MPL Functions and Procedures

Functions

In ordinary mathematical discourse, the statement, “let $f(x) = x^2 + 4$,” defines a computational scheme and does not perform a computation. A computation occurs when the function is invoked with a statement such as $f(2) \to 8$. In MPL, a function definition is used to mimic this operation.

In MPL, a *function definition* has the form

$$f(x_1, \ldots, x_l) \overset{\text{function}}{:=} u,$$

where f is the *function name*, $x_1, \ldots, x_l$ is a sequence of symbols called the *formal parameters*, and u is a mathematical expression. As with ordinary mathematical notation, a function is invoked with an expression of the form

$$f(a_1, \ldots, a_l), \tag{1}$$

where $a_1, \ldots, a_l$ is a sequence of mathematical expressions called the *actual parameters*. When this expression is evaluated, each a_i is evaluated and substituted for the corresponding x_i in u, and then u is evaluated, and the resulting expression is returned as the evaluated form of (1).

Example 1 Consider the function definition

$$f(x) \overset{\text{function}}{:=} x^2 + 4.$$

The function is invoked with an expression such as $f(2)$. When this statement is evaluated, the actual parameter 2 replaces formal parameter x in $x^2 + 4$, and $f(2) \to 8$.

Example 2 Consider the function definition

$$T(y, x) \overset{\text{function}}{:=} \mathit{Derivative}(y, x) + y.$$

The function is invoked with an expression such as $T(\sin(t) + t^2,\ t)$. When this statement is evaluated, the actual parameters $\sin(t) + t^2$ and t are substituted for the formal parameters y and x, and we obtain

$$T(\sin(t) + t^2,\ t) \to \cos(t) + 2\,t + \sin(t) + t^2.$$

In Figure 1 we give function definitions in Maple, Mathematica, and MuPAD that implement the MPL definitions in Examples 1 and 2.

```
f := x -> x^2+4;

T := (y,x) -> diff(y,x)+y;
```

(a) Maple.

```
f[x_] := x^2 + 4

T[y_, x_] := D[y, x] + y
```

(b) Mathematica.

```
f := x -> x^2+4;

T := (y,x) -> diff(y,x)+y;
```

(c) MuPAD.

Figure 1. Function definitions in Maple, Mathematica, and MuPAD that correspond to the MPL definitions in Examples 1 and 2. (Implementation: Maple (mws), Mathematica (nb), MuPAD (mnb).)

Procedure Definitions

MPL procedures extend the function concept to *mathematical operators* that are defined by a sequence of statements. The general form of a procedure is given in Figure 2. The first line of the procedure gives the *procedure name* and a sequence of formal parameters. The **Input** section contains each of the formal parameters x_i along with a brief description of the type of expression that replaces it when the procedure is invoked. MPL procedures always return a mathematical expression as output, and the **Output** section contains a brief description of this expression.

The **Local Variables** section contains a sequence of *local variables* that are known and used only by the procedure. The formal parameters and the local variables make up the *local environment* of a procedure. In a real CAS, each time a procedure is invoked, the variables in this environment are given storage locations in the computer's memory, and when the procedure terminates, these locations are released back to the system.

The statements between the delimiters **Begin** and **End** represent the *body* or the executable statements of the procedure. Each statement S_j is either a mathematical expression, an assignment statement, or a decision

```
Procedure  $P(x_1, \ldots, x_l)$;
Input
    $x_1$ : description of input to $x_1$;
        ⋮
    $x_l$ : description of input to $x_l$;
Output
    description of output;
Local Variables
    $v_1, \ldots, v_m$;
Begin
    $S_1$;
    $S_2$;
     ⋮
    $S_{n-1}$;
    $S_n$
End
```

Figure 2. The general form of an MPL procedure.

or iteration structure both of which are defined later in this section.

A procedure is invoked like a function with an expression of the form (1). When the procedure is invoked, each actual parameter a_i is evaluated and then substituted for the corresponding formal parameter x_i, after which each statement S_j in the body is evaluated. In most cases, at least one of the S_j includes a *return statement* that has the form

$$\mathit{Return}(u),$$

u is a mathematical expression. When this statement is encountered, three actions occur: first, the procedure immediately terminates; second, the evaluated form of u is returned as the evaluated form of (1); and finally, control is transferred back to the statement that invoked the procedure. If a *Return* statement is not included, the actions are similar, but now the evaluated form of the last statement S_n is returned by the procedure. We always include a *Return* statement to emphasize what is returned by the procedure.

Example 3 We illustrate this concept by defining a procedure that obtains the equation of a tangent line to a function $f(x)$ at the point $x = a$. Recall that the expression for the tangent line is given by

$$\frac{df}{dx}(a)(x-a) + f(a). \tag{2}$$

```
Procedure   $Tangent\_line(f, x, a)$;
Input
    $f$ : an algebraic expression (formula for a mathematical function);
    $x$ : a symbol (independent variable);
    $a$ : an algebraic expression (point of tangency);
Output
    an algebraic expression that is the formula for the tangent line;
Local Variables
    $deriv, m, line$;
Begin
1   $deriv := Derivative(f, x)$;
2   $m := Substitute(deriv, x = a)$;
3   $line := Algebraic\_expand(m * (x - a) + Substitute(f, x = a))$;
4   $Return(line)$
End
```

Figure 3. An MPL procedure that obtains the formula for a tangent line. (Implementation: Maple (txt), Mathematica (txt), MuPAD (txt).)

