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 (mostly syntax checks)

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

1. State Machine Level Declarations
   1. Event Declaration
      1. Cardinality of the event:
         1. Assume Max Instances of an event: [syntax](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/MaxInstances_2/MaxInstances_2.p);

number of instances greater than assumed: ["assume 0" case](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/MaxInstances_3/MaxInstances_3.p); “assume 1” case;

number of instances are not greater than assumed: [test1](../../Tst/RegressionTests/Integration/Correct/BangaloreToRedmond_Liveness/BangaloreToRedmond_Liveness.p) (“assume 1”), [test2](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p) (“assume 1”)

* + - 1. Assert Max Instances of an event: [syntax](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/MaxInstances_2/MaxInstances_2.p);

number of instances are greater than asserted: [test1](../../Tst/RegressionTests/Feature1SMLevelDecls/DynamicError/MaxInstances_1/MaxInstances_1.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/TokenRing/TokenRing.p) (“assert 1”), [test3](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_9/PushItself.p)

number of instances are not greater than asserted: [test1](../../Tst/RegressionTests/Integration/DynamicError/BangaloreToRedmond/BangaloreToRedmond.p) (“assert 2”), [test2](../../Tst/RegressionTests/Integration/Correct/BangaloreToRedmond_Liveness/BangaloreToRedmond_Liveness.p) (“assert 2”),

[test3](../../Tst/RegressionTests/Integration/Correct/PingPong/PingPong.p) (“assert 1”), [test4](../../Tst/RegressionTests/Integration/Correct/PingPongDingDong/PingPongDingDong.p) (“assert 1”), [test5](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p) (“assert 1”)

* + 1. Payload Type: see section 4.3 below.
  1. Machine Declaration
     1. Queue Size Constraint: [test1](../../Tst/RegressionTests/Integration/DynamicError/TokenRing/TokenRing.p)
     2. Different types of Machines: Real ([test](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p)1), Model ([test1](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/TokenRing/TokenRing.p)), Monitor ([test1](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p))).
     3. Start state [defined](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p) / [not defined](../../Tst/RegressionTests/Combined/StaticError/Duplicates/Duplicates.p).
     4. Monitors:
        1. Hot/cold states: [test1](../../Tst/RegressionTests/Integration/Correct/PingPongMonitor/PingPongMonitor.p)
        2. Not allowed constructs: [new, send, push, pop, model functions, defer, default trans](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/monitors/monitors.p); **monitors cannot refer to “this”**
     5. No main machine declared: [test1](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/purity/purity.p), [test2](../../Tst/RegressionTests/Feature2Stmts/StaticError/lvalues/lvalues.p), [test3](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p)
     6. Multiple main machines declared: **same names, different names**
     7. No start state in machine: [test](../../Tst/RegressionTests/Combined/StaticError/Duplicates/Duplicates.p), [test2](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/purity/purity.p), [test3](../../Tst/RegressionTests/Feature2Stmts/StaticError/lvalues/lvalues.p), [test4](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p), [test5](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p)
  2. Variable Declaration
  3. Function Declaration
     1. Model functions: can only be declared in real machines ([test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/entryExit_1/entryExit_1.p), [test2](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/monitors/monitors.p))
     2. [Wrong type/number of function parameters or return value](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p)
     3. [Return value has incorrect type (undeclared)](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p)
     4. Data impure functions: [test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/lvalues/lvalues.p), [test2](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/purity/purity.p)
     5. Types of formal and actual parameters: [any/int](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/functionAny/functionAny.p),
  4. Anonymous Function Declaration
     1. Return values of anonymous functions: [cannot return value](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p)
  5. State Declaration
     1. Function as entry/exit action: [with arguments](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/entryExit_2/entryExit_2.p)
     2. Undefined function: [for exit](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/entryExit_2/entryExit_2.p), **for entry**
     3. Function cannot take arguments:[for entry**,** for exit](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/entryExit_2/entryExit_2.p)
     4. [Hot/cold states](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p)
     5. Groups of states[: no error](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/Groups/Groups.p); error
     6. Actions:
        1. [multiple actions over the same event](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/monitors/monitors.p)
        2. [Basic semantics of actions](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p" \o "Actions_1)
        3. Exit actions:

exit actions are executed before "goto" transition:

