# 运算符和表达式

词法单元可组合成更大的构造块，如按照附录B 中Modelica文法表达式部分给定的规则定义表达式。

本章介绍表达式的计算规则，表达式的可变性概念，内置的数学运算符和函数，以及内置的Modelica专用操作及其函数语法。

表达式可包含变量和常量，表达式有类型，预定义的或者用户定义的。预定义的内置Modelica类型包括实型（Real），整型（Integer），布尔型（Boolean）和枚举类型，枚举类型更多的细节在4.8节中介绍。【*下面简短的预定义类型信息是作为余下内容的背景介绍。*】

## 3.1 表达式

Modelica方程、赋值和声明方程包含表达式。

表达式可以包括基本的运算符，+，-，\*，/，^，等等，第3.2节附表和附录B语法中的定义了运算符的一般优先级。标量和数组参数的运算符语义在第10.6节中定义。

同样可以定义函数并按常规的方式调用它们。参数按位和按名的函数调用语法在第12.4.1节介绍，向量化调用语法在第12.4.4节介绍，内置数组函数在第10.1.1节介绍，其他内置运算符在3.7节介绍。

## 3.2 运算符优先级和结合性

运算符优先级决定了表达式中运算符的计算顺序。同一个表达式中，优先级高的运算符比优先级低的先计算。

下表按照运算符优先级顺序从高到低列出了所有的表达式运算符，其来自附录B中的Modelica语法。所有的运算符都是二元运算符，除后缀运算符和那些与表达式expr一起作为一元运算符的运算符，条件运算符，数组构造运算符{}，连接运算符[]，二元或者三元数组范围构造外。表中同一行上的运算符其优先级相同：

表 3-1 运算符

|  |  |  |
| --- | --- | --- |
| **运算符分组** | **运算符语法** | **示例** |
| 后缀数组索引运算符 | [] | arr[index] |
| 后缀访问运算符 | . | a.b |
| 后缀函数调用 | funcName(function-rguments) | Sin(4.36) |
| 数组构造/合并 | {expressions} [expressions]  [expressions; expressions...] | {2,3} [5,6]  [2,3; 7,8] |
| 乘方 | ^ | 2^3 |
| 乘法和数组点乘 | \* / .\* ./ | 2\*3 2/3  [1,2;3，4].\*[2,3;5,6] |
| 加法和数组点加 | + - +expr -expr  .+ .- | a+b， a-b， +a， -a [1,2;3,4].+[2,3;5,6] |
| 关系运算符 | < <= > >= == <> | a<b, a<=b, a>b，... |
| 一元否定 | not expr | not b1 |
| 逻辑与 | And | b1 and b2 |
| 逻辑或 | or | b1 or b2 |
| 数组范围 | expr : expr  expr : expr : expr | 1:5  start:step:stop |
| 条件运算符 | if expr then expr else expr | if b then 3 else x |
| 命名参数 | ident=expr | x = 2.26 |

条件运算符也可以包括elseif子句。相等“=”和赋值“:=”不是表达式运算符，因为它们只允许单独出现在方程和赋值语句中。所有二元运算符是左结合的，除乘方运算符外（它是非结合的）。数组范围运算符是非结合的。

【*Modelica中一元减法和加法运算符与Mathematica和MATLAB[[1]](#footnote-1)有细微差别，下列表达式是非法的（然而在Mathemmatica[[2]](#footnote-2)和MATLAB中它们是合法的表达式）：*

2\*-2 // = -4 在Mathematica/MATLAB中；在Modelica中是非法的

--2 // = 2在Mathematica/MATLAB中；在Modelica中是非法的

++2 // = 2在Mathematica/MATLAB中；在Modelica中是非法的

2--2 // = 4在Mathematica/MATLAB中；在Modelica中是非法的

*非结合的乘方和数组范围运算符*

x^y^z // 非法的，需用圆括号将其表达清楚。

a:b:c:d:e:f:g // 非法的，向量参数没有给出合法的解释。

】

## 3.3 计算顺序

Modelica工具可以自由地对方程求解，对表达式重新排序，而不用对表达式求值，如果其值对结果没有影响（如布尔型表达式的短路求值）。if语句和if表达式确保只在合适的条件为true才对它们的子句求值，但是产生状态或者时间事件的关系运算符在连续积分期间将从最近一次事件中获取值。

如果数值运算溢出则其结果为未定义。对于字面量，推荐将数值自动转换为更高精度的其他类型。

### 3.3.1 示例：防止计算错误的监护表达式

*【用if监护语句防止表达式计算错误：*

Boolean v[n];

Boolean b; Integer I; **equation**

x=v[I] **and** (I>=1 **and** I<=n); // 非法

x=**if** (I>=1 **and** I<=n) **then** v[I] **else** false; // Correct To guard square against square root of negative number use noEvent:

**der**(h)=**if** h>0 **then** –c\*sqrt(h) **else** 0; // 错误

**der**(h)=**if** noEvent(h>0) **then** –c\*sqrt(h) **else** 0; // 正确

*】*

## 3.4 算术运算符

Modelica提供5种算术运算符来对任意数据类型进行运算：

^ 乘方

\* 乘法

/ 除法

+ 加法

- 减法

这些运算符中有些能用于标量类型和数组类型的组合，参见第10.6节。

下面是modelica文法中定义的运算符语法：

arithmetic\_expression:

