## 20.2 The Alt statement

An alt statement expresses sets of possible alternatives that form a tree of possible execution paths.

***Syntactical Structure***

**alt** "{"

{

"[" [ *BooleanExpression* ] "]"

( ( *TimeoutStatement* |

*ReceiveStatement* |

*TriggerStatement* |

*GetCallStatement* |

*CatchStatement* |

*CheckStatement* |

*GetReplyStatement* |

*DoneStatement* |

*KilledStatement* ) *StatementBlock* )

|

( *AltstepInstance* [ *StatementBlock* ] )

}

[ "[" **else** "]" *StatementBlock* ]

"}"

***Semantic Description***

The **alt** statement denotes branching of test behaviour due to the reception and handling of communication and/or timer events and/or the termination of parallel test components, i.e. it is related to the use of the TTCN‑3 operations **receive**, **trigger**, **getcall**, **getreply**, **catch**, **check**, **timeout, done** and **killed**. The **alt** statement denotes a set of possible events that are to be matched against a particular snapshot.

**Execution of alternative behaviour:**

When entering an **alt** statement, a snapshot is taken.

The alternative branches in the **alt** statement and the top alternatives of invoked altsteps and altsteps that are activated as defaults are processed in the order of their appearance. If several defaults are active, the reverse order of their activation determines the evaluation order of the top alternatives in the defaults. The alternative branches in active defaults are reached by the default mechanism described in clause 20.5.

The individual alternative branches are either branches that may be guarded by a Boolean expression or else-branches, i.e. alternative branches starting with [**else**].

Else-branches are always chosen and executed when they are reached (see below).

Branches that may be guarded by a Boolean expressions either invoke an altstep (*altstep-branch*), or start with a **done** operation (*done-branch*), a **killed** operation (*killed-branch*), **timeout** operation (*timeout-branch*) or a receiving operation (*receiving-branch*), i.e. **receive**, **trigger**, **getcall**, **getreply,** **catch** or a **check** operation. The evaluation of the Boolean guards shall be based on the snapshot. The Boolean guard is considered to be *fulfilled* if no Boolean guard is defined, or if the Boolean guard evaluates to **true**. The branches are processed and executed in the following manner.

An *altstep-branch* is selected if the Boolean guard is fulfilled. The selection of an *altstep-branch* causes the invocation of the referenced altstep, i.e. the altstep is invoked and the evaluation of the snapshot continues within the altstep. Altstep-branches may contain an optional statement block. The optional statement block shall be executed only, if an alternative of the altstep referenced in the altstep-branch has been selected and executed.

A *done-branch* is selected if the Boolean guard is fulfilled and if the specified test component is in the list of stopped components of the snapshot. The selection causes the execution of the statement block following the **done** operation. The **done** operation itself has no further effect.

A *killed-branch* is selected if the Boolean guard is fulfilled and if the specified test component is in the list of killed components of the snapshot. The selection causes the execution of the statement block following the **killed** operation. The **killed** operation itself has no further effect.

A *timeout-branch* is selected if the Boolean guard is fulfilled and if the specified timeout event is in the timeout-list of the snapshot. The selection causes execution of the specified **timeout** operation, i.e. removal of the timeout event from the timeout-list, and the execution of the statement block following the **timeout** operation.

A *receiving-branch* is selected if the Boolean guard is fulfilled and if the matching criteria of receiving operation is fulfilled by one of the messages, calls, replies or exceptions in the snapshot. The selection causes execution of the receiving operation, i.e. removal of the matching message, call, reply or exception from the port queue, maybe an assignment of the received information to a variable and the execution of the statement block following the receiving operation. In the case of the **trigger** operation the top message of the queue is also removed if the Boolean guard is fulfilled but the matching criteria is not. In this case the statement block of the given alternative is not executed.

NOTE 1: The TTCN‑3 semantics describe the evaluation of a snapshot as a series of indivisible actions of a test component. The semantics do not assume that the evaluation of a snapshot has no duration. During the evaluation of a snapshot, test components may stop, timers may timeout and new messages, calls, replies or exceptions may enter the port queues of the component However, these events do not change the actual snapshot and thus, are not considered for the snapshot evaluation.

NOTE 2: Due to the possibility of defining dynamic test configurations, a receiving branch may refer to a disconnected or unmapped port at the time of its evaluation. In TTCN-3, ports belong to the receiving component and matching is related to the top elements in the port queues. Dynamically unmapped and disconnected ports contribute to a snapshot in the same manner as mapped and connected ports. This means, the execution of receiving operations may empty the queues of unmapped and disconnected ports without causing a test case error.

If none of the alternative branches in the **alt** statement and top alternatives in the invoked altsteps and active defaults can be selected and executed, the **alt** statement shall be executed again, i.e. a new snapshot is taken and the evaluation of the alternative branches is repeated with the new snapshot. This repetitive procedure shall continue until either an alternative branch is selected and executed, or the test case is stopped by another component or by the test system (e.g. because the MTC is stopped) or with a dynamic error.

The test case shall stop and indicate a dynamic error if a test component is completely blocked. This means none of the alternatives can be chosen, no relevant test component is running, no relevant timer is running and all relevant ports contain at least one message, call, reply or exception that do not match.

NOTE 3: The repetitive procedure of taking a complete snapshot and re-evaluate all alternatives is only a conceptual means for describing the semantics of the **alt** statement. The concrete algorithm that implements this semantics is outside the scope of the present document.

**Selecting/deselecting an alternative:**

If necessary, it is possible to enable/disable an alternative by means of a Boolean expression placed between the ("[…]") brackets of the alternative.

