Jaanus P.

1990 FORML CONFERENCE PROCEEDINGS

FORTH MODIFICATION LABORATORY

Twelfth Asilomar FORML Conference
November 23-25, 1990
Asilomar Conference Center
Pacific Grove, California

EuroFORML '90 Conference
October 12-14, 1990
Potters Heron Hotel
Ampfield, Nr Romsey
Hampshire, U.K.

A Publication of
The Forth Interest Group
P. O. Box 8231, San Jose, CA 95155

Published in the United States of America

Cover design by Steven Reiling

SESSION 3

ALGEBRAIC SPECIFICATION OF STACK-EFFECTS FOR FORTH PROGRAMS

Jaanus Poial

Tartu University, Estonia

The main idea of this work is to introduce a formalism which allows to check the stack-effects according to the program text. The same formalism will be used in case of Forth-programs being generated by some formal mechanism (we shall deal with the syntax-directed translation scheme). Our attention is concentrated only to the aspect of parameter passing through the stack excluding memory handling, I/O, etc.

ALGEBRAIC SPECIFICATIONS OF STACK-EFFECTS FOR FORTH-PROGRAMS

Assoc. Prof. **Jaanus Põial**

Tartu University, Estonia

The most important quality of the Forth-word is its stack-effect. Particularly strong discipline is required when a large application (hundreds of screens) is written or more than two programmers participate in the project. There are some good tools to trace the program, but in the complicated environment it is an inconvenient task to trace all branches of the program. The main idea of this work is to introduce a formalism which allows to check the stack-effects according to the program text. The same formalism will be used in case of Forth-programs being generated by some formal mechanism (we shall deal with the syntax-directed translation scheme). Our attention is concentrated only to the aspect of parameter passing through the stack excluding memory handling, I/O, etc.

Each Forth-word has an informal specification of its stack-effect given in the form:

input parameter types --- output parameter types

The type lists given above are ordered, the end of the list corresponds to the top of the stack. This specification does not say anything about the essence of the operation.

Our further investigation is based on the theory of semigroups. In [NP70] M. Nivat and J.F. Perrot introduced a 0-bisimple inverse semigroup called polycyclic monoid. We need some notations to express the main ideas:

A - an alphabet (finite set of type names),

A^* - the set of strings over A (set of type lists),

Λ - the empty string ($\Lambda \in A^*$ for arbitrary A),

ab - the concatenation of strings *a* and *b*,

0 - the *null specification* (specifies the error-situation).

The *set of specifications* over A is the union:

$$\Phi(A) = (A^* \times A^*) \cup \{ \mathbf{0} \}.$$

Let $[s_1 \text{ --- } s_2]$ denote a pair $(s_1, s_2) \in A^* \times A^*$.

Here s_1 is the list of input parameters and s_2 is the list of output parameters as above. If there is no need to emphasize the alphabet we use Φ instead of $\Phi(A)$.

The pair $(\Lambda, \Lambda) = [\text{ --- }]$ is called the *empty specification* and denoted **1**.

We may define the *product of specifications* as follows:

1) $\forall\ s \in \Phi :\ s\mathbf{0} = \mathbf{0}s = \mathbf{0}$,

2) $\forall\ s, t \in \Phi \setminus \{ \mathbf{0} \} :\ st = [s_1 \text{ --- } s_2] [t_1 \text{ --- } t_2] =$

$$= \begin{cases} [as_1 \text{ --- } t_2] , & \text{if } t_1 = as_2 , \\ [s_1 \text{ --- } bt_2] , & \text{if } s_2 = bt_1 , \\ \mathbf{0} , & \text{otherwise.} \end{cases}$$

The set Φ is isomorphic to the polycyclic monoid (proof in [NP70]). Consequently:

1) $\forall\ s, t \in \Phi : st \in \Phi$,

2) $\forall\ r, s, t \in \Phi :\ (rs)t = r(st)$,

3) $\forall\ s \in \Phi :\ s\mathbf{1} = \mathbf{1}s = s$,

4) $\forall\ s \in \Phi : s\mathbf{0} = \mathbf{0}s = \mathbf{0}$.

Let Δ be a set of considered operations. Δ^* is a set of sequences from these operations (set of programs).

Specifications are given by the mapping $s : \Delta^* \longrightarrow \Phi$:

1) $\forall\, \Pi \in \Delta : \; s(\Pi) \in \Phi \setminus \{ 0 \}$ is a given specification of the operation Π,

2) $s(\Lambda) = 1$ (the empty program),

3) $\forall\, \omega \in \Delta^*, \; \forall\, \Pi \in \Delta : \; s(\omega\Pi) = s(\omega)s(\Pi)$.