[test1](../../Tst/RegressionTests/Feature1SMLevelDecls/DynamicError/AlonBug_fails/AlonBug_fails.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_6/SendInExitHandledEvent.p)

**exit actions are executed upon explicit pop**

**exit actions are executed upon implicit pop**

exit actions are not executed: [test1](file:///D:\PLanguage\PLang1001\plang\Tst\RegressionTests\Feature1SMLevelDecls\Correct\AlonBug\AlonBug.p); [test2;](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_4/SendInExitNotExecuted.p)

* 1. Transition Declaration
     1. **Different types of events {String, default, halt}**
     2. Different types of transitions: normal or push:
        1. [Pairwise push/pop in a loop](../../Tst/RegressionTests/Integration/DynamicError/PushStatement_1/PushStatement_1.p" \o "PushStatement_1)
        2. Goto transition: [basic syntax](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p), [undefined event](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p),

Goto with action: [function cannot take arguments](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p),

[transition function not defined](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p), [assigment as action](../../Tst/RegressionTests/Integration/DynamicError/Push_Pop_1/Push_Pop_1.p)

[Tests in the “Integration Tests” section](#Integration_GotoTrans)

* + - 1. Push transition: [overriding of action handlers by a pushed state](../../Tst/RegressionTests/Integration/DynamicError/Actions_4/Actions_4.p);

[inheritance of actions and not deferred-by-default](../../Tst/RegressionTests/Integration/DynamicError/Actions_5/Actions_5.p); [inheritance of actions but not of transitions](../../Tst/RegressionTests/Integration/DynamicError/Actions_5/Actions_5.p); [payload with push transitions](../../Tst/RegressionTests/Integration/DynamicError/Actions_6/Actions_6.p);

[Tests in the “Integration tests” section](#Integration_PushTrans)

**transition function not defined**

* + 1. **Functions on transitions Anon or Named.**
    2. Transition to undefined state: goto transition ([test1](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/AnonFuns/anonFunction.p), [test2](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p)); **push transition;**
    3. **Transition on an undefined event: goto transition, push transition**
    4. Function as action in transition cannot take arguments: goto transition ([test1](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p)); **push transition**
    5. **“default” event**
    6. **“halt” event:**
  1. Do Declaration
     1. Actions on different types of events {String, [default](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/AnonFuns/anonFunction.p), halt}
     2. Different types of actions defer, ignore or anon function or named function.
        1. Do decl: action on named function: [test1](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p),
        2. Explicit “defer”: [test1](../../Tst/RegressionTests/Integration/Correct/BangaloreToRedmond_Liveness/BangaloreToRedmond_Liveness.p)
     3. Do declaration: [action on undefined event](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p), [transition function not defined](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p),

[function cannot take arguments](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p)

* 1. **Annotations in P language**
     1. **Zing Annotation for seal/unseal**
     2. **Zing Annotation for state coverage information.**

1. P Statements
   1. Push and pop statements:

2.1.1. Basic syntax/semantics: [test1,](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p) **“Undeclared state name”**

2.1.2. Deferred-by-default semantics: action (“do”) vs transition (“goto”): [test passes,](../../Tst/RegressionTests/Integration/Correct/Actions_2/Actions_2.p) test fails ([test1](../../Tst/RegressionTests/Integration/DynamicError/Actions_2_fails/Actions_2_fails.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/Actions_3/Actions_3.p)); [push with pop](../../Tst/RegressionTests/Integration/DynamicError/Actions_3/Actions_3.p)

2.1.3. [Overriding of action handlers by a pushed state](../../Tst/RegressionTests/Integration/DynamicError/Actions_4/Actions_4.p)

2.1.4. [push and pop statements with if-while control flow](../../Tst/RegressionTests/Integration/DynamicError/PushStatement_1/PushStatement_1.p)

* 1. [Dynamic creation of machines using New; “](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p" \o "Actions_1)**[Undeclared machine”](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p" \o "Actions_1)**
  2. Raise Statement:
     1. Basic tests: [test1](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p);

[parameter should be an event](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p)

[“raise” in entry handled in the same state](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_1/SendRaiseInEntry.p" \o "SendRaiseInEntry)

