SETL Newsletter #41 June 2, 1971

<u>Additional planning details for the current and next phase of SETL implementation</u> J. T. Schwartz

This newsletter brings Newsletter 15 of Feb. 19, 1971 up to date.

<u>Status Report</u>: A first version of the BALMSETL parsing system is now close to working, though very inefficiently (approximately 30,000 times slower than FORTRAN).

1. <u>Improved efficiency</u>: The new BALM implementation should be approximately twice as fast as the old, and, with compilation of the presently interpreted BALM-machine code, approximately 20 times as fast. Improved BALMSETL data structures should provide at least a 5-fold efficiency increase. Thus improvements currently in prospect should give an efficiency increase approximately 100-fold, enabling the anticipated level of experimental work to continue.

2. <u>Further efficiency improvements</u>. Data structures directly and efficiently representing sets, and supporting the fundamental SETL operations, including $x \in a$, a <u>eq</u> b, a <u>with</u> x, a <u>less</u> x, $\ni a$, #a, and $\forall x \in a$ will be designed, and incorporated into the BALM machine and the BALM-machine 6600 compiler as BALM-machine level operations. These data structures and operation implementations must make <u>membership</u> and <u>equality testing</u> efficient in the most commonly occurring cases. Efficient <u>iterators</u> should also be possible. Access-paths to tuples which are set members must yield an efficient implementation of the SETL indexed retrieval and indexed assignment operations. Whatever hopefully very slight modifications to the BALM compiler are necessary to give access to these operations will be made. This will allow the construction of a considerably improved BALMSETL interpretation package, which may be designated as BALMSETL2. The existing parsing programs will be revised to be compatible with BALMSETL2 and to use its features efficiently.

Either BALMSETL2, its pre-compiled version (internal BALM syntax trees), or its BALM-machine version (for the extended BALM machine) will be the target language for the preliminary SETL (BOOTSETL) compiler.

Sets will probably be represented by collections of BALM vectors (either fixed or growing) with access being via hash.

In BOOTSETL (and here lies its main difference from true SETL) reference count keeping and automatic copying of sets, as logically implied by SETL, will not be supported. Thus in BOOTSETL there will exist an explicit *copy set* operation, which the programmer may on occasion have to use (though as a matter of fact most algorithms never require its use).

3. *Language specifications for BOOTSETL*. In addition to and modification of the language as specified in the basic SETL document, the following features will be provided.

a. Square brackets within expressions (Newsletter 33, p. 1)

b. New character and bit-string syntax (Newsletter 34, p. 1,2)

c. Expanded object type function (" 2)

d. Real quantities (" 3)

e. Exponentiation (" 3)

f. Tuples. As described in newsletter 34, p. 3,4, items a through i but not j, except that <*just* a,b> will not be the same as <a,b>, nor may tupl(m,n) be used on the left-hand side of an assignment statement.

g. Revised conventions for *is* (Newsletter 34, p. 5)

h. Built-in union, etc. operators (" ")
except that *u* will also remain valid for union.

i. Nonmembership (" 6)

j. Compound operators for while-iterations(" ")

k. While-when iterations (" ")

l. Then-if forms (" ")

m. At-blocks (" 7)

n. 'Local' subroutines (" ")

o. 'Inverted' form for subroutines and functions (" 8)

p. Modified macro-conventions (" 9-10)

q. Sinister calls as described in Newsletter 30, except that (cf. Newsletter 30, page.17, e;) the code corresponding to a basic SETL operation which is not a retrieval will be a no-op rather than an error call.

r. Iff statement (see Newsletter 35)

Programmer-definable object types will not be provided, nor will any of the proposed efficiency-enhancing 'elaborations' of SETL be provided.

A SETL grammar will be written for the parser package first as a syntax checker, but later supplied with the generative actions needed to produce target code.

4. Optimization: Work on global optimization will go forward and optimization algorithms will be collected and documented. It will probably not be possible to include any global optimizations in the BOOTSETL compiler, but various local optimizations may be incorporated in this compiler's code generation routines.

5. Macroprocessor: The presently specified macro forms (cf. especially newsletter 34, p. 10) will be implemented by a macro-processor which acts prior to the preparse routine, but after lexical analysis. A design for a more powerful syntax macro processor having the flavor of the BALM 'means' processor, but providing for conditional and iterative expansions, will be worked up, and added to the macro-processor routine, either concurrently with the main implementation work, or somewhat later.

6. Read routine: A formatted read routine based upon the parsing programs will be produced.

7. Parallel subsidiary projects. A PL/1 version of the parser package will be produced, and from this a LITTLE version of the parser page. In addition, a Honeywell 516 version of the same package will be produced. Postparse grammars for FORTRAN, COBOL, LITTLE, and BASIC will be produced.

8. Planning for the full SETL implementation.

In the meanwhile, fuller data structures, capable of providing support for the automatic copying employed in true SETL, will have to be designed. The subsequent implementation of true SETL will then aim to include

a. automatic copying in accordance with the true SETL specifications

b. debugging features

c. global optimization, and improved local optimization

d. programmer-defined object types

e. an elaboration language for enhancing efficiency

f. possibly, an improved language-extension method, perhaps based on nodal span parsing.

g. 'mechanism linkages' between subroutines

h. linkage to a lower level language

plus whatever miscellaneous improvements develop in the meanwhile.

Subsequent work may aim at an interactive console system.