The program $\omega \in \Delta^*$ is said to be *correct*, if $s(\omega) \neq 0$, and *closed*, if $s(\omega) = 1$.

We may define a set of correct programs as

$$\mathrm{CORRECT}(\Delta, s) = \{ \omega \in \Delta^* \mid s(\omega) \neq 0 \}$$

and a set of closed programs as

$$\mathrm{CLOSED}(\Delta, s) = \{ \omega \in \Delta^* \mid s(\omega) = 1 \}.$$

Obviously

$$\{ \Lambda \} \subset \mathrm{CLOSED} \subset \mathrm{CORRECT} \subset \Delta^*.$$

These sets are algorithmically solvable because we may calculate the formal specification of a program according to the specifications of existing words. The control structures of Forth need special treatment when writing practical correctness-checker (an attempt is made to include the correctness checking into the editor immediately).

Let $s \in \Phi$. The *inverse element* $s^{-1} \in \Phi$ is defined by the conditions:

1) if $s = 0$, then $s^{-1} = 0$,

2) if $s = [\, s_1 \text{ --- } s_2 \,]$, then $s^{-1} = [\, s_2 \text{ --- } s_1 \,]$.

The *partial order relation* $\leq$ is convenient in the theory of semigroups ([CP67]): $s \leq t$, iff $st^{-1} = ss^{-1}$. Since $0t^{-1} = 00 = 0$, we have $0 \leq t$ for all $t \in \Phi$.

Theorem 1. The following assertions are equivalent:

1) $[\ s_1 \text{ --- } s_2\ ] \leq [\ t_1 \text{ --- } t_2\ ]$,

2) $\exists\ a \in A^*$: $[\ s_1 \text{ --- } s_2\ ] = [\ at_1 \text{ --- } at_2\ ]$,

3) $[\text{ --- } s_1\ ]\ [\ t_1 \text{ --- } t_2\ ]\ [\ s_2 \text{ --- }\ ] = 1$,

4) $[\text{ --- } s_1\ ]\ [\ t_1 \text{ --- } t_2\ ] = [\text{ --- } s_2\ ]$,

5) $[\ t_1 \text{ --- } t_2\ ]\ [\ s_2 \text{ --- }\ ] = [\ s_1 \text{ --- }\ ]$.

Having a partial order relation, the problem of comparable elements arises. At present we know that the null element is comparable with all elements of Φ.

Theorem 2. The following comparability conditions are equivalent for the elements of Φ:

1) $s \neq 0$, $t \neq 0$ and s is comparable with t,

2) there exists an element $r \neq 0$ so that $r \leq s$ and $r \leq t$,

3) there exists an element $u \in \Phi$ so that $s \leq u$, $t \leq u$ and at least one of the conditions $st^{-1} \neq 0$, $s^{-1}t \neq 0$ holds.

Further we need a method to solve inequalities given by such a partial order relation. These inequalities may have a "recurrent" form like $s \leq rst$.

Theorem 3. Inequality $s \leq rst$ by $s \neq 0$ holds in Φ, iff there exist $a, b, c \in A^*$ so that

$ar_1 = s_1$, $ar_2 = bs_1$, $ct_1 = bs_2$ and $ct_2 = s_2$.

We finish the study of algebraic properties of Φ with observing infimum and supremum of subsets of Φ.

An arbitrary two-element subset $\{ s, t \} \subset \Phi$ has the greatest lower bound, which may be expressed as

$$\inf \{s, t\} = \begin{cases} s, \text{ if } s \le t, \\ t, \text{ if } t \le s, \\ 0, \text{ if } s \text{ and } t \text{ are non-comparable.} \end{cases}$$

This definition is obvious (see also Theorem 2). The notion of supremum is more complicated. For the null-element we may define $\sup \{r, 0\} = r$ in the case of all $r \in \Phi$. Let $s, t \in \Phi \setminus \{ 0 \}$. If there exist $a, b, c, d, e \in A^*$ so that $s = [\ abd \text{ --- } abe\ ]$, $t = [\ cbd \text{ --- } cbe\ ]$ and the length of b is chosen maximal (possible), then there exists

$$\sup \{s, t\} = [\ bd \text{ --- } be\ ].$$

This choice of b guarantees the defined upper bound to be the least (see also Theorem 1). If it is impossible to choose these five strings in any way, then no supremum exists.

A set of stack operations Δ and the homomorphism

$$s : \Delta^* \longrightarrow \Phi$$

induce two languages, named CORRECT(Δ, s) and CLOSED(Δ, s) before. A program $\omega \in$ CLOSED(Δ, s) as a whole has neither input nor output parameters. At the same time parameter types inside of ω are compatible, i.e. ω is correct. All "user-oriented" programs must be closed because the stack is only an implementation-level tool. This point of view evokes our special interest to the closed programs.