* + 1. **raise with non-constant event expression**
    2. **raise with constant event expression: raise has NIL payload, raise has non-NIL payload;**

“[argument 1 of "raise" expects an event value](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p)"

**“**[**invalid payload type in raise (cannot send null value)**](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/monitors/monitors.p)**”** (for raise with constant event expression, NIL payload)

**“invalid payload type in raise" (for raise with constant event expression, non-NIL payload)**

* 1. Send Statement:
     1. Basic semantics tests:

[“send” in entry handled in the same state](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_1/SendRaiseInEntry.p" \o "SendRaiseInEntry)

* + 1. **Send with constant event expression: NIL and non-NIL payload cases**
    2. **Send with non-constant event expression**
    3. **Error rules for send:  
       “argument 1 of ”send” expects a machine value”**

“[argument 2 of "send" expects an event value](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p)"

**“invalid payload type in send (cannot send null value)"**

**“**invalid payload type in send": [test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/sends/sends.p), [test2](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p)

* 1. [Monitor Invocation](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p):
     1. **Rules for monitor with non-constant event expression**
     2. **Rules for monitor with constant event expression**
     3. **Error rules for monitor:**

**“Undeclared monitor”**

**“argument 2 of "monitor" expects an event value"**

**“**invalid payload type in monitor (cannot send null value)": [test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/sends/sends.p)

**“**invalid payload type in monitor”: [test1](file:///D:\PLanguage\PLang1001\plang\Tst\RegressionTests\Feature2Stmts\StaticError\sends\sends.p)

* 1. **Function Statement (FunStmt):**

**rules about conformance of function stmts to function decls: 2 cases**

**errors (make sure the tests are for FunStmt, rather then for FunApp, as in 4.4.3.12):**

**“function not defined" (not the case of DoDecl, as in 1.8.3. above)**

**“function requires arguments"**

**“function arguments have incorrect types"**

* 1. **SKIP statement**
  2. Assertions: [single assert](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p), two asserts in a row: [pass/fail](../../Tst/RegressionTests/Integration/DynamicError/Actions_4/Actions_4.p), [with complimentary conditions](../../Tst/RegressionTests/Integration/DynamicError/PushStatement_1/PushStatement_1.p);

**“"assert" expects a boolean value”;**

* 1. Mutating Statements (ASSIGN, REMOVE, INSERT)
     1. ASSIGN:

**rule about subtype relation between LHS and RHS**

**error: “invalid assignment. right hand side is not a subtype of left hand side":**

**cases of LHS/RHS types:** [event/NIL](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p), [NIL/ERROR](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/monitors/monitors.p), [SeqType(ANY)/INT](../../Tst/RegressionTests/Feature2Stmts/StaticError/sends/sends.p), [INT/ERROR](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p), [NmdTupType/NmdTupType, wrong names](../../Tst/RegressionTests/Feature2Stmts/StaticError/nmdType/nmdType.p); **etc.**

[impure functions in LHS](../../Tst/RegressionTests/Feature2Stmts/StaticError/lvalues/lvalues.p)

* + 1. **REMOVE:**

**rule about types of arguments: seq. and map cases;**

**errors:**

**“remove must be applied to a sequence or map"**

**“index must be an integer"**

**“Index may not be in the domain of the map"**

* + 1. **INSERT:**

**rule about types of arguments: 2 cases**

**errors:**

**“insert must be applied to a sequence or a map"**

**“for insert right syntax is seq += (index value) or map += (key value)"**

**“key must be an integer"**

**“value must be a subtype of sequence type"**

**“key not in the domain of the map"**

**“value not in the codomain of the map"**

* 1. Return statement:

“[function must return a value](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p)”;

**“return value has incorrect type”: declared/used cases**: [int/bool](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p); [NIL/bool](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p); **etc.**

[**“**anonymous function cannot return a value”](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p)

* 1. **While Statement: “"while (...)" expects a boolean value";**
  2. ITE Statement: “"if (...)" expects a boolean value": [test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/entryExit_1/entryExit_1.p); [test2](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p);

