Experimental Implementation of SETL

D.Ya.Levin (Novosibirsk)

An activity connected with construction and implementation of the programming languages based on set-theoretic dictions probably will take the form of a programming language of the next generation. SETL project of New-York University is the first and very good example of such activity; it attracted our attention thanks to the works and comminucations of Prof. Schwartz (e.g.[1]). Now it is about 3 years that our small group works in this field. In particular one of the results of this activity is the experemental implementation of the "Siberian version" of SETL (*, which is a subject of this report. The input language and implementation itself are the attempts to adaptate the principles of the canonical SETL to our computer and man-power resources and the tastes as well, which have been formed rather under ALGOL than FORTRAN influence. One of the points which was in mind at the beginning of our work was to remain for user a possibility to lower a level of very high level language.

1. Brief specification of SETL (C).

1.1. The language primitives are as follows: logical constants, integers, strings, bitstrings and "undefined value" (denoted "?"). The usual arithmetics and relation operation are defined for integers. Strings and bitstrings can be described by denotations, e.g.

'EXAMPLE + OF + A + STRING', .01101.

(* called SETL (C).

or by statement, which describe string of lenghth L with undefined components:

S = STRING (L), B = BSTRING (L).

SETL (C) is a language of expressions, so that L is any expression producing positive value, e.g.

STRING (PRINT (2 + X)).

Symbol "=" corresponds to usual assignment and shows that a name before "=" begins to denote value or process in the right-hand part of the construction. String components are accesible by means of "[ ]" operation, so that S[K] and B[K] are an one-symbol string and a logical value correspondingly. A part of a string is extracted by "÷" operation, e.g. C[3÷5] is a string 'B+C', if C is 'A=B+C'. Assignments such as S[K]= SYMBOL, B[K]= LOG, S[K÷L]= SLK,... are also used. For a string S, #S denotes its lenghth, and by the way S[#S+K] has "?" as a value if $k>0$. If S contains SYMBOL than SYMBOL $\in$ S is TRUE; if B contains LOG than LOG $\in$ B is TRUE. Strings and bitstrings can be concatenated (by "+"); bitstrings are provided with bit operations ($\neg$, v,&, $\oplus$).

1.2. The SETL(C) data aggregates are sets and tuples. The set is specified by enumerating of its elements:

{1,2,'S',. 101., {7}, {7,{8}} }

or by means of "set-former"

{X, X $\in$ M | P(X)} :

the set of all such elements from M, for which a logical finction P(X) has a value TRUE. "{ }" denotes an empty set. If M is an interval of integers ([A,B],]A,B],...,[1,B],...) some other kinds of set-former are available:

{X, A $\leq$ X $\leq$ B | P(X) } , {X, A $<$ X $\leq$ B | P(X)} ,...
{X, X $\leq$ B | P(X) } .

All usual set-theoretic operations (union (+), intersection (*), difference (-),... and relations ($\in, \subset, \subseteq$) are defined. If M is a set, M WITH A means M+{A}, M LESS B means M-{B} and ANY M denotes an unpredictible element of M.

Membership ($\in$) and might (#) operations have the usual sense:

(2 + 5) $\in$ {6,8-1} is TRUE, # {1,2,3} is 3,

{ } is 0, # {1,1,2,2,2-1} is 2.

Tuples can be considered as usual vectors, which in particular are defined by enumeration of its components; e.g.

$$T = \langle 1,2,3, \{1\}, \text{'A'}, 1 \rangle .$$

It must be, however, noticed that the above assignment defines not only the first 6 components of T: the value of T[K], k>6, is "?". For example, T[100] = {1,2} changes the value of T's 100th component from "?" to {1,2}.

