MPL's Primitive Structural Operators

In order to analyze and manipulate a mathematical expression, we must access its operators and operands. MPL uses three primitive operators to perform these tasks.

Definition 1 *The operator MPL*

$$Kind(u)$$

is defined by the following rules:

1. *If u is an atomic expression, $Kind(u)$ returns the type of expression (e.g.,* **integer**, **real**, *or* **symbol**).
2. *If u is a compound expression, $Kind(u)$ returns the operator at the root of the expression tree.*

Example 1

$$\begin{aligned}
Kind(x) &\rightarrow \textbf{symbol},\\
Kind(3) &\rightarrow \textbf{integer},\\
Kind(2.1) &\rightarrow \textbf{real},\\
Kind(\pi) &\rightarrow \textbf{symbol},\\
Kind(m*x+b) &\rightarrow +,\\
Kind((a+b)*\sin(x \wedge 2)) &\rightarrow *,\\
Kind((a/b) &\rightarrow *,\\
Kind(2/3) &\rightarrow \textbf{fraction},\\
Kind(\sin(x)) &\rightarrow \sin,\\
Kind(a=b) &\rightarrow =,\\
Kind(\{a,b,c,d\}) &\rightarrow \textbf{set},\\
Kind(x \textbf{ and } y) &\rightarrow \textbf{and},\\
Kind(x-x+2) &\rightarrow \textbf{integer}.
\end{aligned}$$

In the last example, the operand is simplified by automatic simplification to the integer 2.

Definition 2 *If u is a compound expression, the MPL operator*

$$Number\_of\_operands(u)$$

returns the number of operands of the main operator of u. If u is not a compound expression, then Number_of_operands returns the global symbol **Undefined**.

Example 2

$$\begin{aligned} \mathit{Number\_of\_operands}(m*x+b) &\rightarrow 2, \\ \mathit{Number\_of\_operands}(f(x,y)) &\rightarrow 2, \\ \mathit{Number\_of\_operands}(\{a,b,c,d\}) &\rightarrow 4, \\ \mathit{Number\_of\_operands}(n!) &\rightarrow 1, \\ \mathit{Number\_of\_operands}(x) &\rightarrow \textbf{Undefined}. \end{aligned}$$

In the last example, the input expression x is not a compound expression.

Definition 3 *If u is a compound expression, the MPL operator*

$$\mathit{Operand}(u,i)$$

returns the ith *operand of u. If u is not a compound expression or u does not have an* ith *operand, then Operand returns the global symbol* **Undefined**.

Example 3

$$\begin{aligned} \mathit{Operand}(m*x+b,2) &\rightarrow b, \\ \mathit{Operand}(x\wedge 2,1) &\rightarrow x, \\ \mathit{Operand}(\mathit{Operand}(m*x+b,1),2) &\rightarrow x, \\ \mathit{Operand}(\{a,b,c,d\},2) &\rightarrow b, \\ \mathit{Operand}(x-x,1) &\rightarrow \textbf{Undefined}, \\ \mathit{Operand}(2/(-3),2) &\rightarrow 3. \end{aligned}$$

The last two examples are based on the simplified form of the expression.

Keep in mind, because automatic simplification in a computer algebra system may apply the commutative law to reorder the operands in a sum or product, the *Operand* operator may obtain an unexpected result. For example, if $b+a$ is reordered to $a+b$, we obtain

$$\mathit{Operand}(b+a,\,2) \rightarrow b.$$

The operators *Kind*, *Number_of_operands*, and *Operand* are the three basic operations that are used to analyze and manipulate mathematical expressions, and most computer algebra systems have versions of these operators (see Figure 1).

MPL	Maple	Mathematica	MuPAD
Kind(u)	`whattype(u)` and `op(0,u)` for function names	`Head(u)`	`type(u)` and `op(u,0)` for undefined function names
Operand(u,i)	`op(i,u)`	`Part[u,i]` and `Numerator[u]` and `Denominator[u]` for fractions	`op(u,i)`
Number_of_operands(u)	`nops(u)`	`Length[u]`	`nops(u)`
Construct(f,L)	see Figure 2	`Apply[f,L]`	see Figure 3

Figure 1. The primitive MPL structural operators in Maple, Mathematica, and MuPAD. (Implementation: Maple (mws), Mathematica (nb), MuPAD (mnb).)

Construction of Expressions

In some instances, we need to construct an expression with a given operator and list of operands. The MPL operator *Construct* is used for this purpose.

Definition 4 *Let f be an operator ($+$, $*$, $=$, etc.) or a symbol, and let $L = [a, b, \ldots, c]$ be a list of expressions. The MPL operator*

$$Construct(f, L)$$

returns an expression with main operator f and operands $a, b, \ldots, c$.

Example 4

$$\begin{aligned} Construct(" + ", [a, b, c]) &\rightarrow a + b + c, \\ Construct(" * ", [a + b, c + d, e + f]) &\rightarrow (a + b) * (c + d) * (e + f), \\ Construct(g, [a, b, c]) &\rightarrow g(a, b, c), \end{aligned}$$

While Mathematica has an operator that constructs expressions (see Figure 1), Maple and MuPAD do not. However, in both of these languages, the operation can be simulated with a procedure (see Figures 2 and 3).

```
Construct := proc(f,L)
local g,s;
  if f = `!` then RETURN(op(L)!);
  elif member(f,{`and`,`or`}) then RETURN(convert(L,f))
  elif f = `not` then RETURN(not op(L))
  elif f = set then RETURN({op(L)})
  elif f = list then RETURN(L)
  else s := subsop(0=f,g(op(L))); RETURN(eval(s))
  fi
end:
```

Figure 2. A Maple procedure to implement MPL's *Construct* operator. (Implementation: Maple (txt).)

```
Construct := proc(f,L)
begin
  if f = _divide then return(op(L,1)/op(L,2))
  elif f = _subtract of f = _negate then return(op(L,1)-op(L,2))
  elif f = DOM_SET then return({op(L)})
  elif f = DOM_LIST then return(L)
  else return(f(op(L)))
  end_if
end_proc:
```

Figure 3. A MuPAD procedure to implement MPL's *Construct* operator. (Implementation: MuPAD (txt).)

Return to Chapter 1, page 8