2
```

Model Checking

```
module TrafficLights from trafficLights imports nothing

  verifiable
  statemachine TrafficLights {
    in timePassed() <no binding>
       pedestrianButtonPressed() <no binding>
    out << ... >>
    vars int[0..2] pedLights = 0
         int[0..2] carLights = 0
    states (initial = bothRed)
      state bothRed {
        on timePassed [ ] -> carsGreen {
          carLights = 2;
          pedLights = 0;
        }
      }
      state carsGreen {
        on timePassed [ ] -> pedGreen {
          carLights = 0;
          pedLights = 2;
        }
      }
      state pedGreen {
        on timePassed [ ] -> carsGreen {
          carLights = 2;
          pedLights = 2;
        }
      }
  } end statemachine
}
```

	Status	Trace Size
State 'bothRed' can be reached	SUCCESS	
State 'carsGreen' can be reached	SUCCESS	
State 'pedGreen' can be reached	SUCCESS	
Variable 'pedLights' is always between its d...	SUCCESS	
Variable 'carLights' is always between its d...	SUCCESS	
State 'bothRed' has deterministic transitions	SUCCESS	
State 'carsGreen' has deterministic transitions	SUCCESS	
State 'pedGreen' has deterministic transitions	SUCCESS	
Transition 0 of state 'bothRed' is not dead	SUCCESS	
Transition 0 of state 'carsGreen' is not dead	SUCCESS	
Transition 0 of state 'pedGreen' is not dead	SUCCESS	
Condition 'carLights==2&&pedLights==2' c...	FAIL	5

Node	Value
State bothRed	
carLights	0
pedLights	0
State bothRed	
in_event: timePassed	timePasse)
carLights	0
pedLights	0
State carsGreen	
in_event: timePassed	timePasse)
carLights	2
pedLights	0
State pedGreen	
in_event: timePassed	timePasse)
carLights	0
pedLights	2
State carsGreen	
carLights	2
pedLights	2

Inspector

com.mbeddr.ext.statemachines.structure.State — Open Concept Declaration

```
[states] State "pedGreen"[1162046401483365036] in trafficLights
```

