

Integrated Language Definition Testing

Lennart Kats Delft University of Technology
Rob Vermaas LogicBlox
Eelco Visser Delft University of Technology

Language Workbenches

The image shows two side-by-side code editors from a Language Workbench interface.

Left Editor (EntityLang.sdf):

- File: EntityLang.sdf
- Module: EntityLang
- Imports: Common
- Exports
- Context-free start-symbols:
- Context-free syntax:
 - "module" ID Definition* →
 - "entity" ID "{" Property* "}" →
 - ID ":" Type →
 - ID →

Right Editor (example.ent):

- File: example.ent
- Module: example
- Entity User:
 - name : String
 - password : String
 - homepage : URL
- Entity BlogPosting:
 - poster : User
 - body : String
- Entity URL:
 - location : String

Testing

The image shows a code editor with two files open:

- EntityLang.sdf**: A context-free grammar definition file. It includes sections for imports, exports, and context-free syntax. The context-free syntax section contains rules for "module", "entity", ":", and "ID".
- example.ent**: An EntityLang module definition file. It defines three entities: User, BlogPosting, and URL. The User entity has properties name, password, and homepage. The BlogPosting entity has properties poster (of type User) and body. The URL entity has a property location.

A red error marker is visible next to the "User" type in the BlogPosting definition, indicating a type mismatch or error.

```
EntityLang.sdf
module EntityLang
imports Common
exports
+context-free start-symbols
+context-free syntax
"module" ID Definition*      ->
"entity" ID "{" Property* "}" ->
ID ":" Type                   ->
ID

example.ent
i module example
entity User {
    name : String
    password : String
    homepage : URL
}

entity BlogPosting {
    poster : User
    body : String
}

entity URL {
    location : String
}
```

“DOES THE TYPE CHECKER CATCH THIS?”

Testing

The image shows a code editor with two tabs open:

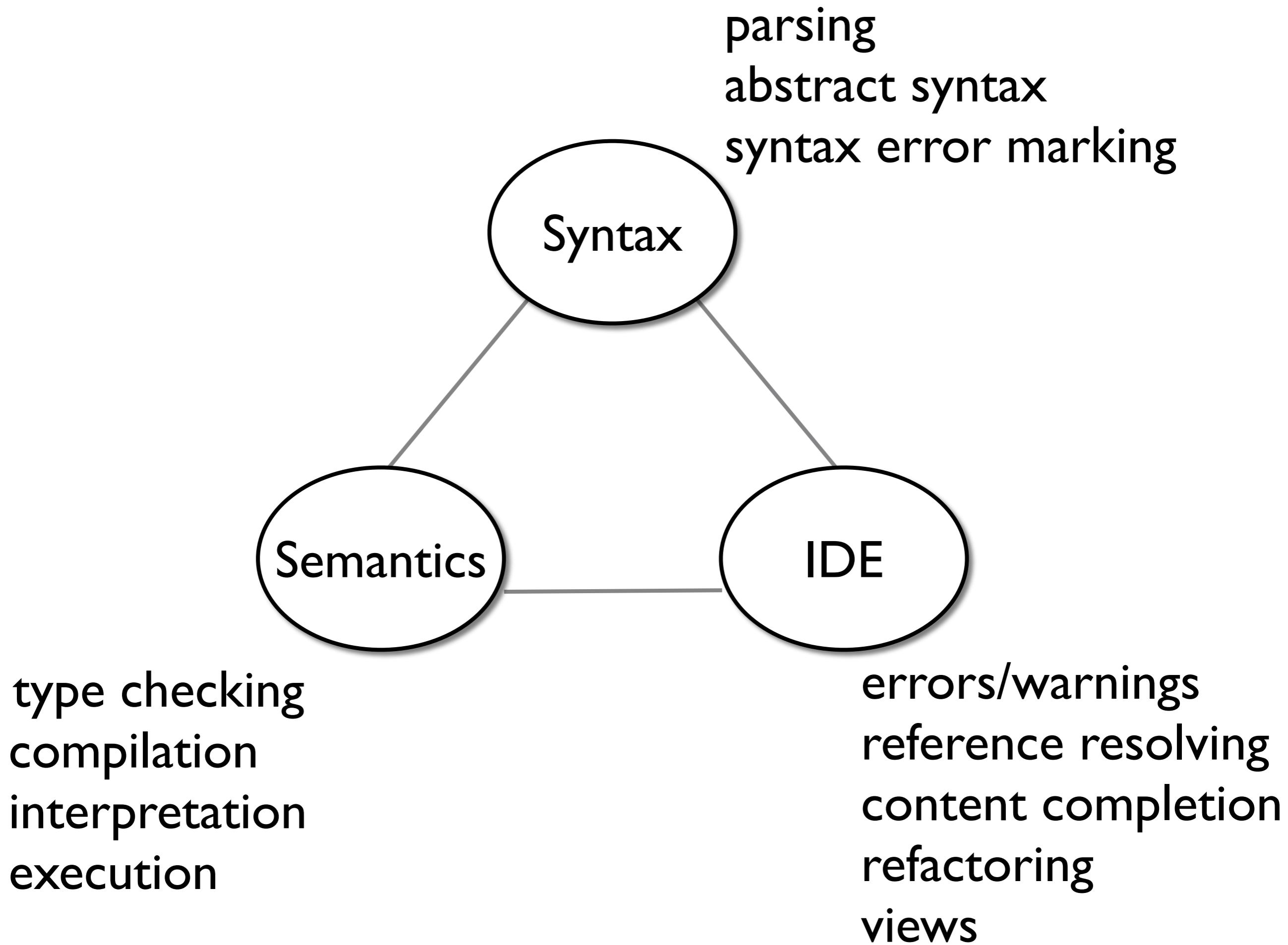
- EntityLang.sdf**: This file defines a grammar for EntityLang. It includes a module declaration, imports for Common, exports, and context-free start-symbols, and a context-free syntax section with rules for "module", "entity", ":", and "ID".
- example.ent**: This file defines entities for User, BlogPosting, and URL. The User entity has properties name, password, and homepage (which is a hyperlink). The BlogPosting entity has poster (User) and body (String). The URL entity has location (String).

