

THE EXTENSION FACILITIES OF IMP

Edgar T. Irons. Institute for Defense Analysis, 100 Prospect Avenue, Princeton, New Jersey 08540. 1969 May 13.

The objective for the extension feature of the programming language Imp is to provide an extension capability powerful enough to describe in detail the construction of a useful compiler for common programming languages, yet simple enough to allow unsophisticated programmers to make useful modifications to the language. Particular emphasis is given to keeping the concepts and constructions concise and few in number.

The base language provides for the expression of algorithms using formulas similar to those of common algorithmic languages but attributing a value to each such formula so that the concepts of expression and statement are merged. Thus, the usual expressions, like $A+B*C$, are used to denote arithmetic values, but $A < B$, $A \leftarrow B$, $A \Rightarrow B$, $A : B$, and $A ; B$ are also expressions denoting, respectively, the truth of the condition $A < B$ (1 or 0), assigning B to A (value B), conditional evaluation of B (value B if $A \neq 0$, otherwise value A and B is not evaluated), tagging of B with the name A (value B), and sequential evaluation of A and then B (value B). Thus a program for computing the sum S of vector A with N values is shown below. The extension facility allows new expressions

```
S ← I ← 0;
T: S ← S + A[I];
(I ← I + 1) ≠ N ⇒ go to T;
```

to be introduced with the use of the notation $A \equiv B$, where A is a construction specifying the syntactic structure of the new item and B gives the value of an instance of the new item. Thus, $(A) = (B) \text{ else}$

$(C) \equiv 'A \Rightarrow (T \leftarrow B; \text{go to } L); T \leftarrow C; L: T'$ defines a new expressions for conditional evaluation whose value is B if $A \neq 0$, otherwise, C . Parentheses are used on the left to denote "parameter expressions" and other symbols to denote their required occurrence in an instance of the new item. Thus, with the above extension, writing $X \Rightarrow 3 \text{ else } N$ has the same effect as writing $X \Rightarrow (T \leftarrow 3; \text{go to } L); T \leftarrow N; L: T$.

In this most elementary form of the extension, the syntactic type of the parameter has been filled in automatically as "expression." However, the syntactic type of the parameters may be given explicitly by using (A, NAME) instead of (A) to specify that the syntactic type of A is NAME . Appending $X \equiv$ to the left of the construction specifies the resultant syntactic type of the extension to be X . Assuming the syntactic type NAME exists already within the system, this fully expressed form of extension can be used to specify a list of NAMEs (NAL) as shown in the example.

```
NAL ≡ (A, NAME) ≡ 'A'
NAL ≡ (A, NAL), (B, NAME) ≡
((K ← J) for J thru cont A;
scr K ← B; A)
```

```

X , Y , Z
NAL
NAL
NAL
NAL
```

```

NAL
NAL - Z
NAL - Y
X
```

Such specification yields the parse and parse tree shown to the right. In the second extension, by omitting the quote marks, we have specified that the right side is a "computation" to be carried out while compiling. Such computations are given in Imp, in this case using the list functions: cont A (the node below A), scr A (the node to the right of A), and A for B thru C (evaluate A for variable B pointing successively to C and its scrs).

The computation specified here is executed after computations on subnodes and its effect is to reduce the parse tree shown above in the following steps:

```

NAL
|
NAL - Z
|
NAL - Y
|
X
```

```

NAL
|
NAL - Z
|
X - Y
```

```

NAL
|
X - Y - Z
```

To illustrate a significant use of computed extensions we will now describe the implementation of a vector operator system allowing most operators to automatically become vector operators. By example, we wish to denote $\sum A_i * B_i$ for $i = 1..n$, where n is the least dimension of A or B , by $+ A * B$. In general, we wish to be able to use any binary operator in place of the $+$ and any arbitrary expression in place of the $A * B$.

We introduce a rudimentary extension for vector declarations which at compile time stores the expression for the length of a vector A in a compile time vector named `LENGTH`: $(A) \text{ is } (B) \text{ long} \equiv \text{LENGTH} \leftarrow \text{valu } A \leftarrow B$. For each binary operator which we will allow to compress a vector expression we introduce an extension like `BINOP` $\equiv + \equiv 'P \leftarrow 0;$ $(P \leftarrow P + Q) \text{ for } I \text{ to } N;$ P' to give the form for the repeated evaluation of the expression, which we will put in for Q . The computed extension for compression is:

$(A, \text{BINOP})[(B)] \equiv (\text{TOP} \leftarrow \text{REP}(B, 100000)) A \text{ with } 'Q' \leftarrow B, 'N' \leftarrow \text{TOP}$

Evaluation of the subroutine, `REP`, replaces each occurrence of a vector variable V in B with its indexed expression $V \downarrow I \uparrow$, and assigns to `TOP` an expression which computes the upper limit for I (maximum 100000) in the iteration of the expression. The result of the computed extension is the expression associated with the `BINOP` A , but with each occurrence of Q in A replaced with the modified expression B , and with N replaced by an expression for the upper limit.

The function `REP` is recursively defined by:

recursive `REP` on L, T is
 $((\text{cont } L) \neq 0 \Rightarrow \text{REP}(\text{cont } L, T);$
 else
 $(S \leftarrow \text{NUMERIC}(\text{LENGTH} \downarrow \text{valu } L \uparrow);$
 $S < T \Rightarrow (T \leftarrow S; Y \leftarrow \text{LENGTH} \downarrow \text{valu } L \uparrow);$
 $K \leftarrow 'A \downarrow I \uparrow' \text{ with } 'A' \leftarrow L;$
 $\text{valu } L \leftarrow \text{valu } K;$
 $\text{cont } L \leftarrow \text{cont } K);$
 $(\text{scr } L) \neq 0 \Rightarrow \text{REP}(\text{scr } L, T); Y);$

With these extensions, writing:

$X \text{ is } 3 \text{ long}; Y \text{ is } 10 \text{ long};$
 $+ [X * (Y + X)]$

is the equivalent of writing

$X \text{ is } 3 \text{ long}; Y \text{ is } 10 \text{ long};$
 $(P \leftarrow 0;$
 $(P \leftarrow P + X \downarrow I \uparrow * (Y \downarrow I \uparrow + X \downarrow I \uparrow))$
 $\text{for } I \text{ to } 3; P)$