Model Checking

```
module TrafficLights from trafficLights imports nothing

  verifiable
  statemachine TrafficLights {
    in timePassed() <no binding>
       pedestrianButtonPressed() <no binding>
    out << ... >>
    vars int[0..2] pedLights = 0
         int[0..2] carLights = 0
    states (initial = bothRed)
      state bothRed {
        on timePassed [ ] -> carsGreen {
          carLights = 2;
          pedLights = 0;
        }
      }
      state carsGreen {
        on timePassed [ ] -> pedGreen {
          carLights = 0;
          pedLights = 2;
        }
      }
      state pedGreen {
        on timePassed [ ] -> carsGreen {
          carLights = 2;
          pedLights = 2;
        }
      }
  } end statemachine
}
```

	Status	Trace Size
State 'bothRed' can be reached	SUCCESS	
State 'carsGreen' can be reached	SUCCESS	
State 'pedGreen' can be reached	SUCCESS	
Variable 'pedLights' is always between its d...	SUCCESS	
Variable 'carLights' is always between its d...	SUCCESS	
State 'bothRed' has deterministic transitions	SUCCESS	
State 'carsGreen' has deterministic transitions	SUCCESS	
State 'pedGreen' has deterministic transitions	SUCCESS	
Transition 0 of state 'bothRed' is not dead	SUCCESS	
Transition 0 of state 'carsGreen' is not dead	SUCCESS	
Transition 0 of state 'pedGreen' is not dead	SUCCESS	
Condition 'carLights==2&&pedLights==2' c...	FAIL	5

Node	Value
State bothRed	
carLights	0
pedLights	0
State bothRed	
in_event: timePassed	timePasse)
carLights	0
pedLights	0
State carsGreen	
in_event: timePassed	timePasse)
carLights	2
pedLights	0
State pedGreen	
in_event: timePassed	timePasse)
carLights	0

Inspector

com.mbeddr.ext.statemachines.structure.State

```
[states] State "pedGreen"[116204640148
```

http://mbeddr.com

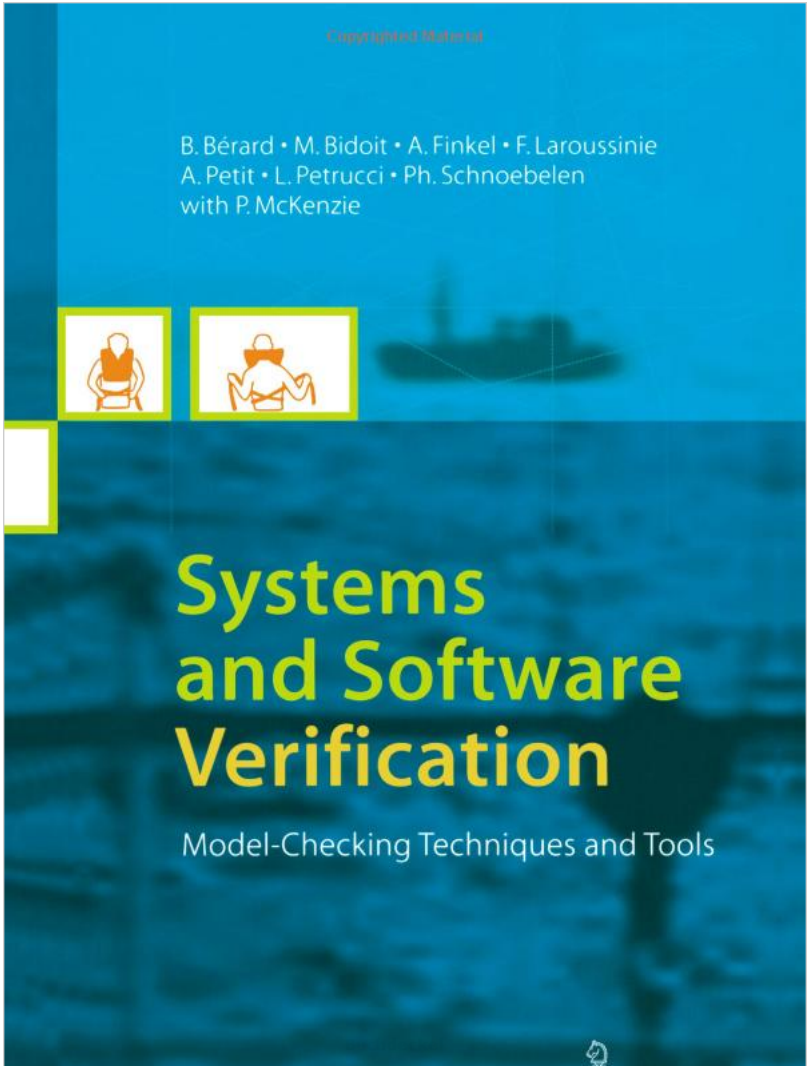
Model Checking

Finds problems in state machines.

