## 20.4 The Interleave statement

The **interleave** statement allows to specify the interleaved occurrence and handling of receiving events including **done**, **killed**, **timeout**, **receive**, **trigger**, **getcall**, **getreply**, **catch** and **check.**

***Syntactical Structure***

**interleave** "{"

{ "[]" ( *TimeoutStatement* |

*ReceiveStatement* |

*TriggerStatement* |

*GetCallStatement* |

*CatchStatement* |

*CheckStatement* |

*GetReplyStatement* |

*DoneStatement* |

*KilledStatement* ) *StatementBlock*

}

"}"

***Semantic Description***

The **interleave** statement allows to specify the interleaved occurrence and handling of the statements **done**, **killed**, **timeout**, **receive**, **trigger**, **getcall**, **getreply**, **catch** and **check**.

Interleaved behaviour can always be replaced by an equivalent set of nested **alt** statements. The procedures for this replacement and the operational semantics of interleaving are described in part 4 of the TTCN‑3 standard (ETSI ES 201 873‑4 [1]).

The rules for the evaluation of an interleaving statement are the following:

1. Whenever a reception statement is executed, the following non-reception statements are subsequently executed until the next reception statement is reached, a **break** statement is reached, or the interleaved sequence ends.

NOTE 1: Reception statements are TTCN‑3 statements which may occur in sets of alternatives, i.e. **receive**, **check**, **trigger**, **getcall**, **getreply**, **catch**, **done,** **killed** and **timeout**. Non-reception statements denote all other non-control-transfer statements which can be used within the **interleave** statement.

1. If none of the alternatives of the **interleave** statement can be executed, the default mechanism will be invoked. This means, according to the semantics of the default mechanism, the actual snapshot will be used to evaluate those altsteps that have been activated before entering the **interleave** statement.

NOTE 2: The complete semantics of the default mechanism within an **interleave** statement is given by replacing the **interleave** statement by an equivalent set of nested **alt** statements. The default mechanism applies for each of these **alt** statements.

1. The evaluation then continues by taking the next snapshot if no **break** statement was encountered.
2. The evaluation of the **interleave** statement is terminated if a **break** statement is executed.

The operational semantics of interleaving are fully defined in part 4 of the TTCN‑3 standard (ETSI ES 201 873‑4 [1]).

***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. Control transfer statements **activate**, **deactivate**, **repeat**, all calls of altsteps and (direct and indirect) calls of user-defined functions, which include reception statements, shall not be used in **interleave** statements.
2. In addition, it is not allowed to guard branches of an **interleave** statement with Boolean expressions (i.e. the '[ ]' shall always be empty). It is also not allowed to specify **else** branches in interleaved behaviour.
3. An **interleave** statement used within the module control part shall only contain **timeout** statements.
4. The restricted use of the control transfer statements **for**, **while**, **do-while**, and **goto** within **interleave** statements is allowed under the following conditions:
   1. The loop statements **for**, **while**, and **do-while** can be used within statements blocks that do not contain reception statements.
   2. The **goto** statement can be used for defining unconditional jumps within statements blocks that do not contain reception statements and for specifying unconditional jumps out of **interleave** statements.

***Examples***

// The following TTCN‑3 code fragment

**interleave** {

[] pCO1.**receive**(mw\_mySig1) {

PCO1.**send**(m\_mySig2);

PCO1.**receive**(mw\_mySig3);

}

[] pCO2.**receive**(mw\_mySig4) {

pCO2.**send**(m\_mySig5);

pCO2.**send**(m\_mySig6);

pCO2.**receive**(mw\_mySig7);

}

}

// is a shorthand for

**alt** {

[] PCO1.**receive**(mw\_mySig1) {

PCO1.**send**(m\_mySig2);

**alt** {

[] PCO1.**receive**(mw\_mySig3) {

alt {

[] PCO2.**receive**(mw\_mySig4) {

PCO2.**send**(m\_mySig5);

PCO2.**send**(m\_mySig6);

PCO2.**receive**(mw\_mySig7)

}

}

}

[] PCO2.**receive**(mw\_mySig4) {

PCO2.**send**(m\_mySig5);

PCO2.**send**(m\_mySig6);

**alt** {

[] PCO1.**receive**(mw\_mySig3) {

PCO2.**receive**(mw\_mySig7);

}

[] PCO2.**receive**(mw\_mySig7) {

PCO1.**receive**(mw\_mySig3);

}

}

}

}

}

[] pCO2.**receive**(mw\_mySig4) {

pCO2.**send**(m\_mySig5);

pCO2.**send**(m\_mySig6);

**alt** {

[] pCO1.**receive**(mw\_mySig1) {

pCO1.**send**(m\_mySig2);

**alt** {

[] pCO1.**receive**(mw\_mySig3) {

pCO2.**receive**(mw\_mySig7);

}

[] pCO2.**receive**(mw\_mySig7) {

pCO1.**receive**(mw\_mySig3);

}

}

}

[] pCO2.**receive**(mw\_mySig7) {

**alt** {

[] pCO1.**receive**(mw\_mySig1) {

pCO1.**send**(m\_mySig2);

pCO1.**receive**(mw\_mySig3);

}

}

}

}

}

}