The procedure definition in Figure 3 is an algorithmic view of what is done to obtain this expression in expanded form. We invoke the procedure with an expression such as

$$Tangent\_line(1/z, z, 3). \tag{3}$$

When this expression is evaluated, the three actual parameters $1/z$, z, and 3 are substituted for the corresponding formal parameters f, x, and a, and then the statements in the procedure are evaluated:

$$\begin{aligned} &deriv := Derivative(1/z,\, z) \to -1/z^2, \\ &m := Substitute(-1/z^2,\, z = 3) \to -1/9, \\ &line := Algebraic\_expand\big((-1/9) * (z - 3) + Substitute(1/z,\, z = 3)\big) \\ &\qquad\qquad \to (-1/9)\, z + 2/3. \end{aligned}$$

Therefore

$$Tangent\_line(1/z, z, 3) \to (-1/9)\, z + 2/3.$$

When we invoked the procedure in Expression (3) for clarity we intentionally chose names for mathematical symbols that were different from the formal parameter names of the procedure. There is no reason, however, to restrict the actual parameters in this way. For example, the procedure can also be invoked with

$$Tangent\_line(1/x, x, 3) \to (-1/9)\, x + 2/3. \tag{4}$$

Keep in mind, however, that the actual parameter x in Statement (4) and the formal parameter x in the procedure declaration

$$\textbf{Procedure}\ \mathit{Tangent\_line}(f, x, a) \tag{5}$$

are different symbols even though they have the same name. When Statement (4) is evaluated, each actual parameter is substituted for the corresponding formal parameter which means that f is replaced by $1/x$, the formal parameter x in (5) by the actual parameter x in (4), and a by 3. Therefore, the differentiation at line 1 is

$$\mathit{Derivative}(1/x, x) \to -1/x^2,$$

where the x that appears here is the one in (4). Similar comments apply to the other statements in the procedure.

Maple, Mathematica, and MuPAD provide procedures that operate as described above. In Figures 4 and 5 we give implementations of *Tangent_line* in these languages.

Global Symbols

A symbol that appears in a function or a procedure that is not a formal parameter or a local variable is called a *global symbol.* Global symbols, which are accessible to both the interactive mode and other functions and procedures, provide another way to pass data to and from a procedure without using the formal parameters or a *Return* statement.

For a simple example, consider a modification of the *Tangent_line* procedure in which the variable *deriv* has been removed from the local section and therefore is considered global. In this case, after evaluating Statement (3) the global variable *deriv* has the value $-1/z^2$ which can now be used by other functions, procedures, or the interactive mode.

Local Variables and Formal Parameters

In order to promote a programming style that works for all languages we adopt the following conventions for the use of local variables and formal parameters in a procedure:

- *In MPL procedures, an unassigned local variable cannot appear as a symbol in a mathematical expression.* In other words, in MPL procedures local variables can only act as programming variables and must be assigned before they appear in a mathematical expression. In situations where a procedure requires a local mathematical symbol, we either pass the symbol through the parameter list or use a global symbol.

```
Tangent_line := proc(f,x,a)

#Input
# f: an algebraic expression (formula for a mathematical function)
# x: a symbol (independent variable)
# a: an algebraic expression (point of tangency)
#Output
# an algebraic expression that is the formula for the tangent line
local
  deriv,m,line;
deriv := diff(f,x);
m := subs(x=a,deriv);
line := expand(m*(x-a)+subs(x=a,f));
RETURN(line)
end:
```

(a) Maple.

```
TangentLine[f_,x_,a_] := Module[
(*Input
  f: an algebraic expression (formula for a mathematical function)
  x: a symbol (independent variable)
  a: an algebraic expression (point of tangency)
Output
  an algebraic expression that is the formula for the tangent line
Local*)
  {deriv,m,line},
deriv  = D[f,x];
m = ReplaceAll[deriv,x->a];
line = Expand[m*(x-a)+ ReplaceAll[f,x->a]];
Return[line]
]
```

(b) Mathematica.

Figure 4. Implementations of the MPL procedure in Figure 3 in Maple and Mathematica. (Implementation: Maple (txt), Mathematica (txt).)

- *Formal parameters in MPL procedures are used only to transmit data into a procedure and not as local variables or to return data from a procedure.* In conventional programming languages, a procedure's

```
Tangent_line := proc(f,x,a)
/*
Input
  f: an algebraic expression (formula for a mathematical function)
  x: a symbol (independent variable)
  a: an algebraic expression (point of tangency)
Output
  an algebraic expression that is the formula for the tangent line
*/
local
  deriv,m,line;
begin
deriv := diff(f,x);
m := subs(deriv,x=a);
line := expand(m*(x-a)+subs(f,x=a));
return(line)
end_proc:
```

Figure 5. A MuPAD implementation of the MPL procedure in Figure 3. (Implementation: MuPAD (txt).)

formal parameters can be used both to transmit data to and from a procedure and as local variables. The situation with CAS languages is more involved, however, because the actual parameters in a procedure call can be mathematical expressions as well as variables. Because of this, the language mechanism that is used to bind the formal parameters with the actual parameters can be rather involved and can vary from system to system. For this reason, the use of formal parameters for anything but the transmission of data into a procedure is system dependent. When we need to return more than one expression from a procedure, we return a list of expressions.

Return to Chapter 1, page 3