1. P Expressions (excluding type-checking – see #4 below).
   1. **New expression.**
   2. Function Application (FunApp): see 4.4.3.12 below.
   3. Different Primitive expressions: compare to 4.4. below: **what else is there to test?**
      1. This, [trigger](../../Tst/RegressionTests/Integration/DynamicError/BangaloreToRedmond/BangaloreToRedmond.p)[, payload](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p), nondet, fair nondet, null and halt
      2. Unary Expression
         1. Not, neg, keys, values, sizeof
      3. Binary Expression
         1. [Add](../../Tst/RegressionTests/Integration/DynamicError/PushStatement_1/PushStatement_1.p), sub, mul, division, and, or, eq([test1](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p)), neq, lt, le, gt, ge, idx and in
      4. Field Access
         1. Tuples and NamedTuples
         2. [Bad field names](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p)
      5. Default Operation
         1. For all data types
      6. Cast operator (as[): payload as machine](../../Tst/RegressionTests/Integration/DynamicError/Actions_1/Actions_1.p), …
      7. NONDE expressions: [nondet choice in real/model/monitor machines and functions](file:///D:\PLanguage\PLang1001\plang\Tst\RegressionTests\Feature2Stmts\StaticError\entryExit_1\entryExit_1.p);

**same test for FAIRNONDET case (modify entryExit\_1)**

1. Data types in P
   1. **Creation of complex data types**
      1. Tuples: [duplicate names in seq decl, in func decl, in payload](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p)
   2. **Subtype relation among data types**: [assignments](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p), …
   3. **Paylod type inference:**

“invalid payload type in send”: [test1](../../Tst/RegressionTests/Feature4DataTypes/StaticError/function_Typos/function_Typos.p), [test2](../../Tst/RegressionTests/Feature2Stmts/StaticError/sends/sends.p), [test3](../../Tst/RegressionTests/Feature4DataTypes/StaticError/payloads/payloads.p)

“invalid payload type in monitor (cannot send null value)”: [test1](../../Tst/RegressionTests/Feature2Stmts/StaticError/sends/sends.p),

"invalid payload type in send (cannot send null value)": [test1](../../Tst/RegressionTests/Feature4DataTypes/StaticError/payloads/payloads.p)

"invalid payload type in raise": [test1](file:///D:\PLanguage\PLang1001\plang\Tst\RegressionTests\Feature4DataTypes\StaticError\payloads\payloads.p)

Tests under \Tst\RegressionTests\Feature4DataTypes\StaticError with the name “payloadXXX”: these test payload type inference which results in error messages related to assignments, maps, tuples, etc.

* 1. **Types of operands in expressions**
     1. **Nullary expressions:** Integer + Boolean + { THIS, TRIGGER, PAYLOAD, NONDET, FAIRNONDET, NULL, HALT }
     2. **Unary expressions: for all all operators: NOT, NEG, KEYS, VALUES, SIZEOF and all applicable types: BOOL, INT, SeqType(dom), SeqType(cod), INT; (lines 246-265 in P.4ml)**

**Error messages to test: “Operator expected a [boolean/integer/map(4 cases)] value”**

* + 1. **Binary expressions:** 
       1. **Cast expressions: “Cast can never succeed”**
       2. **“New” expressions:**

**errors:**

**“Monitors cannot be created with "new""”**

**“Undefined machine type”**

* + - 1. **“Field” operator:**

**errors:**

“Bad field name" ([test1](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p), …)

**“Operator expected a (named) tuple value"**

* + - 1. **{ ADD, SUB, MUL, INTDIV }:**

**errors:**

[“Operator expected first argument to be int”](../../Tst/RegressionTests/Feature3Exprs/StaticError/fields/fields.p)

**“Operator expected second argument to be int”**

* + - 1. **{ LT, LE, GT, GE }:**

**errors:**

**“Operator expected first argument to be int”**

**“Operator expected second argument to be int”**

* + - 1. **{ AND, OR }:**

**errors:**

**“Operator expected first argument to be bool”**

**“Operator expected second argument to be bool”**

* + - 1. **{ EQ, NEQ }**

**errors:**

**“Values cannot be compared because their types are incompatible":**

[bool vs int](../../Tst/RegressionTests/Integration/StaticError/TokenRing_Typos/TokenRing_Typos.p), **etc.**

