Section 7
General Rules for Arithmetic Models

This chapter defines the general rules for arithmetic models.

7.1 Principles of arithmetic models

The purpose of arithmetic models is to specify calculatable mathematical relationships between objects representing physical quantities in the library. Arithmetic models are identified by context-sensitive keywords, because the way how these quantities are measured, extracted or interpreted depends on the context in which the objects are placed.

The quantity identified by the keyword CAPACITANCE may serve as example. In the context of a PIN, it represents pin capacitance. In the context of a WIRE, it represents wire capacitance. In the context of a RULE, it represents the calculation method for a capacitance formed by a layout pattern described within the rule. The context-specific semantic of each arithmetic model are specified in dedicated chapters.

In certain cases, the context alone does not completely specify the semantics of an arithmetic model. Auxiliary definitions within the arithmetic model are needed, represented in the form of annotations or annotation containers.

A simple example is the UNIT annotation, which is applicable for most arithmetic models. It specifies the unit in terms of which the arithmetic model data is represented. The applicable auxiliary objects for each arithmetic model are specified in dedicated chapters.

7.1.1 Global definitions for arithmetic models

In many cases, auxiliary definitions apply globally to all arithmetic models within a certain context, for instance, the UNIT may apply for all CAPACITANCE objects within a library. In order to specify such global definitions, the arithmetic model construct may be used without data.

```
model_definition ::=
   model_keyword [ identifier ] { all_purpose_items }
```

This construct has the syntactical form of an `annotation_container` (see chapter ??).

7.1.2 Trivial Arithmetic Model

The simplest form of an arithmetic model contains just constant data.

```
trivial_model ::=
   model_keyword [ identifier ] = number ;
|  model_keyword [ identifier ] = number { all_purpose_items }
```

This construct has the syntactical form of an `annotation` (see chapter ??).

7.1.3 Arithmetic Model with EQUATION

The arithmetic model data may be represented as an EQUATION. In this case, a HEADER defines the arguments of the equation. It is also possible to use other arithmetic models, which are visible within the context of this arithmetic model, as arguments. Those arguments need not appear in the HEADER.

```
equation_based_model ::=
   model_keyword [ identifier ] {
      [ all_purpose_items ]
      [ equation_based_header ]
      equation
   }

equation_based_header ::=
   HEADER { model_keyword { model_keyword } }
|  HEADER { model_definition { model_definition } }

equation ::=
   EQUATION { arithmetic_expression }
```

The syntax of `arithmetic_expression` is explained in chapter 5.2.

7.1.4 Arithmetic Model with TABLE

The arithmetic model data may be represented as a lookup table. In this case, a TABLE is necessary for the data itself and for each argument.

```
table_based_model ::=
   model_keyword [ identifier ] {
      [ all_purpose_items ]
      table_based_header
      table
      [ equation ]
   }

table_based_header ::=
   HEADER { table_model_definition { table_model_definition } }

table_model_definition ::=
   model_keyword [ identifier ] { all_purpose_items table }

table ::=
   TABLE { symbol { symbol } }
|  TABLE { number { number } }
```

Tables containing symbols are only for lookup of discrete datapoints. Tables containing numbers are for calculation and eventually interpolation of datapoints. The *model*_keyword (see dedicated chapters) defines whether symbols or numbers are legal for a particular table.

The size of the table inside the `table_based_model` must be the product of the size of the tables inside the `table_header`. In order to support interpolation, the numbers in each table inside the `table_header` must be in strictly monotonic ascending order. Details are explained in chapter 5.3.

The table_model_definition can also be used outside the context of a table_header, very much like a model_definition. In this case, the model_definition supplies the same information as the table_model_definition, plus the additional information of a discrete set of valid numbers applicable for the model.

For example, the WIDTH of a physical layout object may have only a discrete set of legal values. Those can be specified using a table_model_definition.

However, the table in a table_model_definition *outside* a table_header shall not substitute the table *inside* the table_header. The former is for definition of a legal set of values, the latter is for definition of the table-lookup indices.

