

Simple Software Measurement for PROLOG.

by Lindsey Spratt, Ph. D.

September 30, 1996

This document is a brief note on software measurement for PROLOG. There is a more extensive discussion in the “Objective Analytic Assessment” chapter of the Ph. D. dissertation *Seeing the Logic of Programming with Sets* by Lindsey Spratt, University of Kansas, 1996. The focus in this document is to measure the *size* of PROLOG programs.

Prolog syntax.

Edinburgh-syntax PROLOG (much the most common syntax for PROLOG) allows for programmer-defined operator-precedence syntax. This notion of “operator” has nothing to do with functional evaluation, it is a purely syntactic notion. The programmer may define that a particular token OP has a particular precedence PRED (an integer in an implementation-dependent range, commonly 0 to 1024), and a particular associativity and position AP. The associativity determines how unparenthesized sequences of OP associate, and the position indicates whether the OP is a prefix, suffix, or infix operator. For example, ‘xfy’ specifies an infix operator (the operator replaces the ‘f’) with right-associativity (‘a OP b OP c’ = ‘a OP (b OP c)’). Operator syntax is used as an alternative to the basic structure syntax—structures written in operator syntax are semantically identical with structures written basic structure syntax. There are several operator definitions which are in the default environment, such as:

```
op(1200, xfx, ‘:-’).  
op(1000, xfy, ‘,’).  
op(500, yfx, ‘+’).
```

The *member* program in operator syntax is:

```
member(X, [X|_]).  
member(X, [_|T]) :- member(X, T).
```

This can also be written without operator syntax:

```
member(X, cons(X, _)).  
:- (member(X, cons(_, T)), member(X, T)).
```

The counting scheme.

Since the operator syntax affects the concrete representation, the token count must be sensitive to it. The natural approach is to consider those tokens which are used syntactically as operators as operator tokens. Additional operators are the enclosing syntactic items: ‘()’, ‘[]’, and ‘{ }’. We can construct a table which defines the token counting method for PROLOG using the definitions of $S_1, \eta set_1, S_2,$ and ηset_2 found in [Bieman et al. 1991]: S_1 is the sequence of operator token types in the program, ηset_1 is the set of operator token types in the program, S_2 is the sequence of operand token types in the program, and ηset_2 is the set of operand token types in the program. A PROLOG program source is a sequence of clauses, where each clause is a term followed by a ‘.’ (period). A term is an atom, number, variable, or structure. A structure has a functor and one or more arguments. The arguments are terms.

term	additions to $S_1, \eta set_1$	additions to $S_2, \eta set_2$
-		$t_{\text{underscore}}$
[]		t_{nil}
symbol		t_{symbol}
functor(term-list)	$t_{()}$	t_{functor}
(term)	$t_{()}$	
term OP term	t_{OP}	
OP term	t_{OP}	
term OP	t_{OP}	
[term-list]	$t_{[]}$	
[term-list term]	$t_{[]}, t_{ }$	
{term}	$t_{\{\}}$	
term .	t_{period}	
term , term-list	t_{comma}	

The member/2 program

Using this counting scheme for the *member/2* example:

$$S_1 = [t_{\text{period}}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{\text{period}}, t_{:-}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{()}, t_{\text{comma}}]$$

$$\eta set_1 = \{t_{\text{period}}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{:-}\}$$

$$S_2 = [t_{\text{member}}, t_X, t_X, t_{\text{underscore}}, t_{\text{member}}, t_X, t_{\text{underscore}}, t_T, t_{\text{member}}, t_X, t_T]$$

$$\eta set_2 = \{t_{\text{member}}, t_X, t_{\text{underscore}}, t_T\}$$

Using this token analysis of the *member/2* program, we get the following counts and size

measurements:

$$\begin{aligned}\eta_1 &= 6 \\ N_1 &= 13 \\ \eta_2 &= 4 \\ N_2 &= 11 \\ \eta &= 10 \\ N &= 24 \\ V &= 24 \cdot \log_2(10) = 80\end{aligned}$$

The union/3 program.