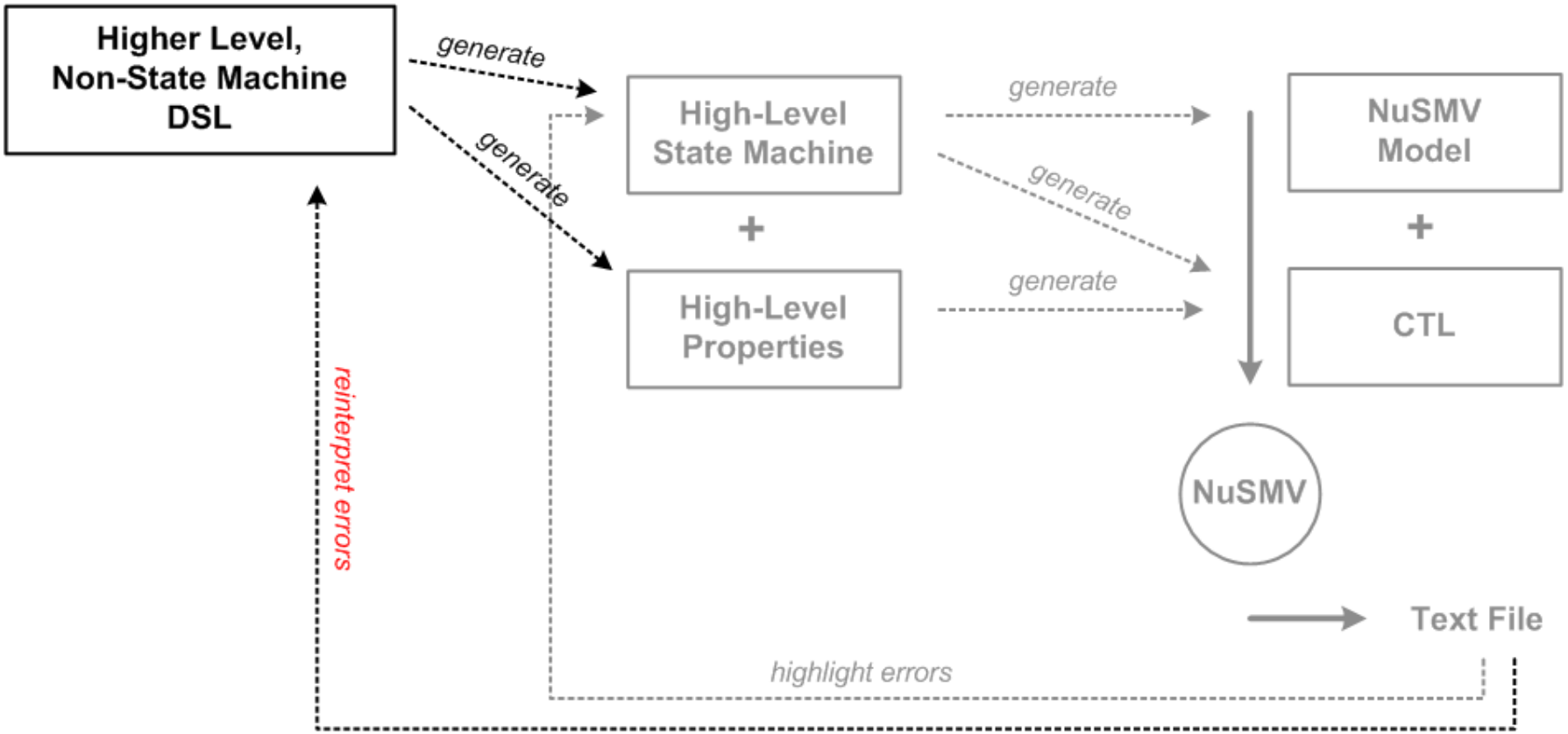
… even ones you didn't think of!

Much more complete than manual testing.