[ add\_op ] term { add\_op term }

add\_op :

"+" | "-"

term :

factor { mul\_op factor }

mul\_op :

"\*" | "/"

factor :

primary [ "^" primary ]

## 3.5 相等、关系和逻辑运算符

Modelica提供一组标准的关系运算符和逻辑运算符，所有这些运算符产生标准的布尔值true或者false。

> 大于

>= 大于或等于

< 小于

<= 小于或等于

== 表达式内相等

<> 不等于

单个等号“=”不能用于关系表达式，仅仅用于方程（第8章，第10.6.1节）和按名称传递参数的函数调用（12.4.1节）中。

逻辑运算符定义如下：

not 否定，一元运算符

and 逻辑与

or 逻辑或

文法规则定义了关系和逻辑运算符的语法。

logical\_expression :

logical\_term { **or** logical\_term }

logical\_term :

logical\_factor { **and** logical\_factor }

logical\_factor :

[ **not** ] relation

relation :

arithmetic\_expression [ rel\_op arithmetic\_expression ]

rel\_op :

"<" | "<=" | ">" | ">=" | "==" | "<>"

以下规则适用于关系运算符：

* 关系运算符<，<=，>，>=，==，<>的定义仅针对简单类型的标量操作数。根据其条件满足与否，布尔类型结果的值分别为true或者false。
* 对于字符串类型的操作数，表达式“str1 op str2”中的每个关系运算符“op”，将按照C函数strcmp定义为strcmp(str1，str2) op 0。
* 对于布尔类型操作数，false<true。
* 对于枚举类型的操作数，运算顺序由枚举字面量声明的顺序给定。
* 在形如“v1 == v2 ”或者“v1 <> v2”的关系中，除非用于函数，v1或者v2不能是实数的子类型。【*定义这个规则的原因是带实数参数的关系被转换为状态事件（state events）（参见第8.5节，事件），而且对于==和<>运算符这个转换变得不必要的复杂（例如，需要用二次交叉函数代替一次交叉函数，在事件实例中甚至需要使用epsilon策略*），此外，在那些寄存器数值长度与主存数值长度不同的机器上测试实型变量的相等是有问题的。】
* “v1 rel\_op v2”形式的关系，其中v1和v2是变量，rel\_op是一个关系运算符，称作基本关系（elementary relations）。如果v1和v2中的一个或两个变量都是实数的子类型，这个关系称作实数基本关系。

## 3.6 其他运算符和变量

Modelica还包括一些内置的运算符，它们不是标准的算术、关系或者逻辑运算符，这些运算符在下面介绍，包括time。time是一个内置变量，而不是一个运算符。

### 3.6.1 字符串连接

在Modelica中，字符串连接（参见Modelica文法）由“+”运算符表示【*例如"a" + "b" 变成 "ab"*】。

### 3.6.2 数组构造运算符

数组构造运算符“{…}”在第10.4节介绍。

### 3.6.3 数组连接运算符

数组连接运算符“[…]”在第10.4.2节介绍。

### 3.6.4 数组范围运算符

数组范围构造运算符在第10.4.3节介绍。

### 3.6.5 If表达式

**if** expression1 **then** expression2 **else** expression3

是if表达式的一个例子。首先，表达式“expression1”被求值（expression1必须是一个布尔类型表达式），如果其值为true，则对表达式“expression2”求值并作为条件表达式的值，否则表达式对“expression3”求值并作为if表达式的值。“expression1”和“expression2”这两个表达式必须与第6.6节给出的条件表达式的类型兼容。带有elseif的条件表达式使用else if代替elseif定义。*【说明：增加*elseif*是为了与条件子句对称】。*短路求值参见第3.3节。

【示例：

Integer i;

Integer sign\_of\_i1=**if** i<0 **then** -1 **elseif** i==0 **then** 0 **else** 1;

Integer sign\_of\_i2=**if** i<0 **then** -1 **else** **if** i==0 **then** 0 **else** 1;

】

### 3.6.6 成员访问运算符

可以使用点符号即“.”运算符，访问类实例成员。

【*示例：R1.R访问电阻器R1的电阻组件R。点符号的另一种用途：局部类是类的成员，同样可以在类名上而不是类实例上用点符号访问。*】

### 3.6.7 内置时间变量

所有声明的time变量都是以time为自变量的函数。time变量是内置的变量，它可以用于所有模型和构造块，并被视作输入变量。它被隐式定义为：

**input** Real time (**final** quantity = "Time"，

**final** unit = "s");

time变量初始值被设置为仿真开始的时刻。

【*示例：*

**encapsulated model** SineSource

**import** Modelica.Math.sin;

**connector** OutPort=**output** Real;

OutPort y=sin(time); // 使用内置time变量。

**end** SineSource;

】

## 3.7 内置的固有的运算符及其函数语法

某些Modelica内置操作具有与函数调用相同的语法，但它们并不就是一个数学函数，因为计算结果不仅仅依赖输入参数还依赖仿真的状态。

也有仅仅依赖输入参数的内置函数，但除了返回值之外它们还可能触发事件。“固有”意味着它们是定义在Modelica语言级别，而不是在Modelica库中。以下是可用的内置固有的运算符函数：