* + - 1. **IDX:**

**errors:**

**“Indexer must be applied to a sequence or map" (both for SeqType and**

**MapType)**

**“Index must be an integer"**

**“Index may not be in the domain of the map"**

* + - 1. **IN:**

**errors:**

**“Value can never be in the sequence" (both for SeqType and MapType)**

**“Value can never be in the map"**

**“"in" expects a sequence or map" (both for SeqType and MapType)**

* + - 1. **Expr produces a tuple type**
      2. **NamedExprs produces a named tuple type**
      3. **Function application:**

**errors:**[**“**function not defined](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/TransDelc_DoDecl/TransDecl_DoDecl.p)”

**“function does not return a value”**

**“**[function requires arguments](../../Tst/RegressionTests/Feature1SMLevelDecls/StaticError/function/function.p)”

**“function arguments have incorrect types”**

## INTEGRATION TESTS: Interaction between featURES

1. **Precedence Relations**
   1. **Local variables > Variables > Events**
   2. **Transitions > Actions >** [**{ Ignore**](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p)**,** [**defer**](../../Tst/RegressionTests/Integration/Correct/Elevator/Elevator.p) **}**
2. **No raise, pop or call in functions.**
3. **Calls should always terminate with a pop.**
4. **Basic semantics tests**

**Note 1:**  For the purposes of testing P semantics, we use delay-bounding scheduling (to avoid randomness). To deal with explicit non-determinism, we use choice bounding option: -bc:1

**Note 2:** The tests should be written in such a way that **regardless of scheduling**, the (preferably single) explicit "assert" in the test case, or Zing exception, should always be reachable.

**Note 3:** We assume one thread per machine instance.

**Note 4:** To test basic semantics of P constructs, re-use static error tests from the section “P Features to test”

**Note 5:** For testing some semantical features, it is necessary to have a pair of tests for a single feature: to confirm that the feature works as expected, comparison between the results of the two tests is needed. For example, to check that atfer the “halt” event, the receiving machine M does not process any event E sent to it afterwards, there are two tests: one test demonstrates that “assert(false)” that would be enabled if the E event got processed, is not in fact enabled (and Zing “passes”); in another test – which only differs in that no “halt” event is ever sent – the “assert” is in fact enabled (and Zing “fails”).

Such paired tests have related names: XXX.p and XXX\_v.p (“v” for “validate”).

Examples: EventSentAfterSentHalt.p and EventSentAfterSentHalt\_v.p; EventSentAfterRaisedHalt.p and EventSentAfterRaisedHalt\_v.p.

* 1. **Single machine.**
     1. Entry actions
        1. “send to itself: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_3/SendInEntry.p)
        2. “send” to itself and “raise”: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_1/SendRaiseInEntry.p)
        3. “raise” and “send” to itself: [test1](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_2/RaiseSendInEntry.p)
        4. **“new”:**
        5. **“push”:**
        6. **“pop”:**
        7. **“skip”:**
        8. **“monitor”:**
        9. **Function stmt:**
        10. **“assert”:**
        11. **Remove/assign/insert:**
        12. **“return”:**
        13. **While/ite/seq:**
     2. Exit function
        1. “send to itself: [test1](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_4/SendInExitNotExecuted.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_5/SendInExitUnhandledEvent.p), [test3](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_6/SendInExitHandledEvent.p), [test4](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_7/SendInExitUnhandledHalt.p)
        2. “raise”: [tests in the “Combined Tests” section](#Combined_RaiseInExit) (error)
        3. Exit function executed upon implicit “pop”: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_15/ImplicitPopExit.p)
        4. Exit function executed upon explicit “pop”: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_16/ExplicitPopExit.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_20/BugReproIdenticalTerms.p), [test3](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_21/BugReproIdenticalTerms_workaround.p)
        5. **“new”:**
        6. **“push”:**
        7. **“pop”:**
        8. **“skip”:**
        9. **“monitor”:**
        10. **Function stmt:**
        11. **“assert”:**
        12. **Remove/assign/insert:**
        13. **“return”:**
        14. **While/ite/seq:**
     3. Transitions
        1. Goto transition: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_6/SendInExitHandledEvent.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_8/GotoToItself.p), [test3](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_14/GotoTransInheritance.p),
        2. Push transition: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_9/PushItself.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_10/Push.p),