A cursor is hovering over the "homepage : URL" line in the User entity definition.

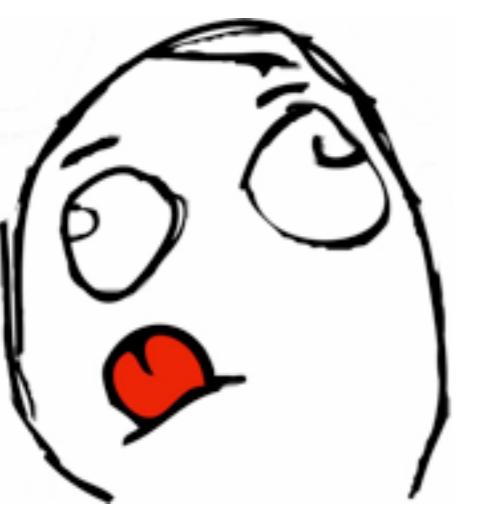
```
EntityLang.sdf
module EntityLang
imports Common
exports
+context-free start-symbols
+context-free syntax
"module" ID Definition*      ->
"entity" ID "{" Property* "}" ->
ID ":" Type                   ->
ID

example.ent
i module example
entity User {
    name      : String
    password : String
    homepage : URL
}
entity BlogPosting {
    poster : User
    body   : String
}
entity URL {
    location : String
}
```

“DOES THIS HYPERLINK POINT TO THE RIGHT PLACE?”



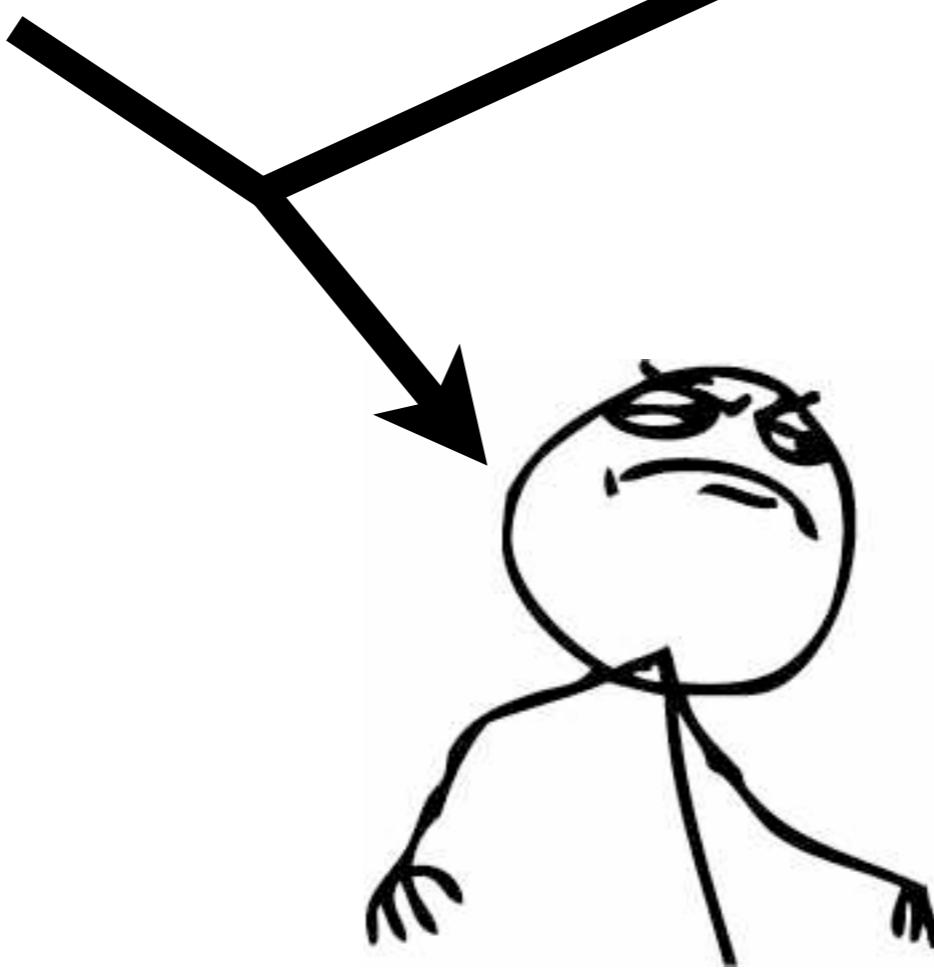
How to test
language services?