* 数学函数和转换函数，参见下面第3.7.1节。
* 导数和专用操作及其函数语法，参见下面第3.7.2节。
* 事件相关的操作及其函数语法，参见下面第3.7.3节。
* 数组运算符/函数，参见第10.1.1节。

除了内置运算符String(…)，本节中所有的操作都只能以参数按位调用的方式调用。

### 3.7.1 数值函数和转换函数

下表是Modelica预定义的数学运算符和函数，以及一些转换函数，且可以根据第12.4.6节的描述向量化，String函数除外。不触发事件的函数在下表中介绍，反之触发事件的数学函数在第3.7.1.1节中介绍。

|  |  |
| --- | --- |
| abs(v) | 扩展为“noEvent(if v >= 0 then v else –v)”。参数v必须为整型或者实数表达式。 |
| sign(v) | 扩展为“noEvent(if v>0 then 1 else if v<0 then –1 else 0)”。参数v必须是整型或者实数表达式。 |
| sqrt(v) | 如果参数v>=0，返回v的平方根，否则报错。参数v必须是整型或者实型。 |
| Integer(e) | 返回枚举类型表达式e的序数。对于枚举类型E=enumeration(e1， ...， en)，计算枚举项E.enumvalue的值，其中Integer(E.e1)=1，Integer(E.en)= n。另请参见第4.8.5.2节。 |
| String(b, <options>) String(i, <options>)  String(r, significantDigits=d, <options>)  String(r,format=s)  String(e,<options>) | 将标量非字符串表达式转换为字符串表示形式。第一个参数可以是Boolean b，Integer i，Real r或者一个枚举型e（第4.8.5.2节）。其他参数必须是命名参数。可选参数<options>可以是：  Integer minimumLength=0：结果字符串的最小长度。如果必要，可用空白字符填充没有使用的空间。  Boolean leftJustified = true：如果leftJustified的值为true，转换结果为左对齐的字符串；如果值为false结果为右对齐的字符串。  对于实数表达式，输出必须遵照Modelica语法的规定。Integer significantDigits=6：定义结果字符串有效数字位数。【例如："12.3456"， "0.0123456"， "12345600"， "1.23456E-10"】。不同类型option选项对应的格式化字符串是：   * 对于实型：(if leftJustified then "-" else "")+String(minimumLength)+"."+ String(signficantDigits)+"g"， * 对于整型：(if leftJustified then "-" else "")+String(minimumLength)+"d"。   格式化字符串：按照ACSI-C规定，由格式化字符串指定一个转换说明符（前面的“%”除外），不能包含长度修饰符，不能对宽度和/或精度使用“\*”。对于所有数值，都允许使用格式说明符“f，e，E，g，G”。对于整数还允许使用“d，i，o，x，X，u”和C语言格式说明符（对于非整数值，如果使用了整型转换字符，Modelica工具可能圆整、截断或者使用一个不同的格式）。  对于整型值，x，X格式（16进制）和c（字符）不会导致与Modelica语法一致的输入。 |

#### 3.7.1.1 触发事件的数学函数

如果用在when子句以及钟控离散时间片（clocked discrete-time partition）之外，本节中的内置函数将触发状态事件（参见第16.8.1节）。【*如果不希望如此，可在其上应用noEvent函数，例如noEvent(integer(v))*】

|  |  |
| --- | --- |
| div(x，y) | 返回x/y的代数商并舍弃所有的小数部分（也称作向零截断）。【*注意：这是C99对“/”的定义；在C89中，负数的结果取决于实现，因必须使用标准的div()函数。*】结果和参数应该为实型或者整型。如果参数为实型则结果为实型，否则为整型。 |
| mod(x，y) | 返回x/y的整型模数，即mod(x,y)=x-floor(x/y)\*y。结果和参数应该为实型或者整型。如果参数为实型则结果为实型，否则为整型。【*注意：当返回值不连续变化的时候，在when子句外会触发状态事件。例如 mod(3,1.4)=0.2， mod(-3,1.4)=1.2， mod(3,-1.4)=-1.2*】 |
| rem(x，y) | 返回x/y的整型余数，例如div(x,y)\*y + rem(x, y) = x。结果和参数应该为实型或者整型。如果参数为实型则结果为实型，否则为整型。【*注意：当返回值不连续变化的时候，在when子句外会触发状态事件。例如rem(3，1.4)=0.2， rem(-3，1.4)=-0.2*】 |
| ceil(x) | 返回小于x的最小整数。结果和参数必须是实型。【*注意：当返回值不连续变化的时候，在when子句外会触发状态事件。*】 |
| floor(x) | 返回不大于x的最大整数。结果和参数必须是实型。【*注意：当返回值不连续变化的时候，在when子句会触发状态事件。*】 |
| integer(x) | 返回不大于x的最大整数。参数必须是实型，结果为整型。  【*注意：当返回值不连续变化的时候，在when子句外会触发状态事件。*】 |

