The University of Arizona
Department of Computer Science
Tucson, Arizona 85721

Corrigenda for *The Macro Implementation of SNOBOL4*

Ralph E. Griswold

February 9, 1987

A list of corrections to *The Macro Implementation of SNOBOL4*, published by W.H. Freeman, follows.

In this list, line numbers are counted from the top of the page. This line numbering system was chosen for clarity, although it causes extra work in locating an error near the bottom of a page. Page headings and blank lines are not counted. In figures, blank areas, rulings, and horizontal lines are not counted, but lines of dots are. In a few places, especially where figures are involved, there are ambiguities in counting lines. The context provided for corrections can be used to resolve these problems.

Brackets ([ ]) surround comments about corrections. In places where a large amount of text is to be inserted, this text has been broken out of the regular column format and enclosed in lines ruled across the page.

I have attempted to assure the accuracy and completeness of the corrections. Please let me know if there are errors or omissions in the corrigenda.

Thanks go to individuals who called errors to my attention: Walter Bosse, John Doyle, Pierre Goyer, John Hallyburton, Ken Moody, Dan Ophir, Forrest Pitts, William Sears, and Martha Wagner.

pages	lines	current text	replacement text
42	16	20	10
45	26	synonymns	synonyms
61	1	4d	4cpd
	3	4d	4cpd
	10–11	variable, ... of the string.	variable with an offset of 4cpd, where cpd is the number of characters that fit into the space occupied by a descriptor.
66	13	POS(M) | POS(P)	(POS(M) | POS(P))
69	4		F [in F field of A descriptor]
	9		F [in F field of A descriptor]
	14		F [in F field of A descriptor]
87	6		–––> [at top descriptor of left block]
	7	–––> [at left block]	[delete arrow]
92	9	42 0 41	41 0 42
	12	60 0 59	59 0 60
93	15	$\mathtt{replace}_3$	$\mathtt{repl}_3$
	20	$\mathtt{replace}_3$	$\mathtt{repl}_3$
94	1	$*_2$	$+_2$ [in node at top]
	7	$+_2$	$*_2$ [at right of arrow]
95	4	DUPL(7,X)	DUPL(X,7)
100	36	$L_i,...,L_m$.	$L_i,...,L_m$ and the value of E is N.
101	4	{E}	{N}
	15		––––––> [move arrow up one discriptor]
	16	––––––>	[delete arrow]
	29	E	N
102	4		*F* [in F field of left descriptor]

103	28		———–> [move arrow up one descriptor]
	29	———–>	[delete arrow]
104	11	{E}	{N}
	32	E	N
105	9	values of A_1,...	values of F, A_1,...
106	1	5	8
107	4		F [in F field of left descriptor]
	30	F	F_i [two places]
108	4		*F* [in F field of left descriptor]
110	10	s	s_i
111	15	L+ n + 2 ...)))	L + d(n + 2 + $(((o_n s_{n-1} + o_{n-1})s_{n-2} + \dots + o_2)s_1 + o_1)$)
	16	$s_n o_n$	$o_n s_{n-1}$
	17	... s_n and l_n are ...	... the last dimension is the ...
112	33	M	M′ = M + 1
	34	N	N′ = N + 1
113	1	M	M′
	1	N	N′ [two places]
	5	N	N′ [two places]
	6	N	N′
121	12	TAB(4)|LEN(2)	TAB(4) | LEN(2)
	14	$\mathtt{replace}_3$	$\mathtt{repl}_3$
124	13	third argument ...	third argument. The offset of the third component is an alternate of 6d rather than a subsequent of 3d as suggested by Figure 8.7.7, since $\mathtt{mfarb}_2$ uses its knowledge of the pattern structure to obviate reprocessing of $\mathtt{mnull}_2$.
125	9–16	There are two ... the pattern for P1.	[replace by text below]

An unevaluated expression may appear in several contexts: as a pattern itself, as an argument of a pattern-valued function, or as an operand of concatenation or alternation.

If an entire pattern is an unevaluated expression, that expression is evaluated and the result is used in pattern matching.

125	23–32	8d *T* ∗ ... `∗F(X) LEN(3)`.	[delete entire figure]
126	7–29	As a pattern ... during evaluation.	[replace by text below]

If an unevaluated expression appears as an operand of alternation or concatenation, a pattern is constructed that provides matching procedures to handle the unevaluated expression during pattern matching. An example is given by

```
P1 = ∗F(X) LEN(3)
```

which constructs the pattern shown in Figure 8.7.9.