Model Checking



mbeddr Future



mbeddr Future

SAT Solving for

PLE Variability

Decision Table Completeness

Abstract Execution

HARMAN

Klaus Birken

Executing a Program using all possible values and execution paths at the same time

Abstract Execution

HARMAN

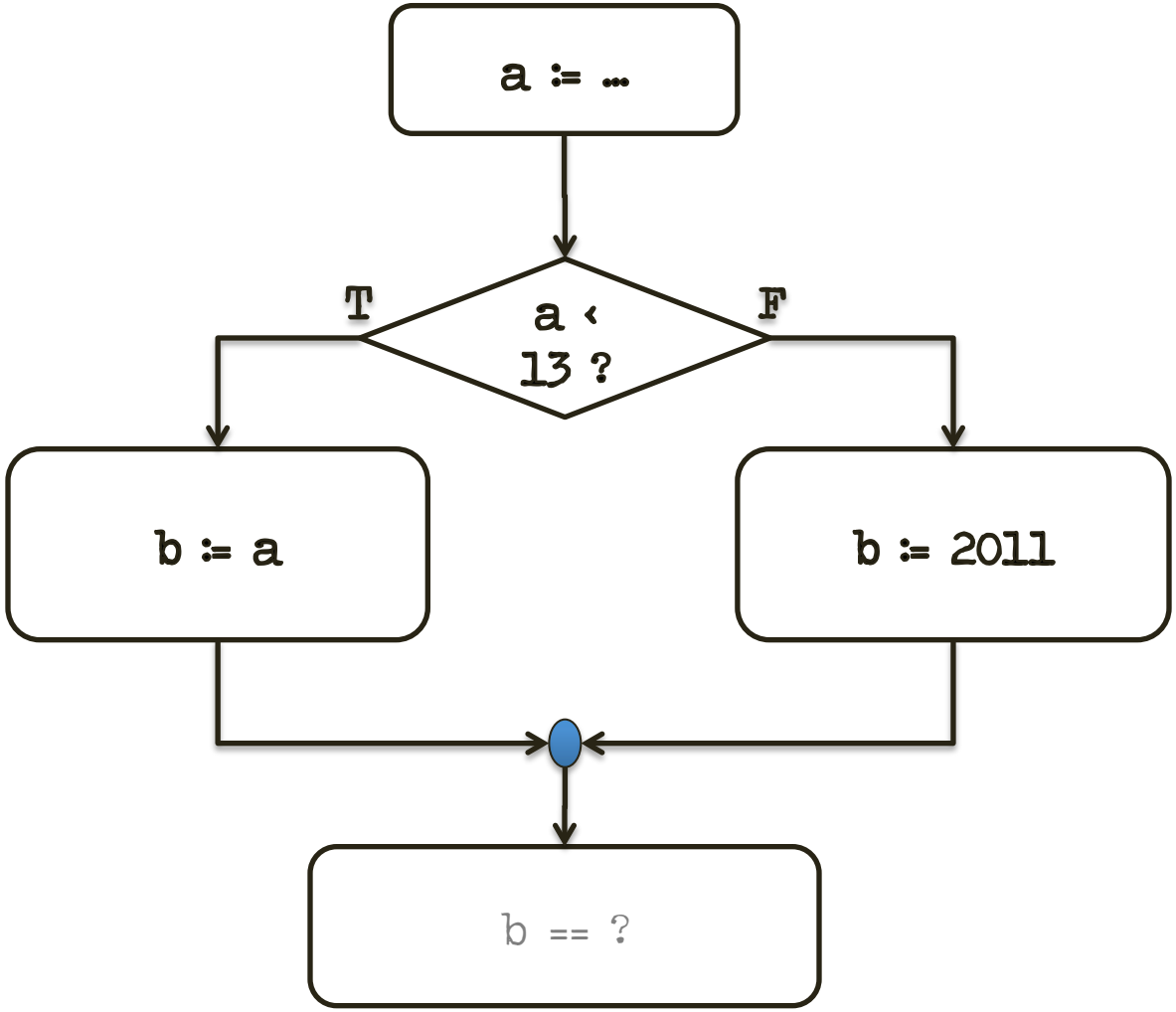
Klaus Birken

$$\{ [1..3] \} + \{ [10..15], 30 \} =$$
$$\{ [11..18], [31..33] \}$$

Abstract Execution

HARMAN

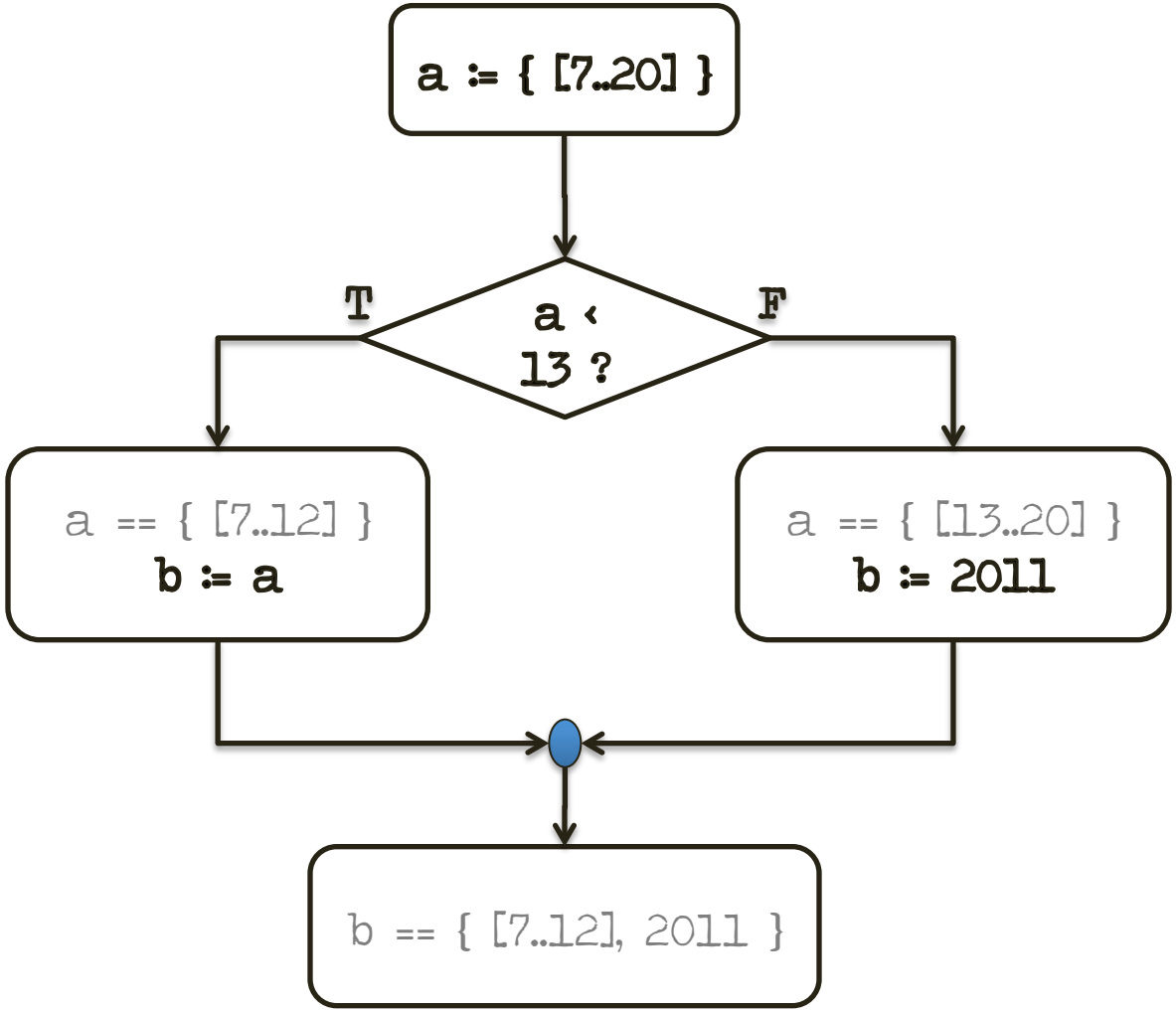
Klaus Birken



Abstract Execution

HARMAN

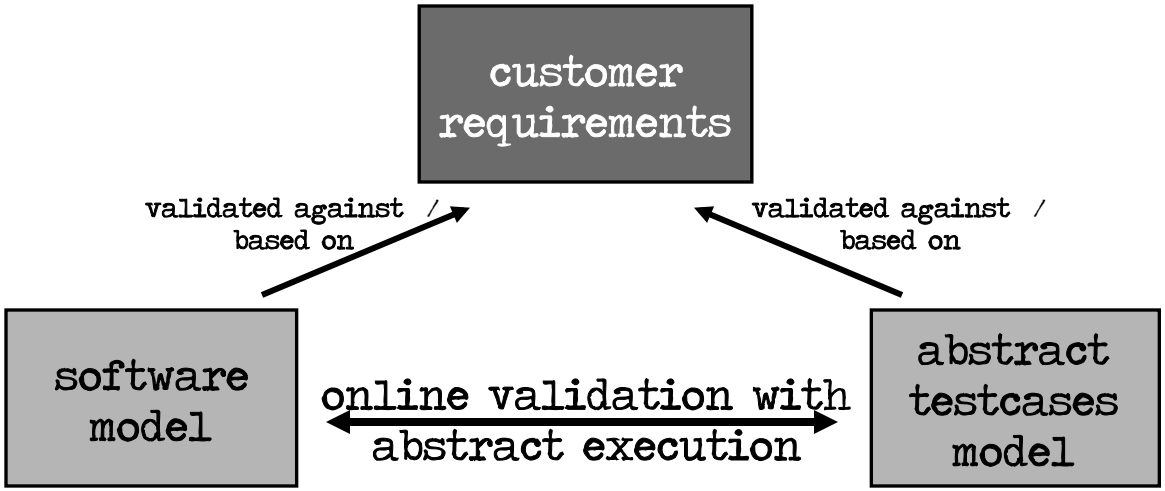
Klaus Birken



Abstract Execution

HARMAN

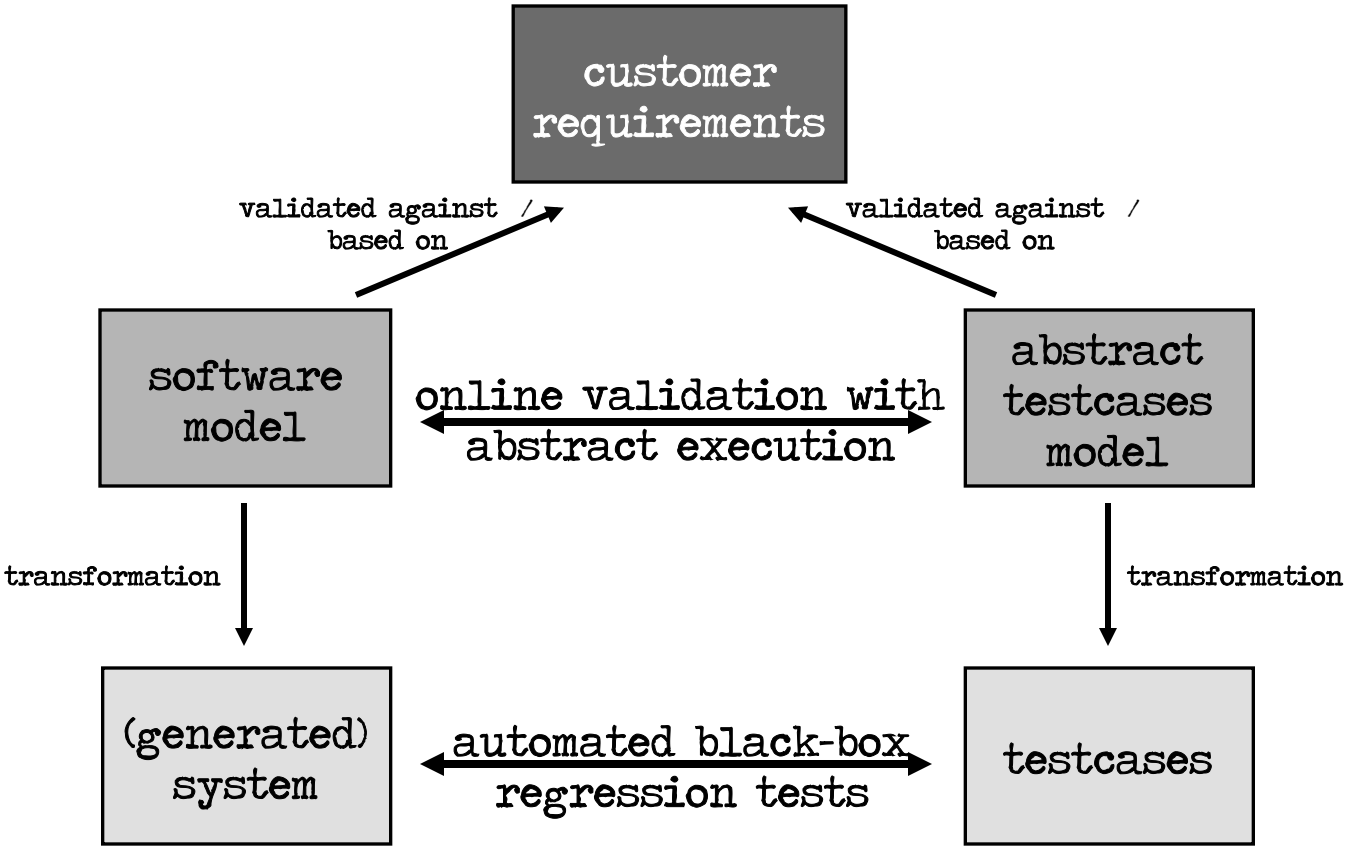
Klaus Birken



Abstract Execution

HARMAN

Klaus Birken



Abstract Execution

HARMAN

Klaus Birken

test for implementation of Audio_Routing_Control service