If all table data are numbers, the table_based_model may also have an optional equation. The equation is to be used when the argument data are out of interpolation range. Without the equation, extrapolation shall be applied for data out of range.

7.1.5 Complex Arithmetic Model

A complex arithmetic model can be constructed by defining a nested arithmetic model within another arithmetic model. The data of the inner arithmetic model is calculated first. Then the result is applied for calculation of the data of the outer arithmetic model.

```
complex_model ::=
   model_keyword [ identifier ] {
      [ all_purpose_items ]
      HEADER { model { model } }
      equation
   }
|  model_keyword {
      all_purpose_items
      HEADER { header_model { header_model } }
      table
      [ equation ]
   }

header_model ::=
   model_definition
|  table_model_definition
|  equation_based_model
|  table_based_model
|  header_table_model

header_table_model ::=
   model_keyword [ identifier ] {
      all_purpose_items
      HEADER { symbol { symbol } }
      TABLE { number { number } }
   }
```

If any header_model is either model_definition or table_model_definition, then the complex_model reduces to the previously defined equation_based_model and table_based_model, respectively. In order to support a table in the general_model, any

header_model must be either table_model_definition or table_based_model, and the numbers in each table inside each header_model must be strictly monotonically increasing.

The header_table_model construct allows to associate symbols with numbers. For example, process corners may be defined as discrete symbols, and they may be associated with process derating factors. The numbers can be used in equations and for interpolation, whereas the symbols cannot.

7.1.6 Containers for arithmetic models and submodels

Containers for arithmetic models are for the purpose of supplementing the context-specific semantics of the arithmetic model. Therefore arithmetic models may be placed in the context of arithmetic model containers, using the following construct.

```
model_container ::=
   model_container_keyword  {
      [ all_purpose_items ]
      model_container_contents { model_container_contents }
   }

model_container_contents ::=
   model_container
|  trivial_model
|  complex_model
```

There is a dedicated set of *model_container*_keywords. In addition, *model*_keywords can also be used as *model_container*_keywords, and dedicated *submodel*_keywords can be used as *model*_keywords. The number of levels in nested arithmetic model containers is restricted by the set of allowed combinations between *model_container*_keywords, *model*_keywords and *submodel*_keywords. This is specified in chapter 7.5.

7.2 Arithmetic expressions

Arithmetic expressions define the contents of an EQUATION. Variables used in the EQUATION are the identifiers of the header_model, if present, or else the *model*_keywords of the header_model.

7.2.1 Syntax of arithmetic expressions

The syntax of arithmetic expressions shall be defined as follows:

```
arithmetic_expression ::=
     ( arithmetic_expression )
   | number
   | [ arithmetic_unary ] identifier
   | arithmetic_expression arithmetic_binary arithmetic_expression
   | arithmetic_function_operator
          ( arithmetic_expression { , arithmetic_expression } )
   | boolean_expression ? arithmetic_expression :
          { boolean_expression ? arithmetic_expression : }
          arithmetic_expression
```

Examples:

```
1.24
- Vdd
C1 + C2
MAX ( 3.5*C , -Vdd/2 , 0.0 )
(C > 10) ? Vdd**2 : 1/2*Vdd - 0.5*C
```

7.2.2 Arithmetic operators

Table 7-1, Table 7-2, and Table 7-3 list unary, binary and function arithmetic operators.

Table 7-1 : Unary arithmetic operators

Operator	Description
+	positive sign (for integer or number)
-	negative sign (for integer or number)

Table 7-2 : Binary arithmetic operators

Operator	Description
+	addition (integer or number)
-	subtraction (integer or number)
*****	multiplication (integer or number)
/	division (integer or number)
******	exponentiation (integer or number)
%	modulo division (integer or number)

Table 7-3 : Function arithmetic operators

Operator	Description
LOG	natural logarithm (argument is + integer or number)
EXP	natural exponential (argument is integer or number)
ABS	absolute value (argument is integer or number)
MIN	minimum (all arguments are integer or number)
MAX	maximum (all arguments are integer or number)

Function operators with one argument (such as `log`, `exp` and `abs`) or multiple arguments (such as `min` and `max`) must have the arguments within parenthesis, e.g. `min(1.2,-4.3,0.8)`.