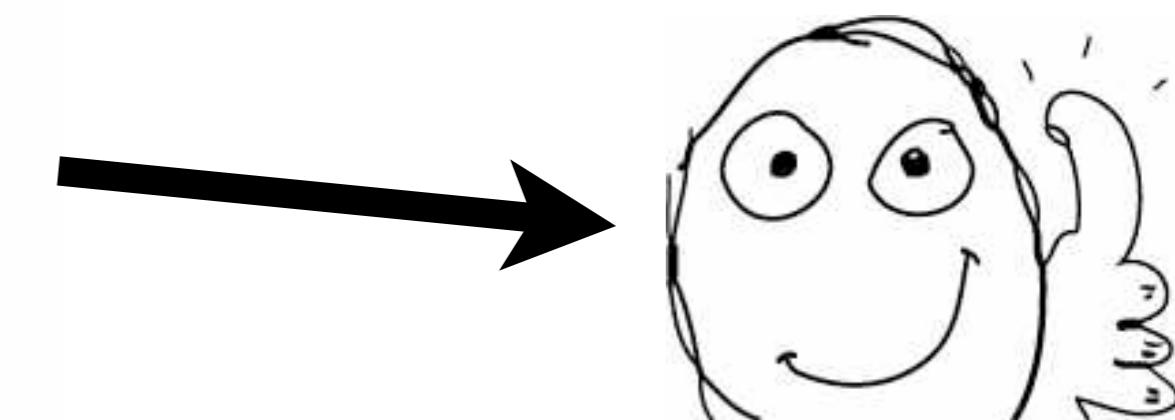
IMPLEMENT
FEATURE, TEST WITH
EXAMPLE



DOESN'T WORK



CLOSE ENOUGH.



DISCARD
TEST

FFF
FFF
FFF
F
U
L

How can we
systematically test
language definitions?

General-Purpose Testing Tools?



A Test Input

```
module Example
```

```
function foo() {  
    bar();  
}
```

```
function bar() {  
}
```

Another Test Input

```
module Example
```

```
function foo() {  
    foo();  
}
```

```
function bar() {  
}
```

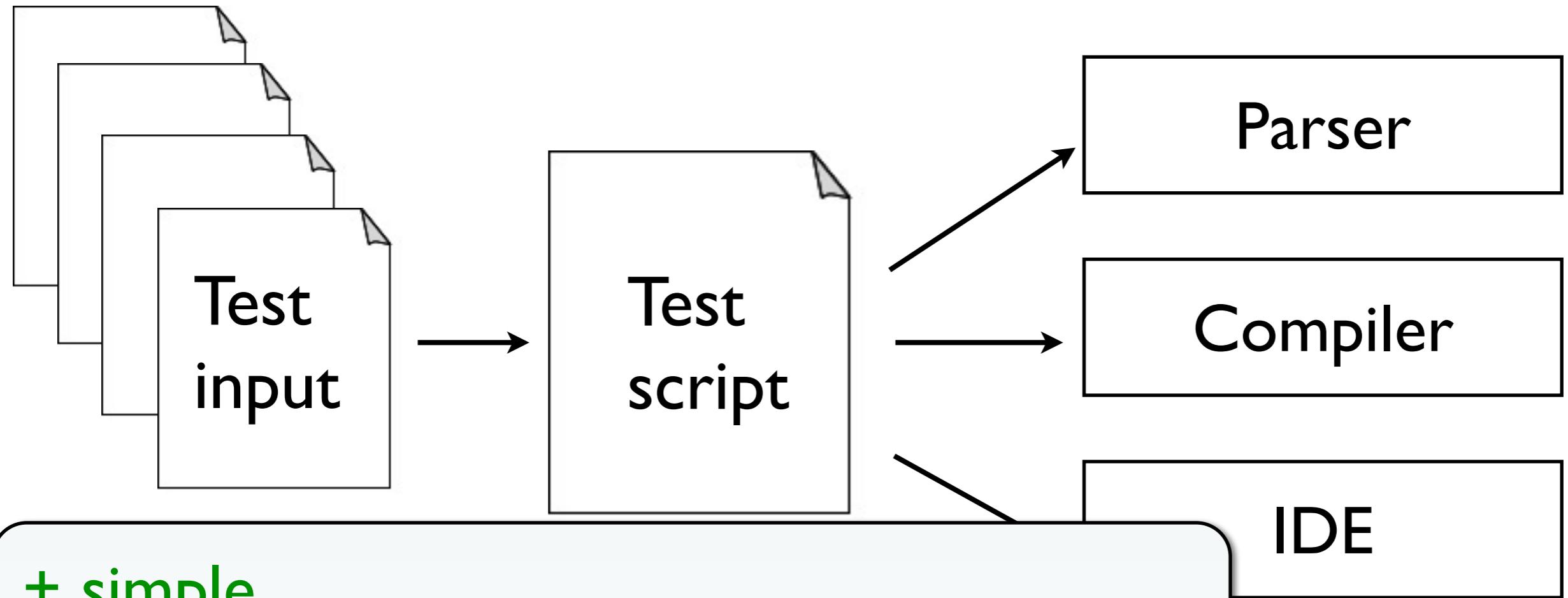
A Negative Test Case

```
module Example
```

```
function foo() {  
    baz();  
}
```

```
function bar() {  
}
```

Automated Testing Infrastructure



- + simple
- language-specific script
- limited expressiveness
- boilerplate code
- ...

