An example of how to initialise the SETL heap dynamically

Stefan M. Freudenberger

SETL Newsletter 217
Courant Institute of Mathematical Sciences
New York University

11. November 1981

Assume we have a large programme which we want to execute often but which requires that some initialisation be done before the actual start of the programme. Then we might be dissatisfied by the performance of the programme because of the time spent to re-initialise for every execution. Luckily there is a way to avoid this, as the following example shows:

```
program example;

    . . .    $ plenty of initialisation code

    . . .    $ actual program which depends on some input

end program example;
```

This example can be modified as follows:

```
program example;

    . . .    $ plenty of initialisation code

  debug rgarb, rdump;
  if getipp('INIT=0/1') = 1 then stop; end if;

    . . .    $ actual programme which depends on some input

end program example;
```

This modified programme is then compiled and initialised as follows (where we use the DEC VAX VMS syntax):

```
$ setl  example /init/dump=example.cod
$ setlx example /q2init=2/q2e=example.q2e/q2h=example.q2h
```

If we want to execute the programme without the initialisation step, we then simply say:

```
$ setlx /q2init/q2e=example.q2e/q2h=example.q2h
```

Since only the VMS implementation provides for the "mapped heap file" feature used in the preceding example, other implementations would compile and initialise as follows (where we use the UNIX syntax):

```
# stlc example.ltl                          # compile example.ltl
# stl  example.cod init dump=example.cod    # initialise
```

and then use the initialised code file via:

```
# stl example [ options ]
```

Several points should be kept in mind.

Firstly, the **debug** statement was added to the implementation to aid the debugging of the compiler, and not as a feature of interest to the general user. The **rgarb** option forces a garbage collection, thus compacting the heap. Since all sections of the heap which might contain live data are written to the dump file, this will assure the smallest possible dump file. The **rdump** option then writes a valid *Q2* file. Its primary purpose was (and is) to create a file which can be processed by the dump formatting routine SETL_DMP. More than one dump can be written to this file. For our purpose, however, only one dump may be taken. When these options are used, one should keep in mind that taking a dump is not exactly performing an **exit** function: several SETL-system-internal values currently are not part of the Q2 file, and thus are not preserved but will assume their normal initial value when execution re-starts. The three most important values lost in this way are the value of the **nargs** operator, which ordinarily returns the number of arguments of the current procedure, the value of the **lev** operator, which ordinarily returns the number of **ok**'s which are currently saved, and the environment chains in backtracked programmes, which are used to restore environments after a **fail** or **succeed**. If the dump is taken in the main programme before the first backtracking primitive has been executed, only some trace flags should be reset, which should be of little interest to the general user. Otherwise, execution most likely will be unpredictable, and results might be wrong.

Secondly, INIT is a programme parameter which cannot be used for any other purpose. In the last line, /INIT=0 is an implicit parameter. Of course, any parameter can be used for this purpose, subject to the normal constraint that it has no other pre-defined meaning.