Sets of pairs (i.e. tuples oflenghth 2) have the special status: they can be treated as graphics of functions; e.g.

$$\Phi = \{\langle \text{'A'}, 32 \rangle, \langle \text{'B'}, 34 \rangle, \langle \text{'C'}, 49 \rangle\}$$

defines, by the way, a function Φ, such that Φ('A') is equal to 32, Φ('B') - 34 etc. Φ can be modified as a usual set by means of expression

$$\Phi = (\Phi \text{ LESS } \langle X, \Phi(X) \rangle) \text{ WITH } \langle X,Y \rangle$$

or by special kind of assignment: $\Phi(X) = Y$ (means "let Φ(X) be equal to Y"). For every X, Φ(X) is always defined, for example, as an undefined value.

1.3. Logical expressions can be composed with $\neg$, v, & and quontifiers $\forall$ and $\exists$:

$\forall X \in M$: P(X) means "for all X from M holds P(X)";

$\exists x \in M$ P(x) means "there exists X in M, such that holds P(X)".

In the above expressions M is not only a set, but probably a tuple, a string or a bitstring; P is any expression, so that complex quantifiered expressions are admissible.

An important case - M is an interval of integers - has a special syntactical form:

$\forall$ X, $A \leq X \leq B$: P(X),

$\forall$ $X \leq B$: P(X).

1.4. Expression of the form

$(X \in M \mid P(X))$ or $(A \leq X \leq B \mid P(X))$

define some element X of M (or some integer from [A,B]) such that P(X) holds.

For example,

$S[K \div (K \leq I \leq \#S) \mid S[I] \in DLM]$

is a substring of S, which begins with S[K] and ends with the first occurence of element from DLM.

1.5. A program consists of sentences, connected with ";" ; a sentence is an assignment, procedure specification, macrodefinition, operation specification, a procedure call, a loop, a conditional of usual form, an object specification or codes, written in other lahguages (see below).

There are two forms of a loop: besides the usual "while-loop", SETL has "for - all - loop":

$\forall X \in M$ :: A(X) - for all X from M perform action A(X), and its modifications:

$\forall X \in M \mid P(X)$ :: A(X),

$\forall X \leq B$ :: A(X) etc.

Macrodefinition has a form of

"newexpression" MEANS "expression",

where "newexpression" is any syntactically correct expression and "expression" is any semantically correct expression, probably, containing special functions to be executed during compilation.

An object specification looks like Algol type description. It contains information which is suggested to be utilized by the system to produce more effective code or to perform computation in more effective way. Moreover, the system can be asked to test every specified object in the given program to found contradictions between its use and specification. The general form of a specification is

prop X = P(X),

where P is any logical expression. There are primitive specificators, e.g. "list": used in prop X = list X it makes the system to choose list representation for a set X; another example is prop X = dif X : it means that for every X=Y with A, A is guaranteered not to be an element of X before the mentioned assignment is executed. "Prop M = 23 ∈ M" is also an example of object specification.

2. Two examples of SETL(C) programs.

2.1. Let us consider a very simple data-base, which contains questionnaires of employeers of the institution. Naturally, they can be represented by a set of tuples, each tuple being an image of a separate questionnair. A tuple has, for example, the following structure:

```
   (surname, post, age,maritalstate,children,salary,department).
/* Here the program begins : */
Q = {<'ivanov', 'director', 42, mar-d, 3, 550, ? >,
     <'petrova','engineer', 28, mar-d, 1, 120, 5 >,
     /* . . . . . . . . . . . . . . . . . . . . . . . . */
     <'sidorov','turner',   38, mar-d, 2, 200, 7> };
   /* several useful macrodefinitions : */
      surname means x[1]; post means  x[2]; /* etc */
      deptmt means  x[7];
  /*  and now examples of inquiries : */
  PRINT ('engineers of the 5th department :',
       {surname, x ∈ Q | post ='engineer' & deptmt = 5} );
  PRINT ('those, who can get pension and their salary': ,
     {<surname, salary> , x ∈ Q | age ⩾ 55 \/ deptmt = 7} );
  PRINT ('average salary of turners :',
        (c = 0; ∀ x ∈ Q | post ='turner':: c=c + salary;;)
        # {x, x ∈ Q | post ='turner'} );
  /* example of discharge : */
     Q = Q  less <'somebody', /* ...*/ >;
```