Operator concrete representation:

```
union([H|X], Y, Z) :-
    (member(H, Y)
     -> Z = T
    );
    Z = [H|T]
),
    union(X, Y, T). % Clause 1a
```

```
union([], Y, Y). % Clause 2a
```

Base representation:

```
:- (union( '.'(H, X), Y, Z),
    \,'(';'('->'(member(H, Y),
                '='(Z, T)),
        \='(Z, '.'(H, T))),
    union(X, Y, Z))). % Clause 1b
```

```
union([], Y, Y). % Clause 2b
```

Figure 1: Original and Base representation form of the PROLOG union/3 program.

We apply this counting technique to the *union/3* solution as follows:

Clause 1a:

$$S_1 = [t_{\text{period}}, t_{\text{:-}}, t_{\text{()}}, t_{\text{comma}}, t_{\text{[]}}, t_{\text{|}}, t_{\text{comma}}, t_{\text{comma}}, t_{\text{()}}, t_{\text{semicolon}}, t_{\text{->}}, t_{\text{()}}, t_{\text{comma}}, t_{\text{=}}, t_{\text{=}}, t_{\text{[]}}, t_{\text{|}}, t_{\text{()}}, t_{\text{comma}}, t_{\text{comma}}]$$

$$\eta_{set_1} = \{t_{\text{period}}, t_{\text{:-}}, t_{\text{()}}, t_{\text{comma}}, t_{\text{[]}}, t_{\text{|}}, t_{\text{semicolon}}, t_{\text{->}}, t_{\text{=}}\}$$

$$S_2 = [t_{\text{union}}, t_{\text{H}}, t_{\text{X}}, t_{\text{Y}}, t_{\text{Z}}, t_{\text{member}}, t_{\text{H}}, t_{\text{Y}}, t_{\text{Z}}, t_{\text{T}}, t_{\text{Z}}, t_{\text{H}}, t_{\text{T}}, t_{\text{union}}, t_{\text{X}}, t_{\text{Y}}, t_{\text{T}}]$$

$$\eta_{set_2} = \{t_{\text{union}}, t_{\text{H}}, t_{\text{X}}, t_{\text{Y}}, t_{\text{Z}}, t_{\text{member}}, t_{\text{T}}\}$$

Clause 2a:

$$S_1 = [t_{\text{period}}, t_{()}, t_{\text{comma}}, t_{\text{comma}}]$$

$$\eta_{set_1} = \{t_{\text{period}}, t_{()}, t_{\text{comma}}\}$$

$$S_2 = [t_{\text{union}}, t_{\text{nil}}, t_Y, t_Y]$$

$$\eta_{set_2} = \{t_{\text{union}}, t_{\text{nil}}, t_Y\}$$

Combined values for these clauses:

$$S_1 = [t_{\text{period}}, t_{:-}, t_{\text{comma}}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{\text{comma}}, t_{\text{comma}}, t_{()}, t_{\text{semicolon}}, t_{->}, t_{()}, t_{\text{comma}}, t_{=}, t_{=}, t_{[]}, t_{|}, t_{()}, t_{\text{comma}}, t_{\text{comma}}]$$

$$+ [t_{\text{period}}, t_{()}, t_{\text{comma}}, t_{\text{comma}}]$$

$$\eta_{set_1} = \{t_{\text{period}}, t_{:-}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{\text{semicolon}}, t_{->}, t_{=}\} \cup \{t_{\text{period}}, t_{()}, t_{\text{comma}}\}$$

$$= \{t_{\text{period}}, t_{:-}, t_{()}, t_{\text{comma}}, t_{[]}, t_{|}, t_{\text{semicolon}}, t_{->}, t_{=}\}$$