Can we design a ***general***
solution for specifying
language tests?

Yes We Can

Generic test specification language

+

Parametrization

Language-Parametric Testing Language (LPTL)

```
module my-tests
```

```
language mob1
```

```
test Cannot assign an integer to a string [[
```

```
module Example
```

```
function test() {  
    var s : String = 1;  
}
```

```
]] 1 error
```

LPTL Opportunities

Expressiveness

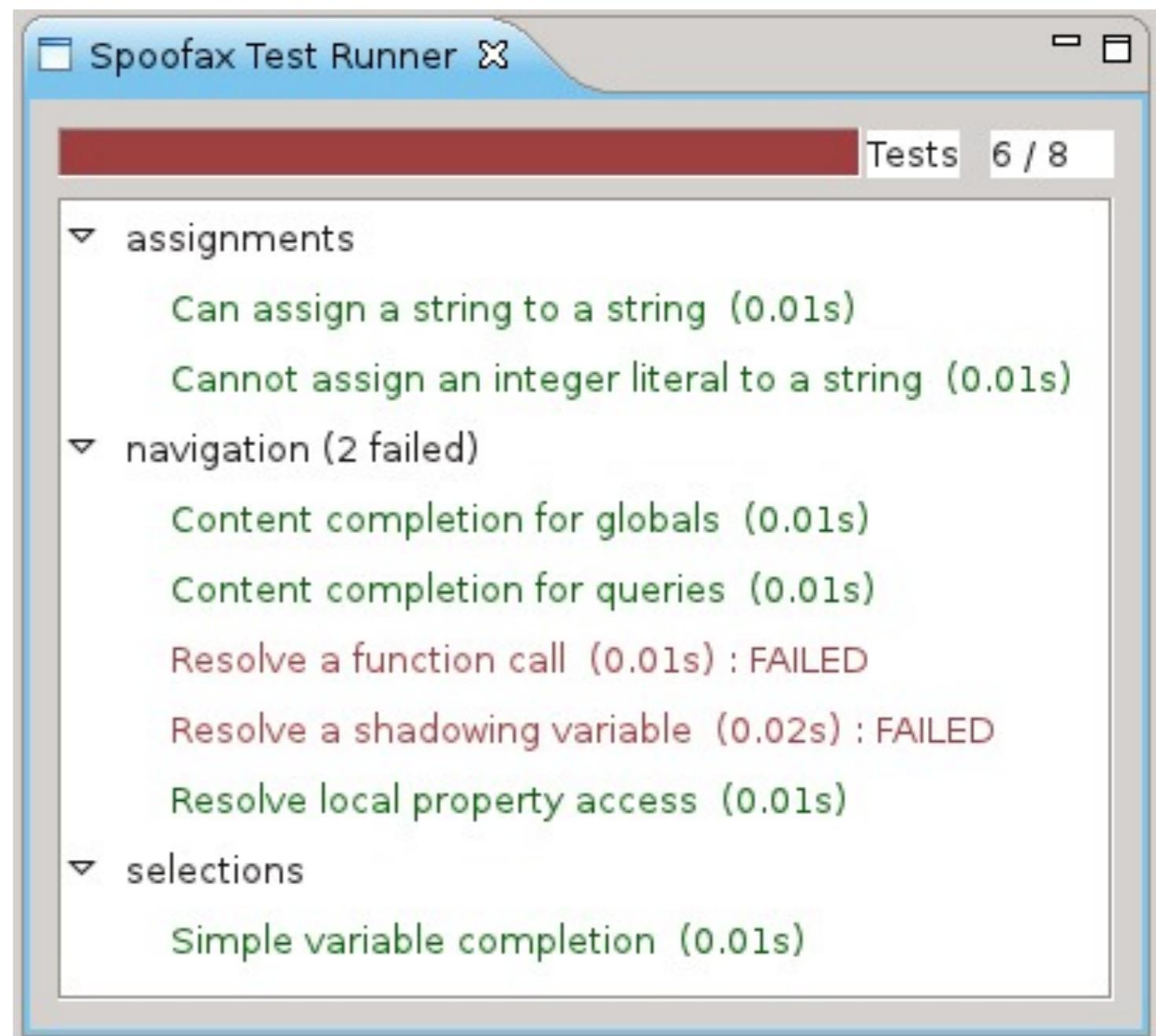
Tool support

LPTL Opportunities (1)