#### 3.7.1.2 内置数学函数和外部内置函数

下表是Modelica中可用的内置数学函数，能够直接调用而不需要在函数名称前加任何包前缀。作为外部内置函数，Modelica.Math库也提供了这些函数。

|  |  |
| --- | --- |
| sin(x) | 正弦 |
| cos(x) | 余弦 |
| tan(x) | 正切（x必须不是： ...,-π/2,π/2,3π/2,...） |
| asin(x) | 反正弦（-1 ≤ x ≤ 1） |
| acos(x) | 反余弦（-1 ≤ x ≤ 1） |
| atan(x) | 反正切 |
| atan2(y, x) | atan2(y,x)函数计算y/x反正切的主值，由这两个参数的符号决定结果的象限。 |
| sinh(x) | 双曲正弦 |
| cosh(x) | 双曲余弦 |
| tanh(x) | 双曲正切 |
| exp(x) | 指数，基于e |
| log(x) | 自然对数（以e为底）（x>0） |
| log10(x) | 以10为底的对数（x>0） |

### 3.7.2 导数和专用操作及函数语法

下表是Modelica语言预定义的导数运算符和专用运算符及其函数语法：

|  |  |
| --- | --- |
| der(expr) | 返回表达式expr的时间导数。如果表达式expr是一个标量，它必须是实数的子类型。表达式和它所有的子表达式必须是可微的。如果expr是一个数组，该运算符作用于它所有的元素上。对于非标量参数，本函数按照第10.6.12节所述矢量化。【*对于实型参数和常量，其结果是标量零或者是大小与该变量大小相同的数组。*】 |
| delay(expr，delayTime， delayMax)  delay(expr，delayTime) | 当time>time.start + delayTime时返回expr(time–delayTime)，当time <= time.start + delayTime时返回expr(time.start)。这些参数，即expr，delayTime，delayMax，必须为实型的子类型，另外delayMax必须是一个参数表达式。必须保持关系：0 <= delayTime <= delayMax，否则会出错。如果delayMax没有在参数列表中提供，delayTime必须为一个参数表达式。参见第3.7.2.1节。对于非标量参数，本函数按照第10.6.12节所述向量化。 |
| cardinality(c) | 【*已弃用。在Modelica下一个版本中将被删除，不应再使用。*】  以整数的形式返回连接方程中（内部或者外部）连接器示例c出现的次数。参见第3.7.2.3节） |
| homotopy(actual=actual， simplified=simplified) | 标量表达式“actual”和“simplified”是实数的子类型。Modelica翻译器应将本运算符映射为下面两种形式之一：   1. 返回“actual”【*普通的实现*】 2. 为了求解方程的代数系统，在求解过程中，本操作应该返回以actual结尾的两个参数的组合，【*例如，actual\*lambda + simplified\*(1-lambda)，其中lambda是从0到1的同伦参数*】。求解过程必须保证同伦方程返回“actual”。   参见第3.7.2.4节。对于非标量参数，本函数按照第12.4.6节所述向量化。 |
| semiLinear(x， positiveSlope， negativeSlope) | 如果x>=0，返回positiveSlope\*x，否则返回negativeSlope\*x。返回实数类型结果。参见第3.7.2.5节【*特别是当x=0的情况下*】。对于非标量参数，本函数按照第10.6.12节所述向量化。 |
| inStream(v) | inStream(v)运算符只允许用在流连接器（Stream Connector）中定义的流变量（Stream Variable）v上，并且返回的是靠近连接点的流变量v的值，假定流是从连接点到组件。这个值是从流动变量（flow variable）的流连接方程和流变量变量计算而来。本操作可向量化。更多的细节参见第15.2节。 |
| actualStream(v) | actualStream(v)操作返回任意方向的流变量v的实际值。本操作是可向量化的。更多的细节参见第15.3节。 |
| spatialDistribution( in0， in1， x， pv， iP， iV) | spatialDistribution(…)操作能估算可变速传输特性，参见第3.7.2.2节。 |
| getInstanceName() | 返回仿真模型/块的名称加上调用函数示例的完全限定名称的字符串。参见第3.7.2.6节。 |

下文中详细介绍其中的几个运算符。

#### 3.7.2.1 delay

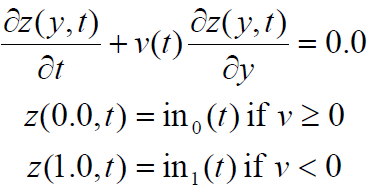
【*通过在（内部）积分多项式上插值，deylay()操作可得到一个可靠数值实现，此外通过在包含表达式expr历史值的缓冲上进行线性插值得到一个简单的数值实现。如果没有更多的信息，全部的延迟信号时间历史信息就需要被存储起来，因为在仿真期间延迟时间可能会改变。为了避免过度的存储需求以及提高效率，必须通过delayMax给出允许的最大延迟时间。*

*这给出了已存储的延迟信息值的上限。对于使用固定步长积分器的实时仿真，在仿真开始前，这些信息对在内部缓冲区上分配必须的存储空间是足够的。对于变步长积分器，积分期间缓冲大小是动态变化的。原则上，delay操作能够打断代数环。为简单起见，这一点不被支持，因为在编译时最小延迟时间必须作为额外的参数给出固定值。此外，为了避免在延迟缓冲中做外插计算，积分最大步长受限于最小延迟时间。*】