$$S_2 = [t_{\text{union}}, t_H, t_X, t_Y, t_Z, t_{\text{member}}, t_H, t_Y, t_Z, t_T, t_Z, t_H, t_T, t_{\text{union}}, t_X, t_Y, t_T]$$

$$+ [t_{\text{union}}, t_{\text{nil}}, t_Y, t_Y]$$

$$\eta_{set_2} = \{t_{\text{union}}, t_H, t_X, t_Y, t_Z, t_{\text{member}}, t_T\} \cup \{t_{\text{union}}, t_{\text{nil}}, t_Y\}$$

$$= \{t_{\text{union}}, t_{\text{nil}}, t_H, t_X, t_Y, t_Z, t_{\text{member}}, t_T\}$$

Using the combined $S_1, \eta_{set_1}, S_2,$ and η_{set_2} values determined above, we can calculate the various token counts:

$$\eta_1 = 9$$

$$N_1 = 24$$

$$\eta_2 = 8$$

$$N_2 = 21$$

These counts yield the size measurements:

$$\eta = 17$$

$$N = 45$$

$$V = 45 \cdot \log_2(17) = 184$$

$$t_{\text{union}}, t_X, t_Y, t_Z] \quad (25)$$

$$+ [t_{\text{union}}, t_{\text{nil}}, t_Y, t_Y] \quad (4)$$

$$\eta \text{set}_2 = \{t_{\text{union}}, t_{\text{nil}}, t_Y, t_Y, t_H, t_X, t_Y, t_Z, t_{\text{member}}, t_T, t_{\text{nil}}\}$$

$$\cup \{t_{\text{union}}, t_{\text{nil}}, t_Y\}$$

$$= \{t_{\text{union}}, t_{\text{nil}}, t_Y, t_H, t_X, t_Y, t_Z, t_{\text{member}}, t_T, t_{\text{nil}}\}$$

Using the combined S_1 , ηset_1 , S_2 , and ηset_2 values determined above for the base-syntax version of the *union/3* program, we can calculate the various token counts:

$$\eta_1 = 3$$

$$N_1 = 25$$

$$\eta_2 = 14$$

$$N_2 = 29$$

These counts yield the size measurements:

$$\eta = 17$$

$$N = 54$$

$$V = 54 \cdot \log_2(17) = 221$$

Discussion.

This demonstrates the typical result that using the operator syntax gives a smaller program compared to the base syntax. The total number of unique tokens does not change (17, in this case), but the total number of tokens does (45 versus 54).

In the operator syntax there are fewer parentheses and argument-list commas than in the base syntax. In the transformation from the operator-syntax to the base-syntax, each binary operator term ‘term OP term’ is replaced by ‘OP(term, term)’, one token (the operator OP) in the first case versus three tokens (the operator OP, the parentheses, and the argument list comma) in the second case. Similarly, the unary operator terms go from one token (the OP) to two tokens (the OP and the parentheses). The list syntax also provides a substantial savings: ‘[term1, ..., termN]’ becomes “.’(term1, ‘.’(term2, ... ‘.’(termN, [])...))”. This changes N tokens in the list syntax for a list of N terms (one for the brackets and N-1 for the commas) into 3N+1 tokens in the base syntax (a ‘.’, a pair of parentheses, and an argument comma for each term, plus the ‘[]’ (nil) token for the end of the list). Depending on the program, the size difference between the operator/list syntax and the base syntax can be quite large.

Bieman et al. 1991 “Moving from Philosophy to Practice in Software Measurement” by James Bieman, Norman Fenton, David Gustafson, Austin Melton, and Robin Whitty. Pages 38-59 in *Formal Aspects of Measurement: Proceedings of the BCS-FACS Work-*

shop on Formal Aspects of Measurement, South Bank University, London, 5 May 1991
edited by Tim Denvir, Ros Herman, and R. W. Whitty. Springer-Verlag:London. 1991.