7.2.3 Operator priorities

The priority of binding operators to operands in arithmetic expressions shall be from strongest to weakest in the following order:

1. unary arithmetic operator (+, -)
2. exponentiation (**)
3. multiplication (*), division (/), modulo division (%)
4. addition (+), subtraction (-)

7.3 Construction of arithmetic models

Input variables, also called *arguments of arithmetic models*, appear in the HEADER of the model. In the simplest case, the HEADER is just a list of arguments, each being a context-sensitive keyword. The model itself is also defined with a context-sensitive keyword.

The model can be in equation form. All arguments of the equation must be in the HEADER. The ALF parser should issue an error if the EQUATION uses an argument not defined in the HEADER. A warning should be issued if the HEADER contains arguments not used in the EQUATION.

Example:

```
DELAY {
      ...
      HEADER {
            CAPACITANCE {...}
            SLEWRATE {...}
      }
      EQUATION {
            0.01 + 0.3*SLEWRATE + (0.6 + 0.1*SLEWRATE)*CAPACITANCE
      }
}
```

If the model uses a TABLE, then each argument in the HEADER also needs a table in order to define the format. The order of arguments decides how the index to each entry is calculated. The first argument is the innermost index, the following arguments are outer indices.

```
DELAY {
      HEADER {
            CAPACITANCE {
                  TABLE {0.03 0.06 0.12 0.24}
            }
            SLEWRATE {
                  TABLE {0.1 0.3 0.9}
            }
      }
      TABLE {
            0.07 0.10 0.14 0.22
            0.09 0.13 0.19 0.30
            0.10 0.15 0.25 0.41
      }
}
```

The first argument CAPACITANCE has 4 entries. The second argument SLEWRATE has 3 entries. Hence DELAY has 4*3=12 entries. For readability, comments may be inserted in the table.

```
TABLE {
//capacitance:0.03 0.06 0.12 0.24
//            -------------------   slewrate:
              0.07 0.10 0.14 0.22 // 0.1
              0.09 0.13 0.19 0.30 // 0.3
              0.10 0.15 0.25 0.41 // 0.9
}
```

Comments have no significance for the ALF parser, nor has the arrangement in rows and columns. Only the order of values is important for index calculation. The table can be made more compact by removing new lines.

```
TABLE { 0.07 0.10 0.14 0.22 0.09 0.13 0.19 0.30 0.10 0.15 0.25 0.41 }
```

For readability, the models and arguments can also have names, i.e. object IDs. For named objects, the name is used for referencing, rather than the keyword.

```
DELAY rise_out{
      ...
      HEADER {
            CAPACITANCE c_out {...}
            SLEWRATE fall_in {...}
      }
      EQUATION {
            0.01 + 0.3 * fall_in + (0.6 + 0.1* fall_in) * c_out
      }
}
```

The arguments of an arithmetic model can be arithmetic models themselves. In this way, combinations of TABLE- and EQUATION-based models can be used, for instance, in derating.

Analogous with FUNCTION, both EQUATION and TABLE representation of an arithmetic model are allowed. The EQUATION is intended to be used when the values of the arguments fall out of range, i.e. to avoid extrapolation.

7.4 Annotations for arithmetic models

Annotations and annotation containers described in this chapter are relevant for the semantic interpretation of arithmetic models and their arguments.

Example: DELAY=f(CAPACITANCE).
DELAY is the arithmetic model, CAPACITANCE is the argument.

Arguments of arithmetic models have the form of annotation containers. They may also have the form of arithmetic models themselves, in which case they represent nested arithmetic models.

7.4.1 DEFAULT annotation

The *default annotation* allows use of the default value instead of the arithmetic model, if the arithmetic model is beyond the scope of the application tool.

```
DEFAULT = number ;
```

Restrictions may apply for the allowed type of `number`. For instance, if the arithmetic model allows only `non_negative_number`, then the default is restricted to `non_negative_number`.