We investigate the syntax-directed translation scheme ([AU72]) and try to answer the question if there exists an algorithm for detecting, whether or not a given scheme generates only closed programs.

The *syntax-directed translation scheme* is a quintuple $T = (N, \Sigma, \Delta, R, S)$, which consists of the following components:

N - a non-terminal alphabet, $S \in N$ - a fixed initial symbol (an axiom), Σ - an input alphabet, Δ - an output alphabet and R is a finite set of translation rules of form

$$A_0 \longrightarrow x_0 A_1 x_1 \ldots x_{n-1} A_n x_n \ ,\ z_0 B_1 z_1 \ldots z_{n-1} B_n z_n$$

$(x_i \in \Sigma^*,\ z_i \in \Delta^*\ A_i, B_i \in N)$, by which the vector $(B_1, \ldots, B_n)$ is some permutation of the vector $(A_1, \ldots, A_n)$.

If $(B_1, \ldots, B_n) = (A_1, \ldots, A_n)$ for all rules of R, then the syntax-directed translation scheme is said to be *simple*.

The syntax-directed translation scheme defines a set of pairs $(\sigma, \omega) \in \Sigma^* \times \Delta^*$, which may be derived from the pair (S, S). The first components of these pairs constitute an input language of the scheme, the second components - an output language. The string ω is said to be the translation of the string σ. The translation scheme may also be treated as a pair of grammars $T = (G_1, G_2)$, defined by R.

Let the input grammar G_1 be a reduced context-free grammar ([AU72]). For the output grammar we use a notation $G_2 = (N, \Delta, P, S)$. The output language of T is a set

$$L_2 = \{ t \mid t \in \Delta^* \ \&\ S \overset{+}{\underset{G_2}{\Longrightarrow}} t \}.$$

Let the output symbols $\Pi \in \Delta$ have specifications $s(\Pi)$, i.e. $s(\Delta) \subset \Phi \setminus \{ 0 \}$.

The syntax-directed translation scheme T is said to be *correct* if $L_2 \subset \mathrm{CLOSED}(\Delta, s)$.

The system of inequalities I (T, s) is defined:

1) An unknown $Z(A) \in \Phi$ is introduced for each $A \in N$.

2) The rules of G_2 are replaced by inequalities - the rule of form $A \longrightarrow X_1 \ldots X_k$ induces $Z(A) \le Y_1 \ldots Y_k$, where $Y_i = s(X_i)$, if $X_i \in \Delta$, and $Y_i = Z(X_i)$, if $X_i \in N$. If the right part of the rule is empty, then we take $Z(A) \le 1$.

3) The inequality $1 \le Z(S)$ is added where S is the axiom of the scheme T.

The following auxiliary sets are introduced for each nonterminal symbol $A \in N$:

$$C(A) = \{ (u, v) \in \Delta^* \times \Delta^* \mid S \Longrightarrow^* uAv \},$$

$$L(A) = \{ \omega \in \Delta^* \mid A \Longrightarrow^+ \omega \}.$$

Theorem 4. The following assertions are equivalent:

1) the syntax-directed translation scheme T is correct,

2) the system of inequalities I (T, s) is solvable,

3) for each nonterminal symbol $A \in N$ there exists a supremum

$$m(A) = \sup \{ [s(vu)]^{-1} \mid (u, v) \in C(A) \},$$

by which the following inequality holds

$$m(A) \le \inf \{ s(\omega) \mid \omega \in L(A) \}.$$

This theorem allows to check an initial translation scheme for which Forth is the output language.

It may happen that it is hard to classify the parameters of stack operations because there are many type-independent operations like DUP, SWAP, DROP in Forth, etc. For that case it is useful to introduce "wild card" (or "free") symbols, which are able to replace an arbitrary type name (let us use asterisks to express "wild card" symbols). The following examples of specifications are used to illustrate this approach:

DUP	[* --- * *]	copies the top element on the top of the stack,
SWAP	[** * --- * **]	interchanges two top elements,
!	[* *addr* ---]	stores the element of arbitrary type at *addr* (mixed specification).

It is possible to generalize the operation of multiplication for "wild card" symbols with some restrictions (no new "wild cards" may appear on the right side of any specification). █

References

[AU72] Aho A.V., Ullman J.D. *The theory of parsing, translation and compiling.* vol.1: Parsing. - Englewood Cliffs, 1972.

[CP67] Clifford A.H., Preston G.B. *The algebraic theory of semigroups.* vol.2. - Rhode Island, 1967.

[NP70] Nivat M., Perrot J.F. Une gèneralisation du monoïde bicyclique. - *C. R. Acad. Sci. Paris,* 271A, 1970, p.824-827.