Module system

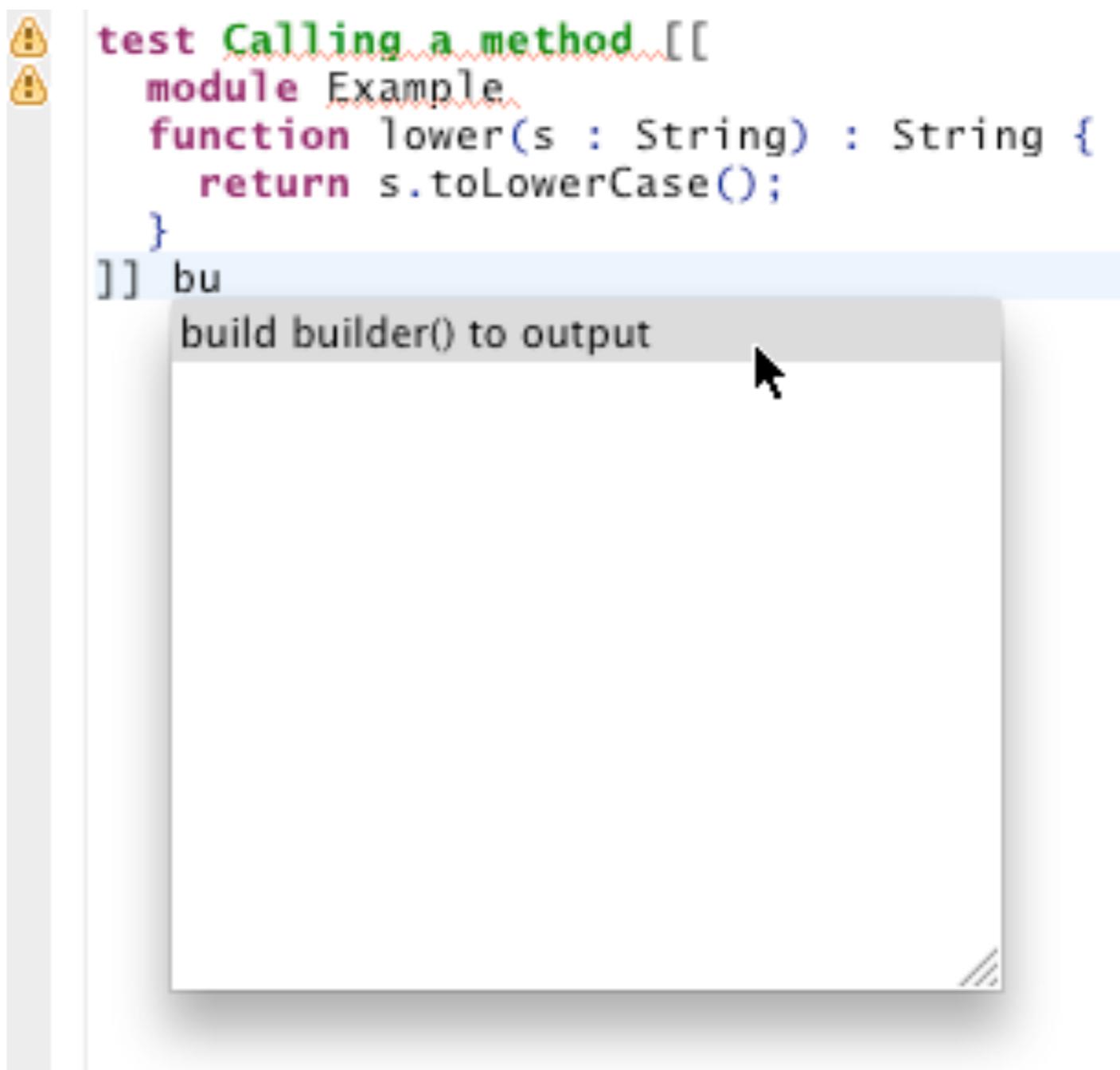
+

GUI Test Runner



LPTL Opportunities (2)

IDE support
for test
specifications



A screenshot of an IDE interface showing a code editor and a tool palette. The code editor contains the following LPTL-like code:

```
test Calling a method [[
  module Example
    function lower(s : String) : String {
      return s.toLowerCase();
    }
]] bu
```

The cursor is positioned at the end of the word "bu". A tooltip window is open, displaying the text "build builder() to output" with a small arrow pointing towards the cursor. The background shows parts of the IDE's user interface, including a toolbar with icons and a status bar.

LPTL Opportunities (3)

Immediate
test
evaluation

The screenshot shows a software interface with two windows. The top window, titled '*1-syntax.spt', contains LPTL test code for a language called 'Calculang'. It includes a successful test for addition ('Add') and an unsuccessful test for multiplication ('Multiply'). The bottom window, titled 'Calculang.sdf', contains the context-free grammar for 'Calculang', defining terminals like INT and '+', and non-terminals like Exp and Plus.

```
*1-syntax.spt
module syntax

language Calculang

test Add [[
  1 + 2
]] parse succeeds

test Multiply [[
  1 * 2
]]
```