#### 3.7.2.2 spatialDistribution

【*许多应用涉及可变速传输特性建模。对这种无限维系统建模一个选择是用ODE进行估算，但是这需要大量的状态变量，并且可能导致数值扩散或者数值震荡。另一个选择是通过合适的采样、插值和存储分布的位移，使用内置操作持续跟踪函数z(y，t)的空间分布。在这种情况下，ODE求解器隐藏了操作的内部状态。*】

spatialDistribution()操作允许高效地估算无限维问题的解



其中z(y，t)是传输量，y是归一化的空间坐标（0.0 ≤ y ≤ 1.0)，t是时间，v(t)=der(x)是正常的传输速度，边界条件设置在y=0.0或者y=1.0，这取决于速度的符号。调用语法是：

(out0， out1) = spatialDistribution(in0， in1， x， positiveVelocity initialPoints = {0.0， 1.0}，

initialValues = {0.0， 0.0});

其中in0， in1， out0， out1， x， v都是实数的子类型。positiveVelocity是布尔型，initialPoints 和 initialValues是大小相同的实数的子类型数组，它们包含y坐标和描述z(y，t0)的初始分布的有限点集的z值。out0和out1的值由z(0.0， t)和z(1.0， t)的解给出。in0和in1是在z(0.0， t) 和 z(1.0， t)的边界条件（每个时间点上只使用一个in0和in1)。initialPoints数组元素必须按非降序排序。按照第12.4.6节描述的向量化规则，本操作不能被向量化。本操作只有参数in0和in1（in0和in1必须大小相同）能够向量化，返回大小相同的向量化的输出out0和out1；参数initialPoints和initialValues相应地被向量化。

z(..)的解可以根据特性：

*z*(*y* + ∫*tt*+β*v*(α)*d*α，*t* +*β*) = *z*(*y*， *t*) 描述， 对于所有*β*，只要在值域内。

这允许基于边界条件插值直接求解。**spatialDistribution**操作可以用以伪代码给出的构造块描述：

**block** spatialDistribution

**input** Real in0;

**input** Real in1;

**input** Real x;

**input** Boolean positiveVelocity;

**parameter** Real initialPoints(**each** min=0,**each** max=1)[:] = {0.0,1.0};

**parameter** Real initialValues[:] = {0.0， 0.0};

**output** Real out0;

**output** Real out1;

**protected**

Real points[:];

Real values[:];

Real x0;

Integer m;

**algorithm**

**if** positiveVelocity **then**

out1:=interpolate(points, values, 1—(x-x0));

out0:=values[1]; // similar to in0 but avoiding algebraic loop

**else**

out0:=interpolate(points,values,(x-x0));

out1:=values[end]; // similar to in1 but avoiding algebraic loop

**end if**;

**when** <*acceptedStep>* **then**

**if** x>x0 **then**

m:=size(points,1);

**while** (if m>0 **then** points[m]+(x-x0)>=1 else false) **then**

m:=m-1;

**end while**;

values:=cat(1, {in0}， values[1:m], {interpolate(points, values,1-(x-x0))} );

points:=cat(1, {0}, points[1:m] .+ (x1-x0)， {1} );

**elseif** x<x0 **then**

m:=1;

**while** (if m<size(points,1) then points[m]+(x-x0)<=0 else false) then

m:=m+1;

**end while**;

values:=cat(1, {interpolate(points, values, 0-(x-x0))},values[m:end],{in1});

points:=cat(1, {0}, points[m:end] .+ (x1-x0), {1});

**end if**;

x0:=x;

**end when**;

**initial algorithm**

x0:=x;

points:=initialPoints;

values:=initialValues;

**end** spatialDistribution;

【*上面描述的无限维问题能够按下面的方式阐述：*

*der(x) = v;*

*(out0,out1)=****spatialDistribution****(in0,in1,x,v>=0,initialPoints,initialValues);*

*当速度符号改变的时候，事件在这瞬间产生——如果这不是必须的，可以用noEvent()阻止事件产生。*

*如果已知速度一直为正数，那么out0可以被忽略，例如：*

*der(x) = v;*

*(,out1) = spatialDistribution(in0, 0, x, true, initialPoints, initialValues);*

*技术上使用spatialDistribution()操作相关的案例有电气传输线路建模，煤气管道和管道网络，供水和集中供暖，洒水系统，拉伸体内的脉冲传播，传送带，和液压系统。上面的例子中，当不止一个数以速度v传播的时候，管道必须向量化。*】

#### 3.7.2.3 cardinality（已弃用）

【*基于以下原因，cardinality操作已弃用，并将在未来的版本中移除。*

* *反射操作可能导致早期类型检查更困难。*
* *几乎总是被以奇怪的方式滥用。*
* *没有用于键合图，尽管当初是为这个目的而引入。*

】

【*cardinality（） 操作在模型中定义连接依赖方程，例如：*

**connector** Pin

Real v;

**flow** Real i;

**end** Pin;

**model** Resistor

Pin p, n;

**equation**

**assert(cardinality(p) > 0 and cardinality(n) > 0， "Connectors p and n of Resistor must be connected");**

// Equations of resistor

...

