DEWAR79
Dewar, Robert B. K., Arthur Grant, Ssu-Cheng Liu, Jacob T. Schwartz, and Edmond Schonberg. "Programming by Refinement, as Exemplified by the SETL Representation Sublanguage", ACM Transactions on Programming Languages and Systems 1, 1(1979), 27-49. An abstract algorithm expressed as a SETL program may be pleasing to behold and "efficient" in its rated time complexity, and yet be disappointingly slow when implemented on a real machine. This can happen, for instance, when a SETL compiler picks a representation for some set or map that is not as suitable as the one which a prescient programmer would have used, either because it doesn't have enough information to take advantage of restrictions that the programmer knows about but never makes explicit, or simply becuase the heuristics it used to "guess" the best representation failed to do so. By inserting declarations from the SETL Representation Sublanguage into an otherwise "finished" SETL program, the programmer can usually induce the compiler to choose data structures that improve the running time significantly. Ideally, this doesn't involve any changes to the existing text whatsoever. For example, a declaration such as

```
base b
```

defines a "base set" (or simply "base") b which is not a variable of the SETL program, but a Sublanguage variable. The virtue of such a definition is that variables of the SETL program can be described relative to the base and therefore relative to each other. Given a base b, then, the declarations

```
x: elmt b;
f: local smap (elmt b)
```

identify x (a variable that is actually used in the SETL program) as an element of b, and f (another SETL variable) as a single-valued map over a domain whose elements are also all in b. There declarations provide a means whereby the compiler can do extra type-checking (vanilla SETL is "weakly typed"). Here, it could make sure that all references to f are consistent with f's definition as a single-valued map. The sublanguage also allows the compiler to change what might have been a hash table lookup for f(x) into little more than a "load with fixed offset" machine instruction. The local keyword in the declaration of f, in fact, tells the compiler that the values of f (i.e., the range elements) can be stored right inside the "element blocks" of b, at fixed offsets from corresponding domain elements. If the keyword had been indexed, the range of f would still be almost as efficiently stored for f(x) references, but would also be readily available for other kinds of access. Maps and elements are not the only objects that can be described using the Representation Sublanguage, of course. Sets, including subset relationships where applicable, tuples, and all the atomic types (such as integers and strings) are fair game. Definitions can be grouped and nested, allowing the programmer to be as specific about data structures as he would have had to be in a lower-level language (e.g., using "record" data types). In the humble opinion of the reviewer, the foregoing is a much more readable and succinct introduction to the Representation Sublanguage than the one contained in the paper, although it depends entirely upon it for raw material. Be that as it may, the paper is the most complete tutorial available in published form, and should be consulted for "the expanded treatment". The authors deserve credit for raising the important point that most of a program's "refinement" ought to be carried out in abstract terms, with details of specific "computer-oriented" data structures ignored as long as possible. A longer discussion of this subject would have been welcome, especially as "Programming by Refinement..." is the title. In fact, the bulk of the paper is about the Representation Sublanguage. For readers interested in a very tidy summary of the SETL language itself, a brief overview near the beginning covers most of the key features. The last five pages of the paper are devoted to a tantalizing vision of what Automatic Data Structure Selection could be like, and how it might work. This subject has since been taken up in another article [1].

References Schonberg, Edmond, Jacob T. Schwartz, and Micha Sharir, "An Automatic Technique for Selection of Data Representations in SETL Programs", ACM Transactions on Programming Languages and Systems 3,2(1981),126-143.