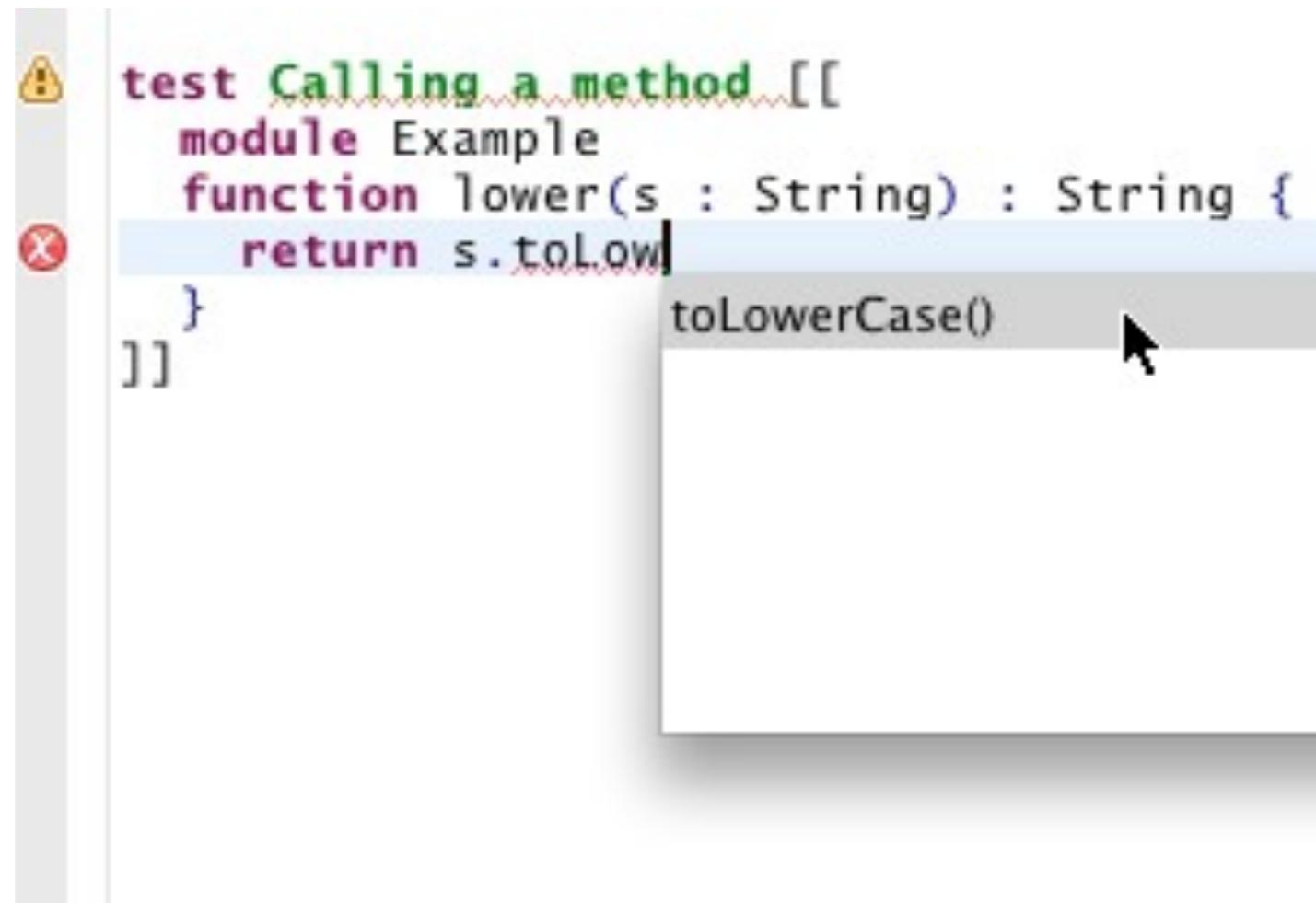
```
Calculang.sdf
context-free syntax

INT -> Exp {cons("Int")}

Exp "+" Exp -> Exp {cons("Plus")}
```

LPTL Opportunities (4)

IDE support
for test
inputs



A screenshot of an IDE interface showing a code editor and a floating code completion tooltip. The code in the editor is:

```
test Calling.a.method [[
  module Example
    function lower(s : String) : String {
      return s.toLowerCase()
    }
]]
```

The cursor is positioned at the end of the line `s.toLowerCase()`. A code completion tooltip is open, displaying the suggestion `toLowerCase()` with a small arrow indicating it is a suggestion.

LPTL Opportunities (5)

Reduced
boilerplate

```
setup [[  
  module Example
```

```
  imports stuff
```

```
  function test() {  
    [...]  
  }  
}]
```

```
test Cannot assign ... [[  
  var s : String = 1;  
]] 1 error
```

LPTL Opportunities (6)

Wide set of
test conditions

1 **error**

2 **warnings**

/expected here/

parse fails

complete ... to ...

resolve ... to ...

refactor ... to ...

build ...

run ...

Testing Syntax (1)

```
test Proper declaration [[  
    var s : String = "a";  
]] parse
```

```
test Java-like declaration [[  
    String s = "a";  
]] parse fails
```

Testing Syntax (2)

```
test Proper declaration [[  
    var s : String = "a";  
]] parse to VarDecl("s", _)
```

```
test Precedence [[  
    3 + 1 * 2  
]] parse to [[  
    3 + (1 * 2)  
]]
```

Testing Error Markers

```
test Variable declaration [[  
    var s : String = "a";  
]] 0 errors
```

```
test Bad variable declaration [[  
    var s : String = 25;  
]] 1 error /wrong type/
```

Testing References

```
test [[  
  module Example  
  
    function foo() {  
      [[bar]]();  
    }  
  
    function [[bar]]() {  
    }  
  ]] resolve #1 to #2
```

Testing Code Generation..?

```
test [[
  function foo() {
    return 3;
  }
]] build generate-javascript to [[
  var foo = function foo() {
    return 3;
  };
]]
```

Testing Execution

```
setup [[  
    application execution  
  
    function test() : Num {  
        // init  
        [...]  
    }  
]]  
  
test Arithmetic [[  
    return 1 + 1;  
]] run run-test to 2
```

Implementation

Spoofax Testing Language

(spofax.org)