“push” with implicit “pop”: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_11/PushImplicitPopWithSend.p) [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_17/PushImplicitPopWithRaise.p)

“push” with explicit “pop”: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_12/PushExplicitPop.p)

“push” with action inherited by the pushed state: [test1](../../Tst/RegressionTests/Integration/DynamicError/SEM_OneMachine_13/PushTransInheritance.p)

* + - 1. **“default” as a transition trigger:**
      2. **“halt” as a transition trigger:**
      3. **Error transitions: see paper, end of section 3.1 and section 5.**
    1. **“Do” declaration:**
       1. **“default” as a trigger:**
       2. **“halt” as a trgger:**
       3. “defer” as an action:[test1](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_18/UnhandledEventDeferred.p)

**event both deferred and handled in the same state (handler should override)**

**event deferred in one state, but has a handler in another state (handler should override)**

* + - 1. “ignore” as an action: [test1](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_19/UnhandledEventIgnored.p)

**event ignored in one state, but has a handler in another state**

* + - 1. **defer/ignore interaction:**

**same event both deferred and ignored**

**two events in front of the queue: 1st is ignored, 2nd is deferred**

**two events in front of the queue: 1st is deferred, 2nd is ignored**

* + - 1. **Anonymous function as an action:**
      2. **Named function as an action:**
    1. **Statements:**
       1. **“push” statement**
       2. **“raise” statement**
       3. **“send” statement**
       4. **Monitor invocation**
       5. **Function statement**
       6. **“skip”**
       7. **“assert”:**
       8. **Remove/assign/insert:**
       9. **“return”:**
       10. **While/ite/seq:**
    2. **“Special” events: “halt”, “default”.**
       1. no “unhandled exception” for unhandled “halt”: [test1](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_7/SendInExitUnhandledHalt.p)
    3. **Hot/warm/cold states:**
       1. Hot/cold states in liveness: see section [“Liveness Tests”](#LivenessTests)
       2. **Warm states**
    4. **Deadlocked state:**
       1. Queue is empty:

Deadlocked state is the only one on the stack**:** [**test1**](../../Tst/RegressionTests/Feature1SMLevelDecls/Correct/AlonBug/AlonBug.p)

There are other states on the stack:[**test1**](../../Tst/RegressionTests/Integration/Correct/SEM_OneMachine_22/AlonBug_1.p)

**All events are deferred:**

**Deadlocked state is the only one on the stack: test1**

**There are other states on the stack: test1**

* + 1. **Sending multiple identical events to the queue:**

**only one instance is kept in the queue if the payloads are the same, or if no payload:**

**multiple instances kept in yhr queue if the payloads are different:**

* + 1. **Priorities among transitions and actions:**
       1. **For an event, there’s both a handler defined in a state, and an inherited handler (local handler overrides)**
       2. **(Local) step or push transition has higher priority over actions (paper, p. 6)**
    2. **Multi-threaded runtime tests: 1 thread per machine instance**
       1. **One instance of the machine, 1 thread**
       2. **Multiple istances of the machine, multiple threads**
  1. **Two machines**
     1. **Basic features for communications between machines**
        1. **“send”**
        2. **“raise”? is it local only?**
        3. **“new”**
        4. **extend tests for one machine for stack, inheritance, etc. for the two-machines case**
     2. **Various number of instances of the two machines**
     3. **“halt” event in the two-machines context:**
        1. event “sent” after “halt” is ignored: [test1](../../Tst/RegressionTests/Integration/Correct/SEM_TwoMachines_1/EventSentAfterSentHalt.p), [test2](../../Tst/RegressionTests/Integration/DynamicError/SEM_TwoMachines_2/EventSentAfterSentHalt_v.p)
        2. event “raised” after “halt” is ignored: test1, test2
     4. **“default” event in the two-machines context**
     5. **Various protocols for communications between machines:**
        1. **Main machine and ghost machine (env): client/server protocol**

**non-det case (**use choice bounding option: -bc:1)

**deterministic case**

* + - 1. **Async mode of communication**
      2. **Sync mode of communication (PingPong sample?)**
    1. **Multi-threaded runtime tests: 1 thread per machine instance**
       1. **Single instance of each machine**
       2. **Single instance of one machine, multiple instances of another machine**
  1. **Test cases that exercise semantical errors**
     1. **Exercise all values of PRT\_STEP, PRT\_STATUS in Prt.h**
     2. **Exercise all error logging messages in PtoZing.cs (search for:** "\"<")

