SETL Newsletter 87 November 15,1972

Workplan for the next phases J. Schwartz

of the SETL project.

This newsletter will outline the next main phases of our work, along lines which it is hoped are reasonable. The first phase will be fleshed out with a fair amount of detail; subsequant phases considerably less so.

Phase 1. Development of a faster SETLB system; enhancement of same.

Phase 2. New SETL syntax installed; associated semantic changes.

Phase 3. Development of SETL optimiser and data structure elaboration language.

Phase 1 in detail: Our essential strategy during phase 1 will be to use the logic, and much of the text, of the BALM system, adapting it however to drive Hank Warren's new run time library (SRTL) The basic tactic will be to use a BALM translator very much like the present translator, but to have this link to the SRTL routines for the execution of prinitives. This will permit systematic transition from the present BALM-written run-time library (BALMSETL) to the much faster new SRTL. Subphases will probably be as follows:

Phase 1.0: Modify the BALM translator to generate inline code which handles SRTL data formats for the BALM semantic types (which will in large part become identical with corresponding SETL semantic types), rather than the present BALM-BALMSETL data formats.

This will involve replacing the present BALM garbage collector with the SRTL garbage collector(which is already debugged)and redoing in LITTLE a relatively small number of offline BALM primitive routines, presently written in a mixture of FORTRAN and assembly language. This will give us a functioning BALM system, at essentially the present level of efficiency, compatible however with the SRTL library. This system will serve as a development and test-generation matrix during the remainder of phase 1.

Phase 1.1. Progressively install SRTL routines for the important SETL primitives. If necessary, we can simply install one Set-Theoretic primitive after another, testing them as we go. However, if an appropriate technical plan for doing so can be devised, it might be better to interface between SRTL and the presently existing BALMSETL routines, progressively replacing BALMSETL library code as one goes. (The problem which must be faced along this line is that of interfacing with the data-object representation used in the run-time library.) This phase would then culminate in what is essentially the BALM/SRTL system for which we will aim. At first, however, the system might be cluttered with removable bits of interfacing code.

Phase 1.2. By removal of all extraneous bits of interfacing code we would immediately produce the BALM/SRTL system which is our find phase 1 goal.

Additional ongoing work during this phase will be: human factors and syntactic improvements, work toward an interactive system, optimisation of LITTLE, development of a big library of SETL algorithms. Proposed syntax improvements during this phase: iftree statement installed, regularisation of multiple assignment, local rather than global as default for variables, possible elimination of 'do; - compute;' in SETLB.

Improvements in BALM: introduction of modified BEGIN-END having no implication for labels, to allow elimination of the irritating label-related restrictions which now affect us.

During this phase we will also rewrite the SETLB translator in SETL, possibly emphasising use of the BALM operations, and possibly adding a few efficiency-vital primitives (such as a lexical scan a top-down parse interpreter primitive, etc.).

An improved method allowing the easy installation of LITTLE-written primitives should also be worked out. This should package the following action in a convenient form: compile a LITTLE subroutine or subroutines representing a primitive, apply the system loader to it if necessary, bring it into the SETL memory area and set up all links needed to make it available as a primitive.

A 360/LITTLE and a 360/Translator will also be prepared.

We may also want to think some about the use of secondary memory thru SETL, i.e. , its use for data base experiments.

Phase 2. in general outline: Phase 2 will be characterised by the development of a new SETL front end. The principal facilities to be provided by this front end are as follows: Systematic namescoping capable of supporting large libraries of routines; user-definable object types with operations applying in an object-kind dependent way; modification of argument values from within subroutines and systematic implementation of the 'sinister call' facility described in a previous newsletter.

During phase 2 of our work we should profit substantially from the availabilty of a SETLB system fast enough to allow its use for algorithmic experimentation on a fairly wide scale.

we should attempt to stick to the development method which we preach: initial development of all algorithms in SETL, annotation in the DSEL to define an implementation precisely, followed by just enough lower-level coding (in LITTLE) to reach acceptable efficiency. Lower-level code packages ought normally be primitives which can be interfaced to existing SETL operations.