1. THE PROBLEM

Currently there is much interest in the design of programming languages which allow:

1. parametrized types, the construction of new types from old [1];
2. operations that are polymorphic in that they have type parameters (e.g., an operation for sorting a vector of any size);
3. limiting the operations that can be applied to a type [2];
4. types that are abstract in that the type representation cannot be used outside the type declaration [3];

The first two of these raise the theoretical problem:

When can an actual parameter of type T be substituted for a formal parameter of type T'?

This is similar to, but not the same as the theoretical problem for abstract data types:

When can a type T be represented as another type T'?

In this paper we discuss these two theoretical problems, not the design of new programming languages. The literature on the design problem is extensive; those interested should begin by looking at such languages as CLU, ALPHARD, MESA, EUCLID [4] and pondering on the TINMAN requirements [5]. In the remainder of this section we give an example of (1)-(4); in section 2 we emphasize the abstractness of all type declarations; in section 3 we give a simple, precise definition of data types and relate it to other definitions in the literature; in section 4 we propose a solution of the two theoretical problems.

The usual example of a parametrized type is STACK(EL), with stacks of values of type EL as its values. We can define polymorphic operations for such specifications as:

\[
\begin{align*}
\text{PUSH} &: \text{STACK(EL)} \times \text{EL} \rightarrow \text{STACK(EL)} \\
\text{NEW} &: \text{STACK(EL)} \\
\text{POP} &: \text{STACK(EL)} \rightarrow \text{STACK(EL)} \\
\text{TOP} &: \text{STACK(EL)} \rightarrow \text{EL}
\end{align*}
\]

in many existing programming languages. In some of these languages we can limit the permissible operations on values of STACK(EL) to PUSH, NEW, POP and TOP.
Once a limit has been put on the operations on values of STACK(EL), there may be many useful representations of this parametrized type.

The concept of a parametrized type should be distinguished from the construction of new types from types and constructors that are provided ab initio by the programming language. In PASCAL one can write

```
TYPE ENTRY = RECORD
    identifier: ALFA;
    attribute : INTEGER;
END
```

and there are no parameters in this construction of a new type from the primitive types, ALFA and INTEGER. Suppose a language allows both the type ENTRY and the parametrized type STACK(EL) with operations PUSH, NEW, POP and TOP. Then the language may well allow the declaration like:

```
TYPE SymbolTable = STACK(STACK(ENTRY))
```

and the definition of operations for such specifications as

- BlockEntry, BlockExit: SymbolTable \rightarrow SymbolTable
- Initialize: SymbolTable
- Extend: SymbolTable \times ENTRY \rightarrow SymbolTable
- Find, Offset, BlockNumber: SymbolTable \times ENTRY \rightarrow INTEGER.

Presumably the language will also allow one to limit the operations on values of SymbolTable to the seven specified above. If so, there may be many useful representations of the type SymbolTable - some using different representations of stacks, and some not even using stacks.

![Values of the abstract type SymbolTable](image)

### Figure 1. Values of the abstract type SymbolTable

#### 2. ABSTRACT TYPES

All types are "abstract" in the sense that a programmer never knows what the values of a type really are. Suppose a language allows the definition of a type

```
VECTOR = ARRAY [low bound, high bound] OF REAL.
```

The programmer can read or write values into a variable v of type VECTOR by constructions like