IaTestGen

A unit test generator for implementations of the upcoming IEEE interval arithmetic standard written in Java

M.Jedich, M.Nehmeier, A.Dallmann, J. Wolff von Gudenberg

Institute of Computer Science
University of Würzburg
Germany

SWIM 2013

Outline

1 Motivation

2 Domain Specific Language

3 Design

4 Summary

Outline

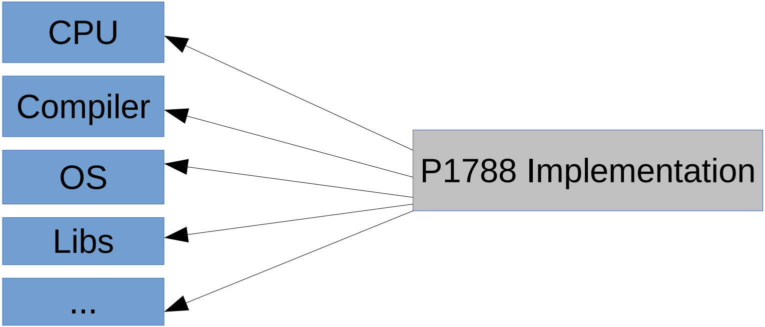
1 Motivation

2 Domain Specific Language

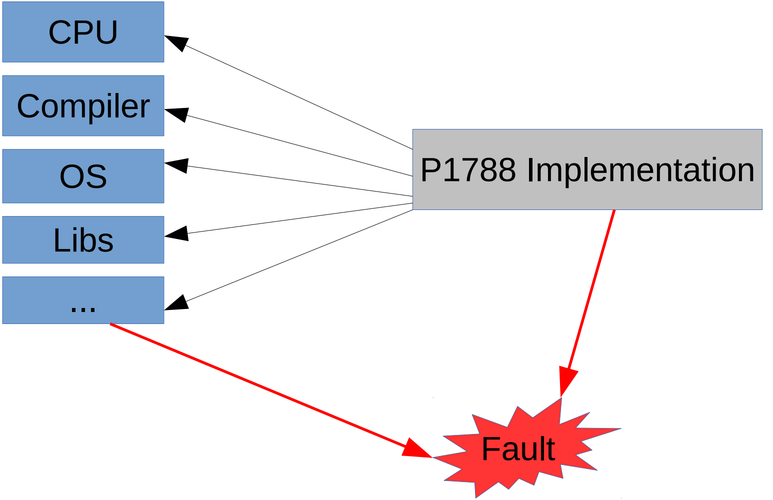
3 Design

4 Summary

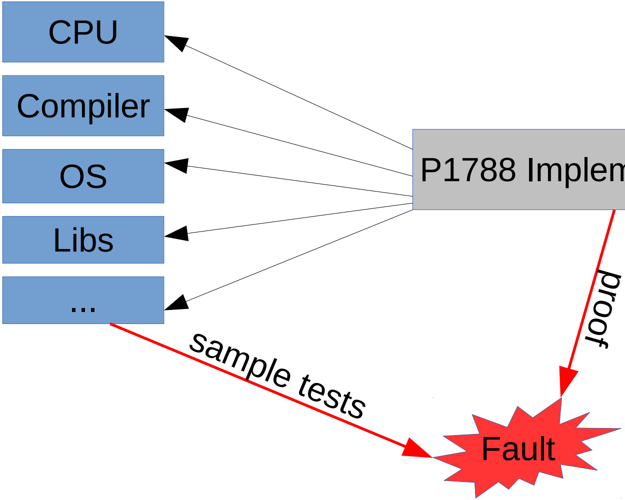
Motivation



Motivation



Motivation


CPU
Compiler
OS
Libs
...
P1788 Implementation
proof
sample tests
Fault

Motivation

How to specify tests for every implementation?

- Use an abstract representation (DSL).
- Provide means to run tests on different platforms and implementations.

Benefits:

- Tests must only be specified once.
- Tests can be written in a well readable format.
- Can be used as regression tests by developers.
- Gives confidence in implementations.

Motivation

How to specify tests for every implementation?

- Use an abstract representation (DSL).
- Provide means to run tests on different platforms and implementations.

Benefits:

- Tests must only be specified once.
- Tests can be written in a well readable format.
- Can be used as regression tests by developers.
- Gives confidence in implementations.

Outline

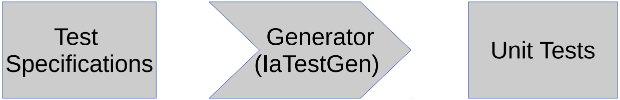
1 Motivation

2 Domain Specific Language

3 Design

4 Summary

The Generation Process



1. Test specifications are written in DSL.
2. Generator parses the specifications.
3. Produces unit tests using a concrete strategy for programming language and implementation.