**end** Resistor;

】

基数在移除条件组件以后计算，且不可用于可扩展连接器、可扩展连接器中的元素、或者连接器数组（但是能用于连接器数组标量元素）。cardinality操作只能用于断言条件和不包含连接（及类似的操作——参见第8.3.3节）的if声明语句中。

#### 3.7.2.4 homotopy

【*在动态仿真问题的初始阶段，经常有大量的非线性方程系统需要借助于迭代求解器求解。迭代求解器的收敛严重地依赖于选择未知变量的初始估值。通过提供模型的替代简化版本，然后连续不断地将简化模型转换为实际模型，迭代过程能更健壮地执行，收敛甚至可能不需要精确的初始估值。在系统方程的表述中，这个转换可以用表达式表述为：*

*lambda\*actual + (1-lambda)\*simplified*

*通常也称之为同伦转换。如果仔细选择simplified表达式，问题的解随着lambda连续改变，因此经过足够的很少几个步骤，可能得到问题actual最终的解。*

*homotopy操作可以按参数顺序调用，或者最好按参数名称调用以提高代码的可读性。*

*推荐在整个模型上只执行一个homotopy迭代器，且不要对非线性代数方程系统分别执行多个homotopy。原因如以下结构所示：*

**w** = **f**1(**x**) // 有homotopy操作

**0** = **f**2(der(**x**), **x**, **z**, **w**)

*这里，f2是一个非线性方程系统。homotopy操作作用在这个非线性代数系统的“输入”变量上，并且改变了这个非线性代数系统的特性。唯一有用的方式是在f1和f2上一起执行homotopy迭代。*

*上面建议的方法是“概念性的”，因为更高效的实现是有可能的，例如，通过决定最小循环迭代循环，这包含出现homotopy操作的第一个BLT块的方程以及直到描述一个非线性代数方程系统的最后一个BLT块中的所有方程。*

*通过在全局范围内定义下面的函数，可以简单实现一个homotopy操作：*

**function** homotopy

**input** Real actual;

**input** Real simplified;

**output** Real y;

**algorithm**

y := actual;

**annotation**(Inline = true);

**end** homotopy;

*示例1：*

*在电气系统中，如果开关是代数方程的一部分，常常很难求解非线性代数方程。通过以“平直”二极管特性开始，然后用homotopy操作移动到期望的“陡峭”特性，一个理想的二极管模型可用下面的方式实现：*

**model** IdealDiode

...

**parameter** Real Goff = 1e-5;

**protected**

Real Goff\_flat = max(0.01， Goff);

Real Goff2;

**equation**

off = s < 0;

Goff2 = **homotopy**(actual=Goff， simplified=Goff\_flat);

u = s\*(**if** off **then** 1 **else** Ron2) + Vknee;

i = s\*(**if** off **then** Goff2 **else** 1 ) + Goff2\*Vknee;

...

**end** IdealDiode;

*示例2：*

*在电气系统中，既然零电源稳态初始化容易得到，所有的电压源电压从零开始以及所有的电流源电流从零开始，通常是很有用的。一个典型的电压源可以定义为：*

**model** ConstantVoltageSource

**extends** Modelica.Electrical.Analog.Interfaces.OnePort;

**parameter** Modelica.SIunits.Voltage V;

**equation**

v = **homotopy**(actual=V, simplified=0.0);

**end** ConstantVoltageSource;

*示例3：*

*在流体系统建模中，由于二次项，由于对流体特性的依赖，压力/流速关系是高度非线性的。使用一个简化的线性模型（调谐标称计算点），能使整个模型更少线性化并因此更容易求解而不需要精确的初始值。这里使用命名参数是为了进一步提高可读性。*

**model** PressureLoss

**import** SI = Modelica.SIunits;

...

**parameter** SI.MassFlowRate m\_flow\_nominal "Nominal mass flow rate";

**parameter** SI.Pressure dp\_nominal "Nominal pressure drop";

SI.Density rho "Upstream density";

SI.DynamicViscosity lambda "Upstream viscosity";

**equation**

...

m\_flow=**homotopy**(actual=turbulentFlow\_dp(dp,rho,lambda), simplified = dp/dp\_nominal\*m\_flow\_nominal);

...

**end** PressureLoss;

*示例4：*

*注意homotopy操作****不能****用于合并相互没有关系的表达式，因为在合并两个定义明确的系统时可能产生奇异系统。*

**model** DoNotUse

Real x;

**parameter** Real x0 = 0;

**equation**

der(x) = 1-x;

**initial equation**

0 = homotopy(**der**(x), x - x0);

**end** DoNotUse;

*初始化方程展开为：*

0 = lambda\*der(x) + (1-lambda)\*(x-x0)

*求解这个两个方程可以得到：*

x = (lambda+(lambda-1)\*x0)/(2\*lambda - 1)

*当lambda=0时得到正确的值x0，lambda=1时得到正确值1，然而不幸的是lambda=0.5时有一个奇点。*

】

#### 3.7.2.5 semiLinear

（参见第3.7.2节semiLinear的定义）。

在某些情况下，如果第一个参数（x）变成0，带有semiLinear()函数的方程变成欠定的，即有一个无穷大解。在这种情况下，为了得到一个有意义的解，在翻译期间推荐使用下面规则转换方程：

**规则1：下列方程**

y = semiLinear(x, sa, s1);

y = semiLinear(x, s1, s2);

y = semiLinear(x, s2, s3);

...

y = semiLinear(x, sN, sb);

...