**Else branch in alternatives:**

Any branch in an **alt** statement can be defined as an else branch by including the **else** keyword between the opening and closing brackets at the beginning of the alternative. The statement block of the else branch is always executed if no other alternative textually preceding the else branch has proceeded.

**Default mechanism:**

It should be noted that the default mechanism (see clause 20.5) is always invoked at the end of all alternatives. If an **else** branch is defined, the default mechanism will never be called, i.e. active defaults will never be entered.

NOTE 4: It is also possible to use **else** in altsteps.

NOTE 5: It is allowed to use a **repeat** statement within an **else** branch.

NOTE 6: It is allowed to define more than one else branch in an alt statement or in an altstep, however always only the first else branch is executed.

**Re-evaluation of alt statements:**

The re-evaluation of an **alt** statement can be specified by using a **repeat** statement (see clause 20.3).

**Invocation of altsteps as alternatives:**

TTCN‑3 allows the invocation of altsteps as alternatives in **alt** statements (see clause 16.2.1). When an altstep is explicitly invoked as an alternative, the optional statement block following the altstep call shall also be executed.

**Continue execution after the alt statement:**

Behaviour execution continues with the statement following the **alt** statement when one of the branches of the **alt** or invoked defaults is selected and completely executed, or a branch of an **altstep** used in an altsteps-branch is selected and the branch and the optional statement block following the invoked altstep are completely executed.

Execution also continues with the statement following the **alt** statement if a **break** statement is reached in the statement block of the selected branch of an **alt** statement, of an **altstep** used in an altstep-branch, or of an **altstep** invoked as default.

The **alt** statement can also be left by using a **goto** statement in the selected branch of the **alt** (i.e. no branches of altsteps and defaults can be considered in this case), and execution continues with the statement following the label, **goto** is pointing to.

***Restrictions***

In addition to the general static rules of TTCN‑3 given in clause 5 and shown in table 15, the following restrictions apply:

1. The open and close square brackets ("[…]") shall be present at the start of each alternative, even if they are empty. This not only aids readability but also is necessary to syntactically distinguish one alternative from another.
2. The evaluation of a Boolean expression guarding an alternative shall not have side effects. To avoid side effects that cause an inconsistency between the actual snapshot and the state of the component, the same restrictions as the restrictions for the initialization of local definitions within altsteps (clause 16.2) and the restrictions imposed on the contents of functions called from special places (clause 16.1.4) shall apply.
3. The evaluation of the event of an alt branch shall not have side effects. To avoid side effects that cause an inconsistency between the actual snapshot and the state of the component or introduce indeterminism in the evaluation of the following alt branches or the re-evaluation of the same alt branch, the restrictions imposed on the contents of functions called from special places (clause 16.1.4) shall apply to expressions occurring in the matching part of an alternative..
4. The evaluation of an altstep invoked from an alt branch, if none of the alternatives in the altstep is chosen, shall not have side effects. To avoid side effects the restrictions imposed on the contents of functions called from special places (clause 16.1.4) shall apply to the actual parameters of the invoked altstep.
5. The else branch shall not contain any of the actions allowed in branches guarded by a boolean expression (i.e. an **altstep** call or a **done**, a **killed**, a **timeout** or a receiving operation).
6. An **alt** statement used within the module control part shall only contain **timeout** statements.

***Examples***

EXAMPLE 1: Nested alternatives

**alt** {

[] MyPort.**receive** (MyMessage) {

**setverdict** (**pass**);

MyTimer.**start**;

**alt** {

[] MyPort.**receive** (MySecondMessage) {

MyTimer.**stop**;

**setverdict** (**pass**);

}

[] MyTimer.**timeout** {

MyPort.**send** (MyRepeat);

MyTimer.**start**;

a**l**t {

[] MyPort.**receive** (MySecondMessage) {

MyTimer.**stop**;

**setverdict** (**pass**)

}

[] MyTimer.**timeout** { **setverdict** (**inconc**) }

[] MyPort.**receive** { **setverdict** (**fail**) }

}

}

[] MyPort.**receive** { **setverdict** (**fail**) }

}

}

[] MyTimer.**timeout** { **setverdict** (**inconc**) }

[] MyPort.**receive** { **setverdict** (**fail**) }

}

EXAMPLE 2: Alt statement with guards

**alt** {

[x>1] L2.**receive** { // Boolean guard/expression

**setverdict**(**pass**);

}

[x<=1] L2.**receive** { // Boolean guard/expression

**setverdict**(**inconc**);

}

}

EXAMPLE 3: Alt statement with else branch

// Use of alternative with Boolean expressions (or guard) and else branch

**alt** {

:

[**else**] { // else branch

MyErrorHandling();

**setverdict**(**fail**);

**stop;**

}

}

EXAMPLE 4: Re-evaluation with repeat

**alt** {

[] PCO3.**receive** {

count := count + 1;

**repeat** // usage of repeat

}

[] T1.**timeout** { }

[] **any** **port**.**receive** {

**setverdict**(**fail**);

**stop**;

}

}

EXAMPLE 5: Alt statement with explicitly invoked altstep

**alt** {

[] PCO3.**receive** { }

[] AnotherAltStep() { // Explicit call of altstep AnotherAltStep as alternative.

**setverdict**(**inconc**) // Statement block executed if an alternative within

// altstep AnotherAltStep has been selected and executed.

}

[] MyTimer.**timeout** { }

}

EXAMPLE 6: Alt statement with forbidden function calls

**alt** {

[] getPort().**receive**(t(p())) { } // forbidden if getPort, t or p has side effects

[] AnotherAltStep(f()); // forbidden if f has side effects

[] MyTimer[i(p())].**timeout** { } // forbidden if i or p has side effects

[g()] getComponent(p()).**done** {} // forbidden if g, getComponent or p has side effects

}