## 15.11 Concatenating templates of string and list types

Templates of string and list types (bitstring, octetstring, hexstring, charstring, universal charstring, record of, set of, and array) can be concatenated from several single (in-line) templates using the concatenation operation. With the exception of charstring and universal charstring templates, each single template shall have the same root type.

The single templates of binary string types shall evaluate only to the matching mechanisms specific value, combined template, *AnyValue* without a length modifier, *AnyValue* or *AnyValueOrNone*, both constrained to a fixed length.

The concatenation of templates of binary string types results in the sequential concatenation of the single templates from left to right, with two exceptions: matching symbol *AnyValue* without a length modifier shall be replaced by a single *AnyElementsOrNone* matching symbol before concatenation and matching symbols *AnyValue* and *AnyValueOrNone* that are each constrained to a fixed length N shall be replaced by N *AnyElement* matching symbols before concatenation.

The single templates of list types shall evaluate only to the matching mechanisms specific value, combined template, *AnyValue* with or without a length modifier, *AnyValueOrNone* with a length modifier.

The concatenation of templates of list types results in the sequential concatenation of the single templates from left to right, with two exceptions: before concatenation, an *AnyValue* matching symbol without a length modifier shall be replaced by a single *AnyElementsOrNone* matching symbol and *AnyValue* and *AnyValueOrNone* matching symbols with a length modifier shall be replaced by an *AnyElementsOrNone* matching mechanism with the same length modifier.

Single templates of charstring and universal charstring types shall evaluate only to specific values, *AnyValue* with or without a length modifier, *AnyValueOrNone* with a length modifier or pattern. When concatenating templates of charstring and universal charstring types, each single template shall be either of the charstring or universal charstring type. When templates of charstring and universal charstring type are both present in the concatenation, the charstring values are implicitly converted to universal charstring values according to the rules specified in clause 6.3.1 before concatenation and the resulting template is of the universal charstring type.

The concatenation of templates of character string types results in the sequential concatenation of the single templates from left to right. In case all operands are specific values, the result of concatenation will be a specific value. In all other cases, if possible, a pattern will be produced. When producing the pattern, the templates are first transformed to a character string according to table 14. In the next step, the strings are concatenated and a pattern is created from the concatenation result. If one of the single templates is a pattern with the @nocase modifier, then all other single templates shall also be patterns with the @nocase modifier and the resulting pattern will also have the **@nocase** modifier.

Table 14: Transformation of character string templates before concatenation

| Concatenation operand | Transformed string |
| --- | --- |
| Specific values (character strings) | The character string is escaped by inserting the "\" character before each of the following characters: "#", "(", ")", "\*", "+", "-", "?", "[", "\", "]", "^", "{", "|", "}". |
| ?, ? length(0..infinity) or \* length(0..infinity) | \* |
| ? length(0) or \* length(0) | empty string |
| ? length(1) or \* length(1) | "?" |
| ? length(1, infinity) or \* length(1, infinity) | "?+" |
| ? length(*n*) or \* length(*n*) | "?#(*n)*" |
| ? length(*n*, infinity) or \* length(n, infinity) | "?#(*n*,)" |
| ? length(*n*, *m*) or \* length(*n*, *m*) | "?#(*n*,*m*)" |
| pattern "*content*" | "*content*" |

Concatenation of character strings in a pattern definition is described in clause B.1.5. Concatenation of character strings in a pattern definition always takes precedence over concatenation of templates. If it is necessary to concatenate a pattern definition with a following character string according to the template concatenation rules (thus automatically escaping the string), the pattern definition shall be enclosed into parentheses.

The concatenation shall be performed completely before using the resulting template (e.g. for assignment or matching) and the result shall be type-compatible with the place of its use.

The length matching attribute shall not follow a template or template field produced by concatenation directly, but in this case the concatenation shall be placed within a pair of parentheses.

***Restrictions***

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

a) All operands of the concatenation operation shall be at least partially initialized.

EXAMPLE 1: Composing templates of binary string types

**template bitstring** mw\_mybit := '010'B & ? & '1'B & ? **length**(1) & '1'B;

// results in the template '010\*1?1'B

// note that & ? & turns to \* within the resulting bitstring as the original ?

// stands for a bitstring of any length

**template bitstring** mw\_mybit2 := '010'B & \* **length**(2);

// results in the template '010??'B

// note that the ability of the AnyValueOrNone matching symbol to match omitted

// values is lost during concatenation

**type bitstring** ConstrainedBitString **length**(3);

// values of this type are always exactly 3 bits long

**template** ConstrainedBitString mw\_mybit3 := ?;

**template bitstring** mw\_mybit4 := '010'B & mw\_mybit3;

// results in the template '010\*'B

// note that the type constraint of the mw\_mybit3 template has no impact on the

// concatenation operation; the AnyValue symbol without a length modifier is always

// replaced with a single AnyElementsOrNone

**template octetstring** mw\_myoct1 := 'ABCD'O & 'EF'O & ? & ? **length**(1) & 'EF'O;

// results in the template 'ABCDEF\*?EF'O

// note that & ? & turns to \* within the resulting octetstring as the original ?

// stands for an octetstring of any length

**template octetstring** mw\_myoct2 := 'ABCD'O & ? **length** (2) & 'EF'O;

// results in the template 'ABCD??EF'O

// (i.e. a 5 octets i.e. 10 hexadecimal digits long value)

**template octetstring** mw\_myoctWrong := 'ABCD'O & ? **length**(2) **length** (4);

// causes an error, no length matching attribute shall directly follow a concatenation

**template octetstring** mw\_myoct3 := ('ABCD'O & ? **length**(2)) **length** (4);

// results in the template 'ABCD??'O

**template hexstring** mw\_myhexPar (**integer** N):=

'ABC'H & ? **length**(N) & 'E'H & ? **length**(1) & 'F'H;

**function** f\_myFunc() **runs on** MyCompType {

**var integer** v\_int := 3;

**var template hexstring** v\_hstring;

:

v\_hstring := 'ABC'H & ? **length**(v\_int) & 'E'H & ? **length**(1) & 'F'H;

//results in the template 'ABC???E?F'H

p.**receive** (mw\_myhexPar(4));

//actual content of mw\_myhexPar is 'ABC????E?F'H

}