7.4.2 UNIT annotation

The *unit annotation* associates units with the value computed by the arithmetic model.

```
UNIT = string | non_negative_number ;
```

A unit specified by a `string` can take the following values (`*` indicates wildcard):

Table 7-4 : UNIT annotation

Annotation string	Description
f* or F*	equivalent to 1E-15
p* or P*	equivalent to 1E-12
n* or N*	equivalent to 1E-9
u* or U*	equivalent to 1E-6
m* or M*	equivalent to 1E-3
1*	equivalent to 1E+0
k* or K*	equivalent to 1E+3
meg* or MEG*[a]	equivalent to 1E+6
g* or G*	equivalent to 1E+9

a. or uppercase/lowercase combination

Arithmetic models are context-sensitive, i.e. the units for their values can be determined from the context. If `UNIT` annotation for such a context does not exist, default units are applied to the value (Section 7.6.2).

Example:

```
TIME { UNIT = ns; }
FREQUENCY { UNIT = gigahz; }
```

If the unit is a string, then only the first character (respectively the first 3 characters in case of `MEG`) is interpreted. The reminder of the string can be used to define base units. Metric base units are assumed, but not verified, in ALF.

There is no semantic difference between

```
unit = 1sec;
```

and

```
unit = 1volt;
```

Therefore, if the unit is specified as

```
unit = meg;
```

the interpretation is `1E+6`. However, for

```
unit = 1meg;
```

the interpretation is `1` and not `1E+6`.

Units in a non-metric system can only be specified with numbers, not with strings. For instance, if the intent is to specify inch instead of meter as base unit, the following specification will not meet the intent:

```
unit = 1inch;
```

since the interpretation is `1` and meters are assumed.

The correct way of specifying inch instead of meter is

```
unit = 25.4E-3;
```

since 1 inch is (approximately) 25.4 millimeters.

7.4.3 CALCULATION annotation

An arithmetic model in the context of a VECTOR may have the CALCULATION annotation defined as follows:

```
calculation_annotation ::=
   CALCULATION = calculation_identifier ;

calculation_identifier ::=
   absolute
|  incremental
```

It shall specify whether the data of the model are to be used by themselves or in combination with other data. Default is **`absolute`**.

The **`incremental`** data from one VECTOR shall be added to **`absolute`** data from another VECTOR under the following conditions:

- The model definitions are compatible, i.e. measurement specifications must be the same. Units are allowed to be different.
  Example: slewrate measurements at the same pin, same switching direction, same threshold values.
- The model definitions for common arguments are compatible, i.e. same range of values for table-based models, measurement specifications must be the same. Units are allowed to be different.
  Example: same values for derate_case, same threshold definitions for input slewrate.
- The vector definitions are compatible, i.e. the `vector_or_boolean_expression` of the VECTOR containing **`incremental`** data must match the `vector_or_boolean_expression` of the VECTOR containing **`absolute`** data, by removing all variables appearing exclusively in the former expression.

Example:

```
VECTOR ( 01 A -> 01 Z ) {
   DELAY {
         CALCULATION = absolute;
         FROM { PIN = A; } TO { PIN = Z; }
         HEADER {
               CAPACITANCE load { PIN = Z; }
               SLEWRATE slew { PIN = A; }
         }
         EQUATION { 0.5 + 0.3*slew + 1.2*load }
   }
}
VECTOR ( 01 A &> 01 B &> 01 Z ) {
   DELAY {
         CALCULATION = incremental;
         FROM { PIN = A; } TO { PIN = Z; }
         HEADER {
               SLEWRATE slew_A { PIN = A; }
               SLEWRATE slew_B { PIN = B; }
               DELAY delay_A_B { FROM { PIN = A; } TO { PIN = B; } }
         }
         EQUATION {- 0.1 + (0.05+0.002*slew_A*slew_B)*delay_A_B) }
   }
}
```

Both models describe the rise-to-rise delay from A to Z. The second delay model describes the incremental delay (here negative), when the input B switches in a time window between A and Z.