可以替换为

s1 = **if** x >= 0 **then** sa **else** sb

s2 = s1;

s3 = s2;

...

sN = sN-1;

y = semiLinear(x, sa, sb);

规则2：下列方程

x = 0;

y = 0;

y = semiLinear(x, sa, sb);

可以替换为

x = 0

y = 0;

sa = sb;

【*对于符号转换，下面的特性是有用的（按照定义）：*

semiLinear(m\_flow， port\_h， h);

*等同于：*

-semiLinear(-m\_flow， h， port\_h);

*semiLinear函数的目的是处理流体系统中的反向流，例如*

H\_flow =semiLinear(m\_flow， port.h， h);

*也就是说，焓流率H\_flow 从质流率计算得到，上游比焓取决于流向。*

】

#### 3.7.2.6 getInstanceName

返回仿真模型/构造块名称加调用本函数实例的完全限定名称的字符串。

【*示例:*

package MyLib

model Vehicle

Engine engine;

...

end Vehicle;

model Engine

Controller controller;

...

end Engine;

model Controller

equation

Modelica.Utilities.Streams.print("Info from: " + getInstanceName());

end Controller;

end MyLib;

*如果MyLib.Vehicle被仿真，调用getInstanceName()将返回“Vehicle.engine.controller”*

】

如果本函数不是在模型或者块内调用（例如在函数或者包内常量内调用），函数返回值是无法确定的。

【*仿真结果不应依赖本函数的返回值。*】

### 3.7.3 事件相关操作及函数语法

下表是Modelica支持的与时间相关的操作及其函数语法。noEvent，pre，edge和change等操作可以根据12.4.6节的描述向量化。

|  |  |
| --- | --- |
| initial() | 初始化阶段返回true否则返回false【*从而在仿真开始的时候触发一个时间事件*】 |
| terminal() | 在成功分析后返回true【*从而保证仿真成功后触发一个事件*】。 |
| noEvent(expr) | 表达式expr内的实型基本关系被当作字面量，即不触发状态和时间事件。参见第3.7.3.2节和第8.5节。 |
| smooth(p， expr) | 如果p>=0，smooth(p，expr)返回expr并表明expr是p次连续可微的，即expr在表达式中的所有实型变量中都是连续的，并且对于出现的所有实型变量其存在偏导，并直到p阶都是连续的。参数p必须是一个标量整型参数表达式。smooth操作中expr的类型只允许是：实型表达式，允许的表达式数组，和只包含允许的表达式的组件的记录。参见第3.7.3.2节。 |
| sample(start，interval) | 在时间点start + i\*interval （i=0，1，...)返回true并触发时间事件，在连续积分期间一直返回false。采样开始时间start和采样间隔interval应为参数表达式，而且应为实型和整型的子类型。 |
| pre(y) | 返回变量y(t)在时间点t的“左极限”y(tpre) 。  在事件触发时刻，y(tpre)是时间点t最后一次事件迭代后变量y的值（见下面的注释）。如果下面三个条件同时满足，则能应用pre()操作：（a）变量y是简单类型的子类型或者是一个记录组件，（b）y是一个离散时间表达式，（c）pre操作没有在函数类中应用。【*注意：能应用于when子句内的连续时间变量上。参见第3.8.3节离散时间表达式的定义。*】pre(y)的第一个值在初始化阶段确定。另请参阅第3.7.3.1节。 |
| edge(b) | 将布尔型变量b展开为“(b and not pre(b))”。对pre()操作的限制同样适用于edge操作（例如不用于函数类中）。 |
| change(v) | 展开为“(v<>pre(v))”。对pre()操作的限制同样适用于change操作。 |
| reinit(x， expr) | 在when子句代码体中，在事件触发时刻用expr重新初始化x。x是一个实型变量（或者实型变量数组），被隐式定义为有状态选择【*因此x必须选择为状态，如果不可能的话就是一个错误*】。在相同变量上reinit操作只能用一次。它只能用于when子句体的方程段中。参见第8.3.6节。 |

下文中详细介绍其中的几个操作。

#### 3.7.3.1 pre

在某个事件触发时刻，在活动的模型方程计算后，如果至少存在一个变量v满足“pre(v)<>v”，则触发一个新的事件。在这种情况下，模型立即被重新计算。该计算序列称为“事件迭代”。如果所有在pre操作中使用的变量v保持“pre(v) == v”，则重新启动积分。

【*如果v和pre(v)只使用在when子句中，既然v在事件迭代过程中不能改变，那么翻译器可能屏蔽对变量v屏蔽事件迭代。找到最小事件迭代循环是一个“实现质量”问题，即不是模型所有部分都需要重新计算。*

