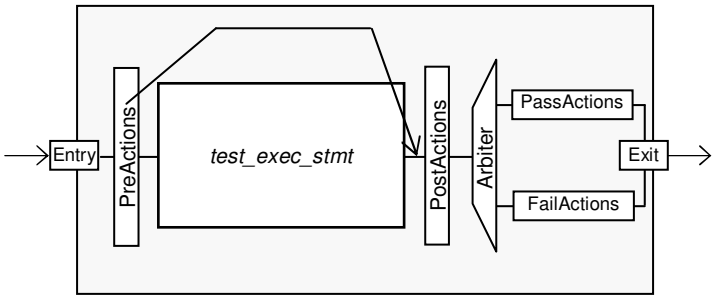
Behavior of PreActions Bypass statement for Tests/Flows

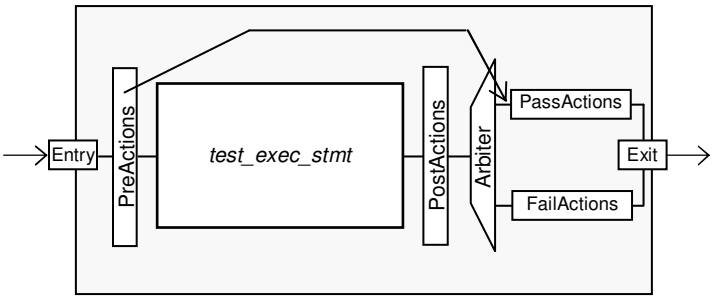
test_exec_stmt = **TestExec;** | **FuncExec** *func_stmt* | **FuncExec** { (*func_stmt*)* } | *flownode_stmt*+
bypass_stmt = **Bypass** (**GoTo <Pass** | **Fail** | VAR_NAME **>** (**SkipActions**));

1. Bypass; Skip test, execute PostActions and either PassActions or FailActions, depending on state of execResult for test or flow. (execResult can be forced to a specific state prior to Bypass statement.



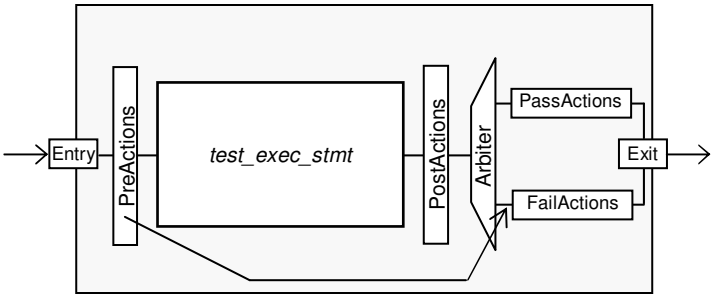

```
PreActions {
      <pre_actions prior to Bypass>
      Bypass;
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

2. Bypass GoTo Pass; Skip test, skip PostActions, and execute PassActions.



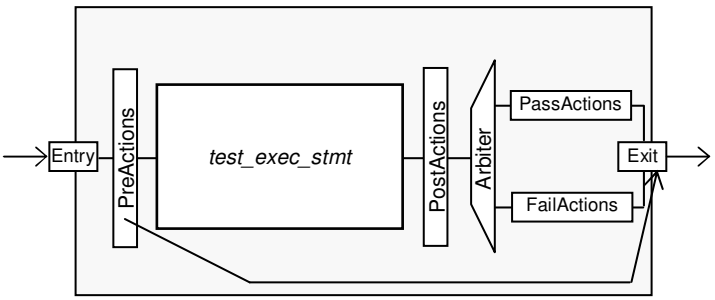

```
PreActions {
      <pre_actions prior to Bypass>
      Bypass GoTo Pass; | Bypass GoTo <var_name>;
      // <var_name> equates to Pass,
      // either as a string ("[pP][aA][sS][sS]")
      // or an integer (0 = Pass, non-zero = Fail)
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