abstract parameter specification

specify reaction of underlying subsystem (with timings!)

```
80  // this is an abstract testcase for service Audio_Routing_Control.
81  testcase goodcase_Audio_Routing_Control {
82      request Audio_Routing_Control {
83          $Input = {[0..6]};
84          $Output = {[0..5]};
85      };
86
87      dsiresponse controlAudioRouting after [0..999] {
88          responseCode = {0};
89          audioResult = {30};
90      }
91
92      posResponse {
93          $Result == {[0..99]};
94      };
95  }
```

Return param $Result is too generic: {[0..99]} (expected: {30})

expected test result: positive response with
0 <= $Result < 100

Warning: implementation will respond with concrete value

Abstract Execution

HARMAN

Klaus Birken

```
266 /**
267  * Implementing service 0x2F_1210_03_Audio_Routing_Control
268  */
269 service 0x2F_1210_03_Audio_Routing_Control Audio_Routing_Control
270 {
271     action
272     {
273         controlAudioRouting : dsirequest DAVTAudioController.DAVTAudioDiagnosis.controlAudioRouting
274         {
275             #audioInput = cast<UInt8>($Input);
276             #audioOutput = cast<UInt8>($Output);
277         }
278     }
279     // add further event handlers and functions here
280     on controlAudioRouting
281     {
282         if ( #responseCode == DAVTAudioDiagnosis.EResponseCode::eOK )
283         {
284             $Result = cast<Audio_Routing_Result>(#audioResult);
285             posResponse;
286         }
287         elseif (#responseCode == DAVTAudioDiagnosis.EResponseCode::eSERVICE_NOT_SUPPORTED_IN_DIAG) {
288             negResponse (0x80);
289         }
290         else {
291             negResponse (0x72);
292         }
293     }
294 }
```

code coverage for abstract execution of testcase against this implementation

Abstract Execution

HARMAN

Klaus Birken

fully explore the state space
with abstract test cases

- maximise coverage,
  increase confidence,
  reduce risk

tightly integrate development and test

- iteratively develop implementation
  and test cases, each uncovering
  incompleteness in the other

The End.

This material is part of my upcoming (early 2013) book

DSL Engineering with Language Workbenches

Stay in touch; it will be cheap or maybe even free :-)

www.voelter.de/dslbook

@markusvoelter
+Markus Voelter