2.2. Now we describe a method of object representation and operation performance on the quotient of sets, generated by a system of subsets of some finite set A.

```
  /*  enumeration of A's elements : */
   k =0; n ={ }; ∀ e ∈A :: n(e) = (k = k + 1);;
  /*  a scale of lenghth # A is prepared by : */
      scale = bstring (# A);
  /*  procedure, which codes any subset of A */
code = proc (M);
```

```
        ∀ k ≤ # scale :: scale [k] = 0;;
/* if M contains an ebment from A, which has a number equal to
N, then the N-th component of scale get a value ".1." : */
        ∀ e ∈M :: scale [n(e)] = .1.;;
/* now we divide scale into D equal parts, where D is any
integer ⩾1; suppose D divides # A without rest and denote the
quontient by Q */
intvl = { };
∀ k ≤Q :: intvl (k) = scale [D* (k-1)+ 1 ÷   D * k];;
/* all the intervals with at least one .1. together with their
numbers (tags) describe M as a subset of A */
  {<tag, intvl (tag)> , tag ≤ Q | .1. ∈ intvl (tag)};;
/* of course, this method is effective if only # code (M)<<Q */
/* intersection of 'tagged' sets :*/
  intst = proc (M1, M2); M = { };
   ∀ x ∈ M1 | (∃ y ∈ M2 | x[1]= y[1]):
        m = m ⊂ <x[1], x[2] & y[2]> ;;
   {x, x ∈ m | .1. ∈ x};;
/* practically, the parameter D must be equal to the computer
wordsize, so that, for example, the BESM-6    computer could
perform operations with intervals extremly fast. */
```

3. Implementation.

The programming system works in the context of a special monitoring system, which provides the user with the usual service, including library, archive, editor and some other facilities.

SETL is implemetted in a semi-interpretative style, so that the system itself consists of a compiler (which produce an intermidiate code) and interpretating system, called SETL-machine, (which performs intermidiate code). The system is oriented not only to the SETL; it understands SETL, BALM and the SETL-machine input language (MSETLL). It must be mentioned that both SETL and BALM are available in two lexical versions : Russian and English. The general scheme admitted for this implementation is quite close to that of BALM system [2]. On the other hand, SETLB implementation [1] considered SETL as an extention of BALM, while SETL (C) and BALM are independent inputs of the system.

The compiler is a table-driven processor, which uses up to the 8 passes to produce target programs. Following the user's demand, the compiler reads the contence of tables from the tape and tunes its components to the concrete input language. Lexical processors constructs lexical codes of input units and takes into account descriptions of new operations. Decomposer builds up an extended syntactical structure of the program, while macroprocessor constructs standart syntactical structure. Semantic processor contains a number of semantic subprograms, responsible for separate syntactical constructions. Semantics considered as a correspondence between fragments of syntactical structure and pieces of target code also depends on the contence of some tables.

Tables can be changed both by manual rewriting of them and by some metaprocessors, which translate information, given in special form, into the components of some tables.

SETL-machine consists of about 250 operations, dealing with registers, stack, heap and other components of memory. In particu-

lar, any component of memory (or of a codeword of internal representation of BALM and SETL objects) is available by means of certain operation, which can be mentioned by its mnemocode in the MSETLL text.

The system permits also to include BALM and MSETLL components into SETL programs and MSETLL fragments into BALM-programs. The most important case of such including is a definition of new operations by means of lower level facilities, so that in the long run one can redefine all SETL operations according to his own needs. Besides, some optimisation can be formulated with terms of a number of macrodefinitions, containing MSETLL code in the right-hand side.

The system is written in EPSILON language for the BESM-6 computer; its size is about 27K words.

1. Schwartz J.T. On Programming, Installment I, An Interim Report on the SETL Project, New York University, 1972.
2. Harrison M. BALM Reference Manual, Version 4, New York University, 1972.