3. Bypass GoTo Fail; Skip test, skip PostActions, and execute FailActions.




```
PreActions {
      <pre_actions prior to Bypass>
      Bypass GoTo Fail; | Bypass GoTo <var_name>;
      // <var_name> equates to Fail,
      // either as a string ("[fF][aA][iI][lL]")
      // or an integer (0 = Pass, non-zero = Fail)
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

4. Bypass GoTo <Pass | Fail | <*var_name*> > SkipActions; Skip test, skip PostActions, skip PassActions or FailActions and return.

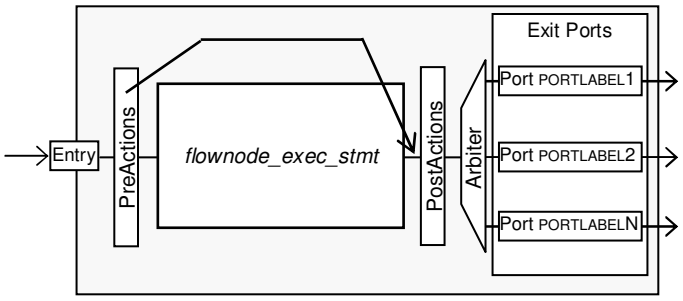



```
PreActions {
      <pre_actions prior to Bypass>
      Bypass GoTo <Pass | Fail> SkipActions; |
      Bypass GoTo <var_name> SkipActions;
      // <var_name> equates to Pass or Fail,
      // either as a string ("[pP][aA][sS][sS]" or "[fF][aA][iI][lL]")
      // or an integer (0 = Pass, non-zero = Fail)
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

Behavior of PreActions Bypass statement for FlowNodes

flownode_exec_stmt = **TestExec** *execute_stmt*
bypass_stmt = **Bypass** (**GoTo** <PORT_LABEL | VAR_NAME > (**SkipActions**));

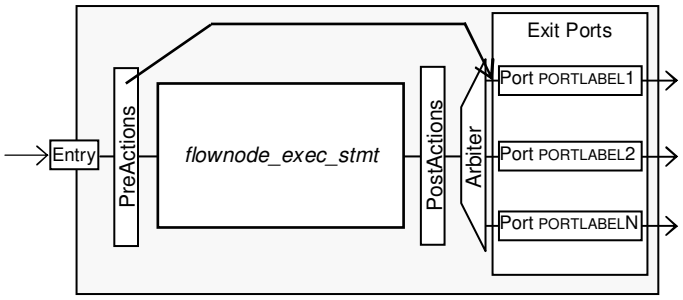
1. Bypass; Skip test, execute PostActions and ExitPorts Actions as determined by Arbiter. Variables and values used to control exit port arbitration can be set to a specific state prior to Bypass statement.



FlowNode

```
PreActions {
      <pre_actions prior to Bypass>
      Bypass;
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

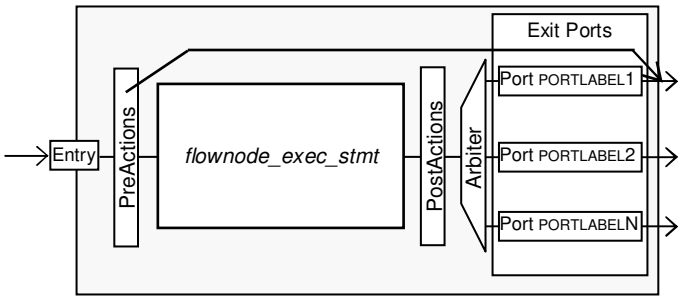
2. Bypass GoTo <PORT_LABEL | VAR_NAME>; Skip test, skip PostActions, execute ExitPorts Actions for ExitPort specified by PORT_LABEL or VAR_NAME. PORT_LABEL is a string identifying the desired exit port. If an empty string, no bypass action occurs. VAR_NAME is either a string variable or an integer variable. If a string variable, it specifies the bypass exit port by label, unless it's an empty string, in which case no bypass action occurs. If an integer variable, it specifies the bypass exit port by index (based on the ordinal order of the exit ports, from top of list to bottom, with the first index being 0).



FlowNode

```
PreActions {
      <pre_actions prior to Bypass>
      Bypass GoTo PORTLABEL1;
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```

3. Bypass GoTo <PORT_LABEL | VAR_NAME> SkipActions; Skip test, skip PostActions, skip ExitPorts Actions, and return via the exit port specified by PORT_LABEL or VAR_NAME. PORT_LABEL is a string identifying the desired exit port. If an empty string, no bypass action occurs. VAR_NAME is either a string variable or an integer variable. If a string variable, it specifies the bypass exit port by label, unless it's an empty string, in which case no bypass action occurs. If an integer variable, it specifies the bypass exit port by index (based on the ordinal order of the exit ports, from top of list to bottom, with the first index being 0).



FlowNode

```
PreActions {
      <pre_actions prior to Bypass>
      Bypass GoTo PORTLABEL1 SkipActions;
      <pre_actions following Bypass>  (NOT EXECUTED)
}
```