*Modelica语言允许混合方程代数系统，其中未知变量是实型、整型、布尔型，或者枚举类型。在单次迭代期间，通过固定布尔型、整型，和/或枚举类型等类型未知变量，这些方程系统能用一个全局固定点迭代方案求解，类似于事件迭代。此外，更高效地实现这些方程系统的求解是一个质量问题，例如，将固定点迭代方案应用于模型方程子集。*】

#### 3.7.3.2 noEvent 和 smooth

noEvent操作意味着实际的基本表达式是从字面上理解而不是生成跨越函数，参见第8.5节。出于性能的原因，要避免事件应使用smooth操作而不是noEvent。在smooth内，工具有不为表达式产生事件的自由。然而，smooth不保证没有事件产生，因此有必要时可以在smooth内使用noEvent。【注意，如果出现变量不连续变化的情况，smooth不保证平滑输出。】

[Example:

Real x，y，z; **parameter** Real p; **equation**

x = **if** time<1 **then** 2 **else** time-2; z = smooth(0， **if** time<0 **then** 0 **else** time); y = smooth(1， noEvent(**if** x<0 **then** 0 **else** sqrt(x)\*x));

// noEvent is necessary.

]

## 3.8 表达式可变性

表达式的可变性概念是指表达式随着时间改变的程度。参见第4.4.4节可变性的概念。从最低可变性开始，表达式的可变性有4个层次：

* + 常量可变性
  + 参数可变性
  + 离散时间可变性
  + 连续时间可变性

对于赋值语句v := expr或者绑定方程v = expr，v必须至少声明为expr的变量。

* + 参数组件和基本类型属性【*例如start*】的绑定方程里的右侧表达式【*即expr*】必须是一个参数或者常量表达式。
  + 如果v是一个离散时间组件那么expr必须是一个离散时间表达式。

### 3.8.1 常量表达式

常量表达式有：

* 实型，整型，布尔型，字符串，和枚举字面量。
* 声明为常量的变量。
* 除了特殊的内置操作initial，terminal，der，edge，change，sample和pre，常量子表达式作为参数（并且函数内没有参数定义）的函数或者操作是常量表达式。

如果常量直接在仿真模型或者在仿真模型中使用，声明为常量的组件应有一个关联的含有常量表达式的方程。常量的值在赋值以后可以修改，除非常量声明为final或者被一个final修饰符修饰。没有关联的声明方程的常量可以用修饰符指定一个。

### 3.8.2 参数表达式

参数表达式有：

* 常量表达式。
* 声明为参数的变量
* 除了特殊内置操作initial，terminal，der，edge，change，sample和pre外，有参数子表达式的函数或者操作是参数表达式。

### 3.8.3 离散时间表达式

离散时间表达式有：

* 参数表达式。
* 离散时间变量，即Integer，Boolean，String类型变量和枚举变量，以及在when子句中赋值的实型变量。
* 所有输入参数都是离散时间表达式的函数调用。
* 所有子表达式都是离散时间表达式的表达式。
* when子句中的表达式，初始化方程，或者初始化算法。
* 除非在noEvent内部：如果至少有一个操作数是实型的子类型（即实型基本关系，参见第3.5节），有序关系（>，<，>=，<=），函数ceil，floor，div，mod，rem是离散时间表达式。如果至少一个子表达式不是离散表达式，它们将产生事件。【*换句话说，noEvent()内的关系，如noEvent(x>1)，不是离散时间表达式。*】
* 函数pre，dege和change产生离散时间表达式。
* 函数内的表达式相当于离散时间表达式。

对于方程expr1 = expr2，如果两个表达式都不是实型基本类型，两个表达式都必须是离散时间表达式。对于记录方程，在根据这个规则检验前，方程被分解为基本类型。【*这个限制保证noEvent()操作不能在when子句外应用到布尔型、整型、字符串或者枚举方程上，因为那时这两个表达式之一不是离散时间的。*】

在if表达式，if子句，while声明或者for子句中，即非离散时间（连续时间，但不是离散时间）控制的开关表达式内以及不在when子句中，对离散变量，离散表达式之间的方程，或者产生事件的实型基本关系/函数赋值是非法的。【为了保证离散变量的所有方程都是离散时间表达式，并且保证不激活事件之间的交叉函数，这个限制是必须的。】

【示例：

**model** Constants

**parameter** Real p1 = 1;

**constant** Real c1 = p1 + 2; // error， no constant expression

**parameter** Real p2 = p1 + 2; // fine

**end** Constants;

**model** Test

Constants c1(p1=3); // fine

Constants c2(p2=7); // fine， declaration equation can be modified

Boolean b;

Real x;

**equation**

b = noEvent(x > 1) // error， since b is a discrete-time expr. and

// noEvent(x > 1) is not a discrete-time expr.

**end** Test;

】

### 3.8.4 连续时间表达式

所有表达式（包括常数、参数和离散表达式）都是连续时间表达式。术语“非离散时间表达式”指那些不是常数、参数和离散表达式的表达式。

1. MATLAB是MathWorks注册商标。 [↑](#footnote-ref-1)
2. Mathematica是Wolfram Research公司注册商标。 [↑](#footnote-ref-2)