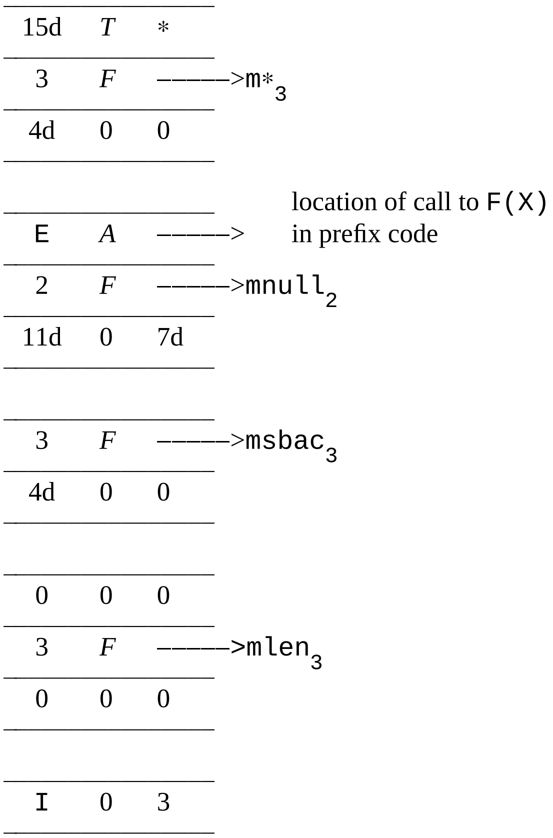

Figure 8.7.9
The Pattern `∗F(X) LEN(3)`.

When the first component is encountered during pattern matching, the matching procedure for $\texttt{m*}_3$ causes the prefix code pointed to by its argument to be interpreted, resulting in the evaluation of `F(X)`. The result is stored for possible later use in the argument descriptor of the component with the matching procedure $\texttt{msbac}_3$. The connector descriptor and cursor position are pushed onto `PATSTK` as in step (3) of the pattern–matching algorithm described in Section 8.7.3. Next SCNR is called at an entry that causes pattern matching to continue, but without the usual initialization. The signals on return are transmitted in the normal manner.

The component with matching procedure $\mathtt{msbac}_3$ is necessary in case the match resulting from $\mathtt{m}*_3$ is successful, but failure of a subsequent component forces backup. In this case, $\mathtt{msbac}_3$ returns control to its argument (the result of evaluating the argument of $\mathtt{m}*_3$) to attempt to match alternatives. If this attempt succeeds, matching continues as before. It it fails, match failure is signaled in the usual fashion. A diagram of component relations is shown in Figure 8.7.10.

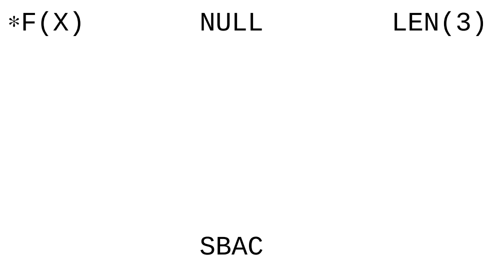

Figure 8.7.10
Relation of Components for Unevaluated Expressions.

127	36	8.7.10	8.7.11
128	22	**8.7.10**	**8.7.11**
132	16	1	0
137	19	2	2 [two places]
140	8	exectuion	execution
144	4	S	[delete]
150	4	$('SUM' N) 10	$('SUM' N) = 10
160	22	ideas	idea
164	2	V_1+V_3	V_1+V_2
169	2	X = 1	X = 1
	6	X = 1	X = 1
170	4	...stack position. Descriptors ...	...stack position. This is the location of the last (most recently pushed) descriptor. Descriptors ...

	20		––––––> [move arrow up one descriptor]
	21	––––––>	[delete arrow]
	27		––––––> [move arrow up one descriptor]
	28	––––––>	[delete arrow]
172	3		––––––> [move arrow up one descriptor]
	4	––––––>	[delete arrow]
	7		––––––> [move arrow up one descriptor]
	8	––––––>	[delete arrow]
	16		––––––> [move arrow up one descriptor]
	17	––––––>	[delete arrow]
185	2	writtern	written
281	14–15	The `DEFINE` ... entry point `F`.	The `DEFINE` block for `F()` consists of two descriptors, one for the entry point, `F`, and one for the name of the function, `F`.
287	28	**9.2.17**	**9.2.16**
	34	**9.2.18**	**9.2.17**