Implementation Techniques

Language embedding

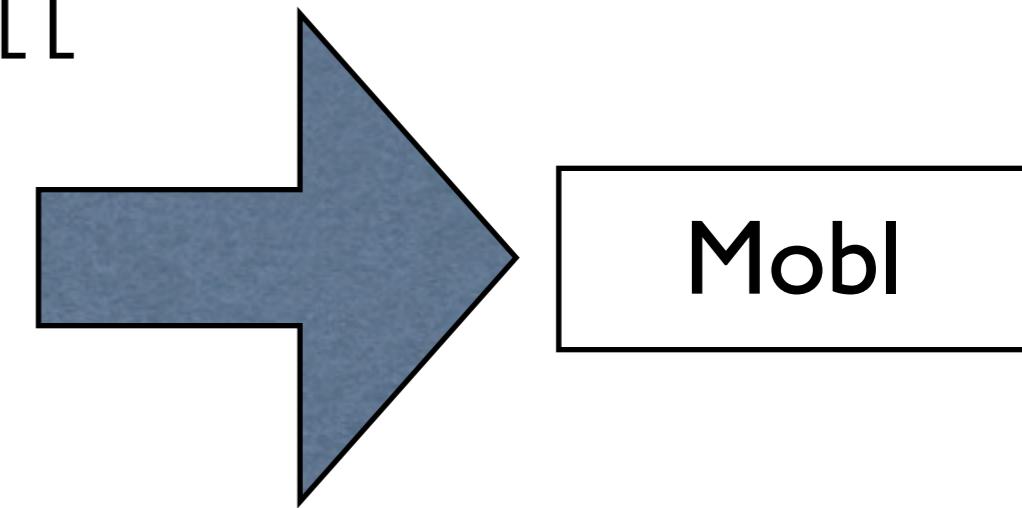
Dynamic instantiation of language services

Language Embedding: Syntax

```
1 module tests
2 language mobl
3 test Java-like declaration [[
4     String s = "a";
5 ]] parse fails
```

Language Embedding: Semantics & IDE (1)

```
test A function call [[  
    function foo() {  
        }  
        fo|  
    }]]
```

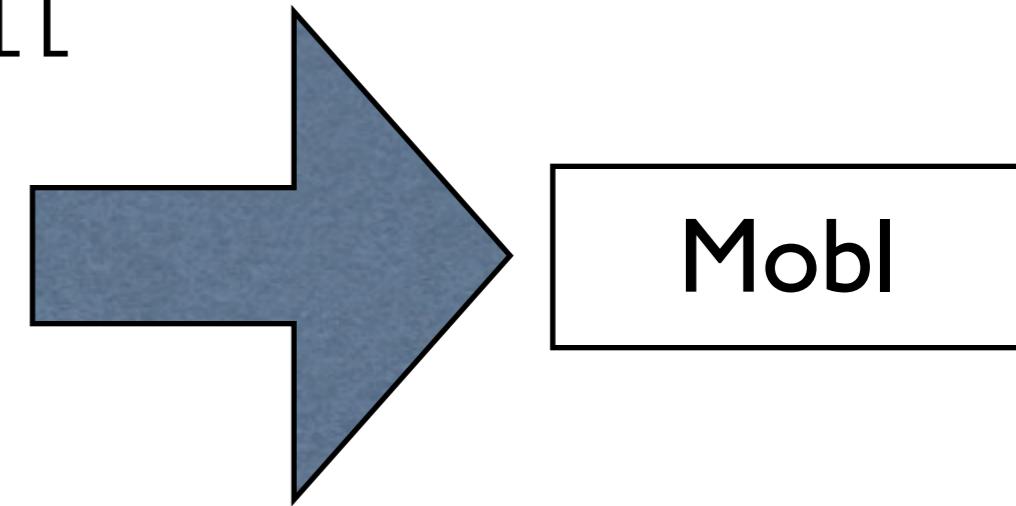


Content complete

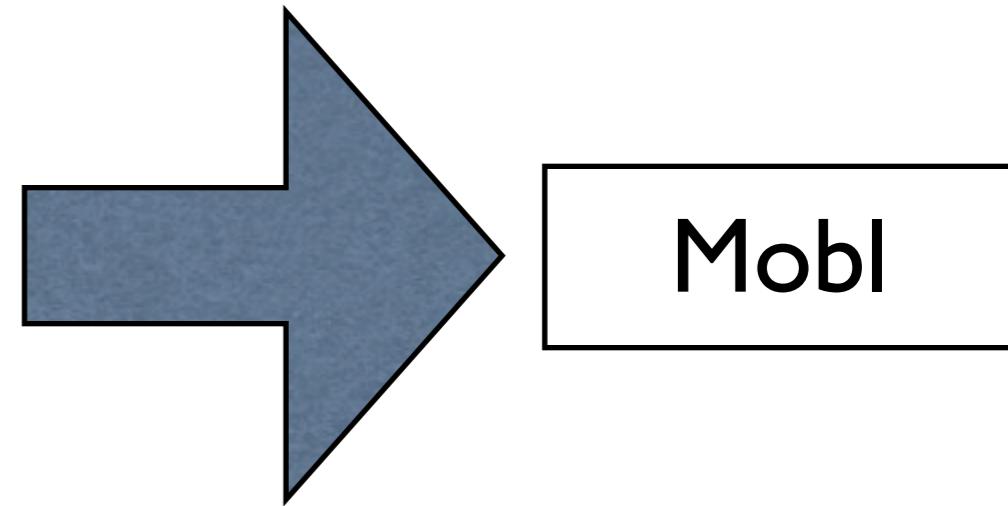
Language Embedding: Semantics & IDE (2)

```
test A function call [[  
    function foo() {  
        }  
        notfoo();  
    }]]
```