The Domain-Specific Language

Important Requirements

- An interval data type.
- Number types and a boolean type.
- Number literals in hexadecimal and binary encoding.
- Typed variables.
- Arbitrary many testcases that can be commented.
- Operations with possibly multiple arguments and return values

The Domain-Specific Language

Variables, Numbers and Intervals

Declaration and initialization of different variables:

```
$lowerLimit01 = double: 0x3FC0A3D70A3D70A4;
$upperLimit01 = double: 0x3FCD70A3D70A3D71;
$lowerLimit02 = float: 0xC0D05BC0;
$upperLimit02 = float: 0x414E147B;
$sampleInteger = int: 5;
$sampleInterval = interval<double>[$lowerLimit01, 0x3FDD70A3D70A3D71];
$sampleInfinite = interval<double>[-inf, +inf];
```

The Domain-Specific Language

Operations

Test for correct addition of two intervals:

```
add(interval<float>[3,5], interval<float>[3,7]) = interval<float>[6, 12];
```

Test for correct determination of midpoint:

```
mid(interval<double>[2,12]) = double: 7;
```

The Domain-Specific Language

A more complex example

```
/**
 * Example Domain-Specific-Language.
 * ...
 */

/**
 * Substraction-test.
 * @description
 * Simple substraction test;
 */
sub( interval<int>[10, 15], interval<int>[1, 2] ) = interval<int>[8, 14];


// variables for addition-test
$lowerLimit01  = double: 0x3FC0A3D70A3D70A4;
$upperLimit01  = double: 0x3FCD70A3D70A3D71;
$lowerLimit02 = double: 0x3FD0A3D70A3D70A4;
$additionResult  = interval<double>[$lowerLimit02, 0x3FDD70A3D70A3D71];

/**
 * Addition-test.
 * @description
 * Simple addition test;
 */
add(interval<double>[$lowerLimit01, $upperLimit01],
    interval<double>[0x3FC0A3D70A3D70A4, 0x3FCD70A3D70A3D71]) = $additionResult;
```

The Domain-Specific Language

Code Generation

Example:

```
/**
 * Bisection
 */
$secondOutput = interval<int>[7, 10];
bisect(interval<int>[5, 10]) = interval<int>[5, 7], $secondOutput;
```

The Domain-Specific Language

Code Generation

```
  BOOST_AUTO_TEST_CASE(testcase_02_bisect)
{
// input parameter 1:
int input_01_lower = 5;
int input_01_upper = 10;
interval<int, P> input_01(input_01_lower, input_01_upper);

// expected output parameter 1:
int output_01_lower = 5;
int output_01_upper = 7;
interval<int, P> output_01(output_01_lower, output_01_upper);

// expected output parameter 2:
int output_02_lower = 7;
int output_02_upper = 10;
interval<int, P> output_02(output_02_lower, output_02_upper);

// operation to test: bisect
interval<int, P> lib_output_01 = bisect(input_01).first;
interval<int, P> lib_output_02 = bisect(input_01).second;

...
}
```

The Domain-Specific Language

Code Generation

```
 BOOST_AUTO_TEST_CASE(testcase_02_bisect) {

    ...

// assert function for output 1:
int lo_output_01 = output_01.lower();
int lo_lib_output_01 = lib_output_01.lower();
BOOST_REQUIRE_EQUAL(lo_output_01, lo_lib_output_01);

int hi_output_01 = output_01.upper();
int hi_lib_output_01 = lib_output_01.upper();
BOOST_REQUIRE_EQUAL(hi_output_01, hi_lib_output_01);

// assert function for output 2:
int lo_output_02 = output_02.lower();
int lo_lib_output_02 = lib_output_02.lower();
BOOST_REQUIRE_EQUAL(lo_output_02, lo_lib_output_02);

int hi_output_02 = output_02.upper();
int hi_lib_output_02 = lib_output_02.upper();
BOOST_REQUIRE_EQUAL(hi_output_02, hi_lib_output_02);
}
```

Outline

1 Motivation

2 Domain Specific Language

3 Design

4 Summary

Extending IaTestGen

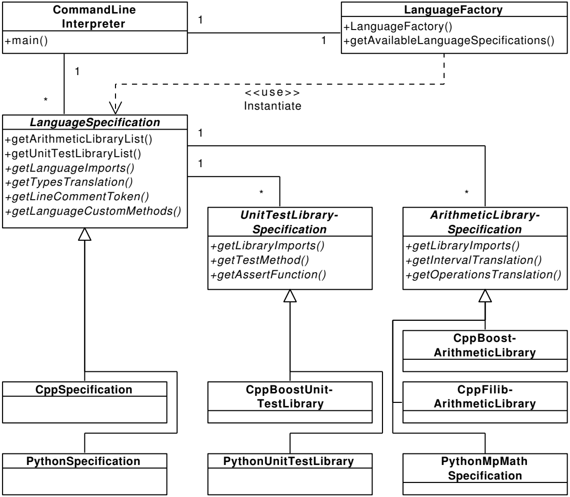
What extension can be provided?

1. Programming Language
2. Testing Framework
3. Interval Arithmetic Library

Templates have to be provided that can be translated into code by the generator.

Extending laTestGen

Design



Extending laTestGen

LanguageSpecification

Provides templates for basic language constructs like:

- Comment token
- Standard imports and definitions
- Basic structure of test file
- Custom methods (will be added to every testfile)

Extending laTestGen

UnitTestLibrarySpecification

Provides templates specific for a testing framework like:

- Imports and definitions
- Basic structure of test suite and test case
- Assert function

Extending IaTestGen

ArithmeticLibrarySpecification

Provides templates specific for a interval library like:

- Imports and definitions
- How to retrieve interval bounds
- Translation from operation name to library function call
- ...

Outline

1 Motivation

2 Domain Specific Language

3 Design

4 Summary

Summary

- DSL can be used to specify a test set for the upcoming interval arithmetic standard.
- IaTestGen can easily be extended to generate tests for many implementations.
- Implementors get an easy way to check their implementations.
- Users can check an implementation in an existing environment.

The End

Thank you for listening!

Any Questions ?