## Liveness tests

1. Simple sampes demonstrating liveness checking by Zing
   1. “Pass” result from Zing
      1. Actual “pass” result: [test1](../../Tst/RegressionTests/Liveness/Correct/Liveness_1/Liveness_1.p)
      2. False “pass” result: [test1](../../Tst/RegressionTests/Liveness/Correct/Liveness_1_falsePass/Liveness_1_falsePass.p)
   2. “Fail” result from Zing:
      1. Infinite loop in the “hot” state: [test1](../../Tst/RegressionTests/Liveness/DynamicError/Liveness_2/Liveness_2.p), [test2](../../Tst/RegressionTests/Liveness/DynamicError/Liveness_2_LoopMachineAdded/Liveness_2_LoopMachineAdded.p)
      2. Deadlock in the “hot” state: (bug!) [test1](../../Tst/RegressionTests/Liveness/DynamicError/Liveness_2_bugFound/Liveness_2_bugFound.p)
      3. Liveness violation but no deadlock, at least one thread is making progress: [test1](../../Tst/RegressionTests/Liveness/DynamicError/Liveness_3/Liveness_3.p)
2. **More complex liveness tests: consider adding “assert” to specify some property that would imply liveness (or its violation).**

## Combined Tests

1. Duplicates: [event defintions, machine declarations, variable declarations, transitions over the same event, actions over the same event](../../Tst/RegressionTests/Combined/StaticError/Duplicates/Duplicates.p), **multiple states with the same name**, **multiple functions with the same name**.
2. Re-definition of variables: [int and event](../../Tst/RegressionTests/Combined/Correct/variableType/variableType.p) (variable hides the name of event); **formal par hides local varialble**
3. **Missing declarations**: undefined symbol (must be declared as formal par, variable or event)
4. **“Control Impure**” tests:
   1. “Impure Control” in exit anonymous function:

PurityError(c, called) :-

c is StateDecl(\_, owner, \_, called, \_), called = AnonFunDecl(owner, \_), ControlImpure(called).

* + 1. “raise” in exit function: [test1](../../Tst/RegressionTests/Combined/StaticError/RaiseInExitFun/RaiseInExitFun.p)
    2. “push” in exit function: [test1](../../Tst/RegressionTests/Combined/StaticError/PushInExitFun/PushInExitFun.p)
    3. “pop” in exit function: [test1](../../Tst/RegressionTests/Combined/StaticError/PopInExitFun/PopInExitFun.p)
  1. **“Impure Control” in non-anonymous exit function:**

PurityError(c, n) :-

c is StateDecl(\_, owner, \_, n, \_), called is FunDecl(n, owner, \_, \_, \_, \_), ControlImpure(called).

* + 1. **“raise”**
    2. **“push”**
    3. **“pop”**
  1. **“Impure Control” in function application:**

PurityError(c, e) :-

SubSE(c, e), e = FunApp(n, \_), called is FunDecl(n, c.owner, \_, \_, \_, \_), ControlImpure(called).

* + 1. **“raise”**
    2. **“push”**
    3. **“pop”**
  1. **“Impure Control” in function statement:**

PurityError(c, e) :-

SubSE(c, e), e = FunStmt(n, \_), called is FunDecl(n, c.owner, \_, \_, \_, \_), ControlImpure(called).

* + 1. **“raise”**
    2. **“push”**
    3. **“pop”**
  1. **“Impure Control” in transition declaration, case of non-anonumous function:**

PurityError(c, n) :-

c is TransDecl(src, \_, \_, n), called is FunDecl(n, src.owner, \_, \_, \_, \_), ControlImpure(called).

* + 1. **“raise”**
    2. **“push”**
    3. **“pop”**
  1. **“Impure Control” in transition declaration, case of anonumous function:**

PurityError(c, called) :-

c is TransDecl(src, \_, \_, called), called = AnonFunDecl(src.owner, \_), ControlImpure(called).

* + 1. **“raise”**
    2. **“push”**
    3. **“pop”**