7.4.4 INTERPOLATION annotation

An argument of a table-based arithmetic model, i.e., a model in the HEADER containing a TABLE statement, may have the INTERPOLATION annotation defined as follows:

```
interpolation_annotation ::=
   INTERPOLATION = interpolation_identifier ;

interpolation_identifier ::=
   fit
|  floor
|  ceiling
```

It shall specify, the interpolation scheme for values in-between the values of the TABLE.

- **`fit`**
  The data points in the table are supposed to be part of a smooth curve. Linear interpolation or other algorithms, e.g. cubic spline, polynominal regression, may be used to fit the data points into the curbe.
- **`floor`**
  The value to the left in the table, i.e., the smaller value is used.

- **ceiling**
  The value to the right in the table, i.e., the larger value is used.

Default is **fit**. Note that for multi-dimensional tables, different interpolation schemes may be used for each dimension.

Example:

```
my_model {
   HEADER {
         dimension1 { INTERPOLATION = fit; TABLE { 1 2 4 8 }
         dimension2 { INTERPOLATION = floor; TABLE { 10 100 }
         dimension3 { INTERPOLATION = ceiling; TABLE { 10 100 }
   }
   TABLE {
         1 7 3 5
         10 20 60 40
         50 30 20 100
         0.8 0.4 0.2 0.9
   }
}
```

Consider the following values:

```
   dimension1 = 6
=> following subtable is chosen:
         3     5     // interpolation between 3 and 5
         60    40    // or between 60 and 40
         20    100   // or between 20 and 100
         0.2   0.9   // or between 0.2 and 0.9
   dimension2 = 50
=> following subtable is picked:
         3     5     // interpolation between 3 and 5
         20    100   // or between 20 and 100
   dimension3 = 50
=> following subtable is picked:
         20    100   // interpolation between 20 and 100
```

7.5 Containers for arithmetic models

The following keywords are defined for objects that may contain arithmetic models

Table 7-5 : Unnamed containers for arithmetic models

Objects	Description
FROM	contains start point of timing measurement or timing constraint
TO	contains end point of measurement or timing constraint
LIMIT	contains arithmetic models for limit values

Table 7-5 : Unnamed containers for arithmetic models

Objects	Description
EARLY	contains arithmetic models for timing measurements relevant for early signal arrival time
LATE	contains arithmetic models for timing measurements relevant for late signal arrival time

The LIMIT container is for general use. The FROM, TO, EARLY, LATE containers are only for use within the context of timing models (see chapter ??).

7.5.1 LIMIT container

A LIMIT container shall contain arithmetic models. The arithmetic models shall contain submodels identified by MIN and/or MAX.

Example:

```
PIN data_in {
      LIMIT {
            SLEWRATE { UNIT = ns; MIN = 0.05; MAX = 5.0;}
      }
}
```

The minimum slewrate allowed at pin `data_in` is 0.05 ns, the maximum is 5.0 ns.

```
PIN data_in {
      LIMIT {
            SLEWRATE {
                  UNIT = ns;
                  MAX {
                        HEADER { FREQUENCY { UNIT=megahz;} }
                        EQUATION { 250 / FREQUENCY  }
                  }
            }
      }
}
```

The maximum allowed slewrate is frequency-dependent, e.g. the value is 0.25ns for 1GHz.

7.6 Arithmetic submodels

Arithmetic submodels are for the purpose of distinguishing different measurement conditions for the same model. The root of an arithmetic model may contain nested arithmetic submodels. The header of an arithmetic model may contain nested arithmetic models, but not arithmetic submodels.

The following arithmetic submodels are generally applicable.