No condition;
error not expected



Dynamic Language Service Instantiation



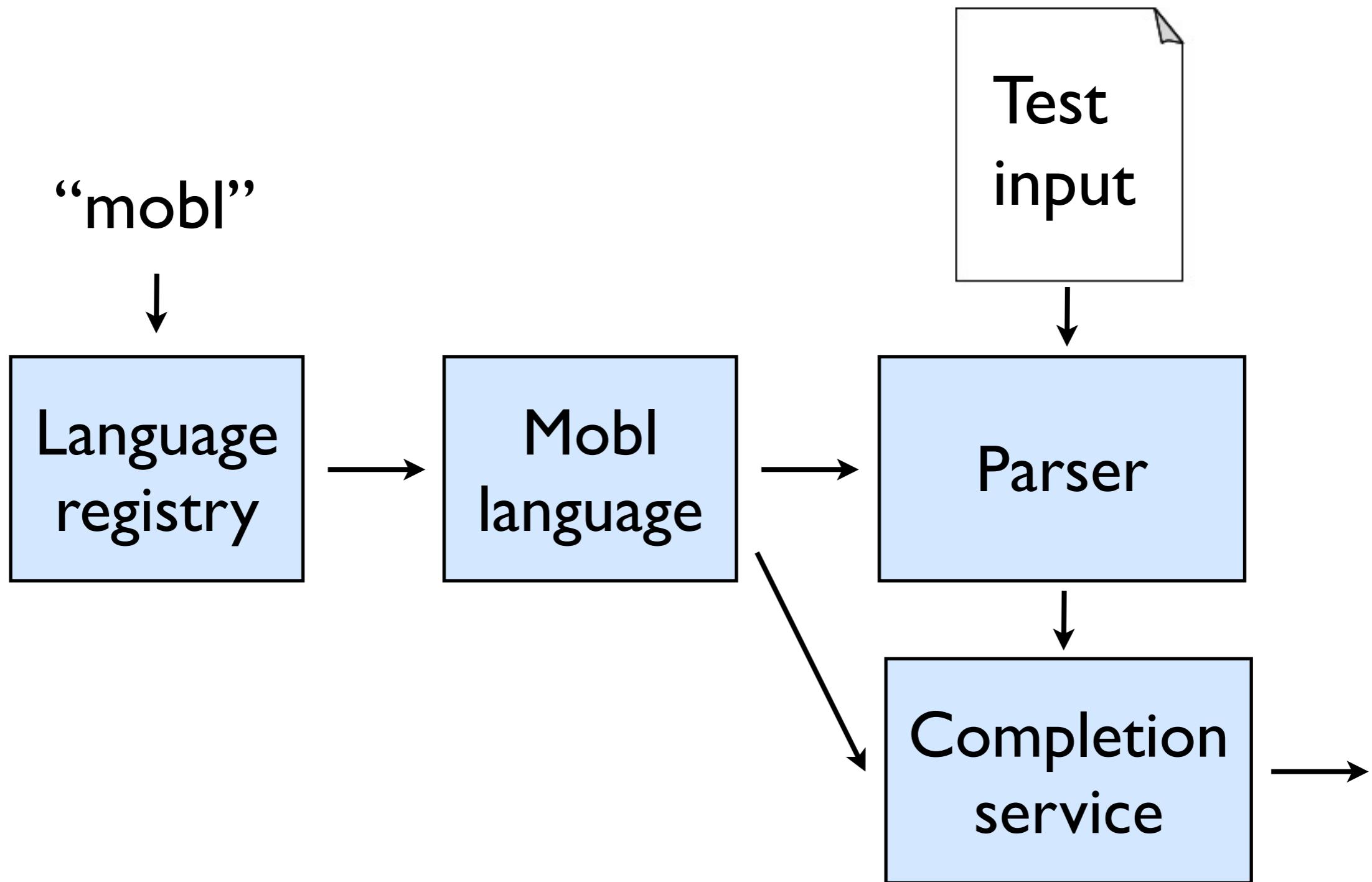
Dynamic Language Service Instantiation

Language registry

Language consists of services

Services have *functional* interfaces

Dynamic Language Service Instantiation



Reflection

- + simple
- + no scripting required
- + IDE helps avoid errors
- + little boilerplate code
- + expressiveness

Conclusions

- General abstraction for language testing
- Explored opportunities in expressiveness and tool support

ADDITIONAL SLIDES

Related: Automatic Test Generation

- Generate tests from grammar
- Requires oracle
- Complementary to our approach

Self-Application

language Spoofax-Testing

```
test Testing testing [[[  
  language Mobi  
  test Testing [[  
    module y  
  ]]  
]]]
```

The Spooftax Language Workbench [OOPSLA 2010]

- Integrated environment for language definition
- Define syntax, semantics, IDE
- Based on Eclipse

www.spooftax.org