A Set of Predicates for Fast Reading and Writing in SICStus

by

Hans Nilsson
ELLEMTEL AB
A Set of Predicates for Fast Reading and Writing in SICStus

Hans Nilsson

Swedish Institute of Computer Science, SICS
Box 1263, S-164 28 Kista, Sweden

and

ELLEMTEL AB
Box 1505, S-125 25 Ålvsjö, Sweden

September 6, 1991

T91:17

Abstract

The usual Prolog predicates for reading and writing are powerful but with some time overhead. In some situations a much faster reading and writing is required for a machine to machine message sending. This report describes the design and implementation of such predicates in SICStus Prolog.
1 Background

The read and write predicates of Prolog are very powerful. They can handle operator declarations and they write in a nice string representation. The drawback is of course that this handling adds some overhead in time.

When the data is written to be read by another Prolog process, it is not necessary to have operators and a representation readable by humans.

This report describes a set of predicates that implements such a simplified format.

2 The Format

The format of the term is a byte coded prefix notation. The idea is that it should be fast to read, since in for instance database applications a term is more often read than written. Therefore is every item preceded by information about how to read it.

The written term has the following major fields (as shown in the grammar below):

**version** a code for the version of the protocol. To make it possible to change the protocol later without breaking compatability.

**header** with some general information about the term:

- **number of variables** how many variables there are in the term. Only variables with two or more references are counted (called multi-ref variables in the grammar).  
- **atom table** those print names of atoms such that it is a space gain to just write them once and have references to them.  
- **number of cells** the number of heap cells that will be needed when reading the term.

**term** the real term representation.

The grammar of the written term is:

```
<written term> ::= <version> <header> <term>

<header> ::= <number of variables> <atom table> <number of cells>

<version> ::= <one character>
<number of variables> ::= <integer>
<atom table> ::= <number of atoms> {<string>}*
<number of atoms> ::= <integer>
<number of cells> ::= <integer>

<term> ::= <list with tail=[]>
  | <list with tail != []>
  | <short integer>
```
<list with tail=[]> ::= l <number of elements> {<term>}*
<list with tail != []> ::= L <number of elements> {<term>}{<tail>}
<short integer> ::= i <integer>
<long integer> ::= I <integer>
<float> ::= f <float number>
<multi-ref variable> ::= V <variable number>
<single-ref variable> ::= v
<atom> ::= a <string>
<multi-ref atom> ::= A <table index>
<struct> ::= s <functor> <arity> {<term>}
<multi-ref struct> ::= S <table index> <arity> {<term>}

<number of elements> ::= <integer>
<variable number> ::= <integer>
<functor> ::= <string>
<arity> ::= <integer>
<table index> ::= <integer>

<tail> ::= <term>

<integer> ::= <decimal characters> <null>
<string> ::= <characters except 'NULL'> <null>
<float number> ::= "characters produced by 'float_to_string'" <null>
<null> ::= 'NULL'

Note:

1. The arguments of a structure are written backwards. This is for a simplified term construction on input.

2. The <number of elements> is the number of list cells, that is, the non-nil tail-term that may exist is not counted.

Example:

    L = [a], <number of elements>=1
    L = [a|T], var(T) <number of elements>=1

3. <table index> is an index into the <atom table>.

4. Variables with the same <variable number> are the same variable.
3 The New Predicates

The set of new predicates all belong to the module prolog. They are:

'$\text{fast\_read}'(-\text{Term})$ Reads a term from the current input stream and unifies it with $\text{-Term}$

'$\text{fast\_write}'(+\text{Term})$ Writes the term $+\text{Term}$ to the current output stream.

'$\text{fast\_count}'(+\text{Term}, -\text{Size})$ If the term $+\text{Term}$ would be written, the size occupied would be $-\text{Size}$ bytes.

'$\text{fast\_buf\_read}'(-\text{Term}, +\text{BufferAddress})$ Reads a term from the memory area with address $+\text{BufferAddress}$ and unifies it with $-\text{Term}$. The memory area is not deallocated.

'$\text{fast\_buf\_write}'(+\text{Term}, -\text{Size}, -\text{BufferAddress})$ Writes the term $+\text{Term}$ to a memory area which this predicate allocates. The area is not garbage collected. The address is $-\text{BufferAddress}$ and the size is $-\text{Size}$ bytes.

4 Performance

The times for the new predicates and for the “old” ($\text{read}$ and $\text{write}$) are shown in table 1. Reading is always faster with the new predicates (13 – 53 times) but writing is sometimes a bit slower. This is good for data that is to be written once but read many times. The total time of one write and one read is always faster with the new predicates (1 – 5 times). Therefore are the new predicates also well suited for applications like socket communication.

Three sets of data was tested on a sun3 and on a sun4. The sets was:

clauses All clauses in the test file was written without blanks or comments on one file using $\text{write}$ and on another one using '$\text{fast\_write}'$. Those files were read with $\text{read}$ and '$\text{fast\_read}$' respectively, and the two readings were timed. The mean value of 10 such operations was calculated.

terms The clauses was splitted\(^1\) and the same kind of test was run.

list All clauses was put in one list. This list was read with a single read.

The size of the data is shown in table 2. The sizes are about the same for the new and old clauses.

References


\(^1\)For example, $p:-q,r$ was splitted into $p, q$ and $r$. 

4
<table>
<thead>
<tr>
<th>Test Host</th>
<th>Data</th>
<th>Read old</th>
<th>Read new</th>
<th>Read ratio</th>
<th>Write old</th>
<th>Write new</th>
<th>Write ratio</th>
<th>Read + Write old</th>
<th>Read + Write new</th>
<th>Read + Write ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>sun3</td>
<td>clauses</td>
<td>1687</td>
<td>102</td>
<td>16.8</td>
<td>847</td>
<td>490</td>
<td>1.7</td>
<td>2534</td>
<td>592</td>
<td>4.3</td>
</tr>
<tr>
<td></td>
<td>terms</td>
<td>1529</td>
<td>100</td>
<td>15.3</td>
<td>672</td>
<td>904</td>
<td>0.7</td>
<td>2201</td>
<td>1004</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>4222</td>
<td>79</td>
<td>53.4</td>
<td>837</td>
<td>827</td>
<td>1.0</td>
<td>5059</td>
<td>906</td>
<td>5.6</td>
</tr>
<tr>
<td>sun4</td>
<td>clauses</td>
<td>557</td>
<td>38</td>
<td>14.7</td>
<td>329</td>
<td>240</td>
<td>1.4</td>
<td>886</td>
<td>278</td>
<td>3.2</td>
</tr>
<tr>
<td></td>
<td>terms</td>
<td>555</td>
<td>41</td>
<td>13.5</td>
<td>260</td>
<td>604</td>
<td>0.4</td>
<td>815</td>
<td>645</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>list</td>
<td>1281</td>
<td>27</td>
<td>47.4</td>
<td>324</td>
<td>366</td>
<td>0.9</td>
<td>1605</td>
<td>393</td>
<td>4.1</td>
</tr>
</tbody>
</table>

Table 1: Times (ms) for reading and writing with the `read`, `writeq`, `fast_read` and `fast_write` predicates

<table>
<thead>
<tr>
<th>Test</th>
<th>Number of things</th>
<th>Size in bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>old</td>
</tr>
<tr>
<td>clauses</td>
<td>46</td>
<td>4731</td>
</tr>
<tr>
<td>terms</td>
<td>192</td>
<td>4791</td>
</tr>
<tr>
<td>list</td>
<td>4831</td>
<td>4272</td>
</tr>
</tbody>
</table>

Table 2: Sizes (bytes) of the test data