Table 7-6 : Generally applicable arithmetic submodels

Objects	Description
MIN	For measured or calculated data: the data represents the minimal value / set of values within a statistical distribution For data within LIMIT container the data represents the lower limit value / set of values
TYP	For measured or calculated data: the data represents the typical value / set of values within a statistical distribution
MAX	For measured or calculated data: the data represents the maximal value / set of values within a statistical distribution For data within LIMIT container the data represents the lower limit value / set of values
DEFAULT	For measured or calculated data: the data represents the default value / set of values to be used per default

The following arithmetic submodels are only applicable in the context of electrical modeling.

Table 7-7 : Submodels restricted to electrical modeling

Objects	Description
HIGH	applicable for electrical data measured at a logic "high" state of a pin
LOW	applicable for electrical data measured at a logic "low" state of a pin
RISE	applicable for electrical data measured during a logic "low" to "high" transition of a pin
FALL	applicable for electrical data measured during a logic "high" to "low" transition of a pin

The following arithmetic submodels are only applicable in the context of physical modeling.

Table 7-8 : Submodels restricted to electrical modeling

Objects	Description
HORIZONTAL	applicable for layout measurements in horizontal direction
VERTICAL	applicable for layout measurements in vertical direction

The semantics of the restricted submodels will be explained in the relevant sections of electrical and physical modeling, respectively.

7.6.1 Semantics of MIN / TYP / MAX

MIN, TYP, MAX indicate that the data of the arithmetic model represent minimal, typical, or maximal values within a statistical distribution. No correlation is assumed or implied between MIN data, TYP data, or MAX data accross different arithmetic models.

Example:

```
DELAY {
   FROM { PIN=A; } TO { PIN=Z; }
   MIN = 0.34; TYP = 0.38; MAX = 0.45;
}
POWER {
   MEASUREMENT = average; FREQUENCY = 1e6;
   MIN = 1.2; TYP = 1.4; MAX = 1.5;
}
```

The MIN value for DELAY may or may not simultaneously apply with the MIN value for POWER. Typically, the case with smaller delay is also the case with larger power consumption.

Within the scope of a LIMIT container, MIN and MAX contain the data for a lower or upper limit, respectively. There must be at least one limit, lower or upper, in each model, but not necessarily both, as shown in the example below.

MIN, MAX inside a model inside a HEADER define the validity range of the data. The model inside the HEADER may contain TABLE or EQUATION. It may also contain HEADER, which represents a nested arithmetic model.

If MIN, MAX is not defined and the data is in a TABLE, the boundaries of the data in the TABLE shall be considered as validity limits.

7.6.2 Semantics of DEFAULT

Arithmetic submodels may be identified by MIN, TYP, MAX or context-restricted keywords. For cases where the application tool cannot decide which qualifier applies, a supplementary arithmetic submodel with the qualifier DEFAULT may be used.

Example:

```
PIN my_pin {
   CAPACITANCE {
         MIN { HEADER { ... } TABLE { ... } }
         TYP { HEADER { ... } TABLE { ... } }
         MAX { HEADER { ... } TABLE { ... } }
         DEFAULT { HEADER { ... } TABLE { ... } }
   }
}
```

Note: The DEFAULT model may also degenerate to a single value and therefore represent a trivial arithmetic model.

In certain cases, there is no supplementary submodel. Instead, one of the already defined submodels is to be used per default. For this case, the DEFAULT annotation may be used to point to the applicable keyword.

Example:

```
PIN my_pin {
   CAPACITANCE {
         MIN { HEADER { ... } TABLE { ... } }
         TYP { HEADER { ... } TABLE { ... } }
         MAX { HEADER { ... } TABLE { ... } }
         DEFAULT = TYP;
   }
}
```

The trivial arithmetic model construct with DEFAULT may also be used for an argument in the context of the HEADER of an arithmetic model. This enables evaluation of the arithmetic model in case the data of the argument can not be supplied by the application tool.

Example:

```
PIN my_pin {
   CAPACITANCE {
         HEADER { TEMPERATURE { DEFAULT=50; TABLE { 0 50 100 } } }
         TABLE { 0.05 0.07 0.10 } }
   }
}
```

The DEFAULT value of CAPACITANCE is 0.07.