Testing Framework for P

# Components in p

There are three main components in the P framework.

1. P Compiler.
2. Executable Code.
3. Verification of P program.

It is important to test each of these components separately and rigorously.

## P Compiler

The P compiler consist of following important components that needs to be tested properly:

1. Syntax Checking
2. Static Checking
3. Type Checking and Type Inference
4. Generated C code
5. Generated Zing code

## Executable Code

Testing the runtime and its interaction with the generated code consists of writing test cases for the following.

1. Semantics of all the operations in P.
2. Dynamic type checking
3. Tests all the 3 platforms that are supported right now:
   1. User
   2. Driver
   3. Distributed

## P + Zing

Testing of P+ZING consists of writing test cases for the following:

1. Semantics of all the operations in P.
2. Dynamic type checking in Zing
3. Interaction with DelayBounding and other Zing optimizations
4. Interaction with Liveness algorithms (MAP and MACE).

## P Features to test

Test all the P constructs listed in P.4ml:

1. State Machine Level Declarations
   1. Event Declaration
      1. Cardinality of the event
      2. Payload Type
   2. Machine Declaration
      1. Queue Size Constraint
      2. Different types of Machines (Real, Model, Monitor).
      3. Start state defined / not defined.
   3. Variable Declaration
   4. Function Declaration
      1. Model functions: can only be declared in real machines
      2. Wrong type/number of function parameters or return value
   5. Anonymous Function Declaration
      1. Return values of anonymous functions
   6. State Declaration
      1. Function as entry/exit action
      2. Undefined function as entry/exit action
   7. Transition Declaration
      1. Different types of events {String, default, halt}
      2. Different types of transitions normal or push
      3. Functions on transitions Anon or Named.
   8. Do Declaration
      1. Actions on different types of events {String, default, halt}
      2. Different types of actions defer, ignore or anon function or named function.
   9. Assume Max Instances of an event
   10. Assert Max Instances of an event.
   11. Annotations in P language
       1. Zing Annotation for seal/unseal
       2. Zing Annotation for state coverage information.
2. P Statements
   1. Push statement
   2. Dynamic creation of machines using New
   3. Raise Statement
   4. Send Statement
   5. Monitor Invocation
   6. Function Call
   7. Skip and Pop Statement
   8. Assertions
   9. Binary Statements (remove, insert and assign)
      1. On all data types.
      2. Assign: invalid LHS, …
   10. Return statement
   11. While Statement
   12. ITE Statement
3. P Expressions
   1. New expression.
   2. Function call as an expression
   3. Different Primitive expressions
      1. This, trigger, payload, nondet, fair nondet, null and halt
      2. Unary Expression
         1. Not, neg, keys, values, sizeof
      3. Binary Expression
         1. Add, sub, mul, division, and, or, eq, neq, lt, le, gt, ge, idx and in
      4. Field Access
         1. Tuples and NamedTuples
         2. Bad field names
      5. Default Operation
         1. For all data types
      6. Cast operator (as)
4. Data types in P
   1. Creation of complex data types
   2. Subtype relation among data types: assignments, …
   3. Passing variables as payloads.

## INTEGRATION TESTS: Interaction between featURES

1. Precedence Relations
   1. Local variables > Variables > Events
   2. Transitions > Actions > { Ignore, defer }
2. No raise, or pop or call in functions.
3. Calls should always terminate with a pop.

## Combined Tests

1. Duplicates: event definitions, machine declarations, variable declarations, transitions over the same event, …

## Test cases architecture

The folder structure for the test cases would look like following

* Root
  + State Machine Level Declarations
  + Statements
  + Expressions
  + Data types
  + Integration Tests
  + Combined Tests

Each subfolder listed above will have the following subfolders (“type of error” level):

* Static Error: Static analysis reports an error (as listed in “P COMPILER” section)
* Dynamic Error: Zinger reports an error
* Correct: Zinger does not report an error

Folders at the “type of error” level might change, according to the requirements in the “Components of P” section of this document. Also, we may want to add more granularity to the “Correct” folder, for example, to separate tests for runtime from tests for (P + Zing).