.4 Case Statements

(1)

A case_statement selects for execution one of a number of alternative sequences_of_statements; the chosen alternative is defined by the value of an expression.

Syntax

(2)

```
case_statement ::=
   case expression is
       case_statement_alternative
      {case_statement_alternative}
   end case;
```

(3)

```
case_statement_alternative ::=
   when discrete_choice_list =>
      sequence_of_statements
```

Name Resolution Rules

(4)

The expression is expected to be of any discrete type. The expected type for each discrete_choice is the type of the expression.

Legality Rules

(5)

The expressions and discrete_ranges given as discrete_choices of a case_statement shall be static. A discrete_choice others, if present, shall appear alone and in the last discrete_choice_list.

(6)

The possible values of the expression shall be covered as follows:

(7)

- If the expression is a name (including a type_conversion or a function_call) having a static and constrained nominal subtype, or is a qualified_expression whose subtype_mark denotes a static and constrained scalar subtype, then each non-others discrete_choice shall cover only values in that subtype, and each value of that subtype shall be covered by some discrete_choice (either explicitly or by others).

(8)

- If the type of the expression is root_integer, universal_integer, or a descendant of a formal scalar type, then the case_statement shall have an others discrete_choice.

(9)

- Otherwise, each value of the base range of the type of the expression shall be covered (either explicitly or by others).

(10)
Two distinct discrete_choices of a case_statement shall not cover the same value.

Dynamic Semantics

(11)
For the execution of a case_statement the expression is first evaluated.

(12)
If the value of the expression is covered by the discrete_choice_list of some case_statement_alternative, then the sequence_of_statements of the _alternative is executed.

(13)
Otherwise (the value is not covered by any discrete_choice_list, perhaps due to being outside the base range), Constraint_Error is raised.

NOTES

(14)
(5) The execution of a case_statement chooses one and only one alternative. Qualification of the expression of a case_statement by a static subtype can often be used to limit the number of choices that need be given explicitly.

Examples

(15)
Examples of case statements:

(16)
```
case Sensor is
   when Elevation  => Record_Elevation(Sensor_Value);
   when Azimuth    => Record_Azimuth  (Sensor_Value);
   when Distance   => Record_Distance (Sensor_Value);
   when others     => null;
end case;
```

(17)
```
case Today is
   when Mon        => Compute_Initial_Balance;
   when Fri        => Compute_Closing_Balance;
   when Tue .. Thu => Generate_Report(Today);
   when Sat .. Sun => null;
end case;
```

(18)
```
case Bin_Number(Count) is
   when 1      => Update_Bin(1);
   when 2      => Update_Bin(2);
   when 3 | 4  =>
      Empty_Bin(1);
      Empty_Bin(2);
   when others => raise Error;
end case;
```