SOME IDEAS ON FORMAL SPECIFICATION OF FORTH PROGRAMS

Jaanus Pöial
Tartu University
jaanus@cs3.ut.ee

There have been different ideas to describe the semantics of Forth. Each of these ways has its own output to practical software development. Operational approach uses kind of abstract machine and programs are considered as state transformers. Algebraic approach has been fruitful to investigate semantics of parameter passing and type inference. Currently an attempt is made to define some features of the Forth language using axioms and deduction rules. Author is thankful to Ms Maris Tudre for her cooperation on this topic.

First of all some predicates have to be defined to describe stacks, memory, dictionary, input stream, output stream etc. Kernel of Forth is defined in form of axioms. Also there exists an axiom scheme which allows to introduce secondary words to proofs without building the full proof tree.

Programs are introduced using deduction rules. There are rules for colon definitions, control structures, *CREATE...DOES>* , etc. The main idea is to find Forth-specific rules, which reflect the freedom to extend the language. Author hopes that *CREATE...DOES>* is a good example.

This approach may help in reasoning about Forth programs, in finding bugs, but also in program construction and synthesis. At present there is no completed theory, these ideas form a basis for further discussion.

Let **Z** denote the domain of "cells", **A** the domain of data space addresses. **F** will be the set of logical formulas and **T** a set of possible symbols in input (or output) stream.

Let us define some basic predicates, which will be used in formulas.

Data stack contains finite number of cells (top last) $S(list)$

$S: \mathbf{Z}^* \to \{false, true\}$

Return stack is similar $RS(list)$

$RS: \mathbf{Z}^* \to \{false, true\}$

Data space in memory $D(address, value)$

$D: \mathbf{A} \times \mathbf{Z} \to \{false, true\}$

Dictionary entries in form $indict\ (\{Precondition\}\ Name\ \{Postcondition\})$

$indict: \mathbf{F} \times \mathbf{T}^* \times \mathbf{F} \to \{false, true\}$

Output stream $out(string)$

$out: \mathbf{T}^* \to \{false, true\}$

Input stream $instream\ (string, delimiter)$

$instream: \mathbf{T}^* \times \mathbf{T} \to \{false, true\}$

There are axioms for primary words in form $\{Precondition\}\ Word\ \{Postcondition\}$ and an axiom scheme for secondary words in form

$$\{\mathtt{R}\&\mathtt{P}\&indict(\{\mathtt{P}\}\ Name\ \{\mathtt{Q}\})\}\ Name\ \{\mathtt{R}\&\mathtt{Q}\},$$

where $\mathtt{P}, \mathtt{Q}, \mathtt{R} \in \mathbf{F}$.

It is considered that $\{false\}\ Prog\ \{false\}$ and $\{\mathtt{P}\}$ <*empty program*> $\{\mathtt{P}\}$ hold for all programs $Prog$ and all $\mathtt{P} \in \mathbf{F}$.

The following are examples of axioms for different groups of Forth words to demonstrate the meaning of defined predicates.

$\{S(l)\}$ Integer $\{S(l\ Integer)\}$ for numbers,

$\{S(l\ n_1\ n_0)\}\ SWAP\ \{S(l\ n_0\ n_1)\}$ for stack manipulation,

$\{S(l\ n\ a)\}\ !\ \{S(l)\&D(a, n)\}$ for memory management,

$\{S(l\ n)\&RS(k)\}\ {>}R\ \{S(l)\&RS(k\ n)\}$ for the return stack,

$\{S(l\ n)\}\ .\ \{S(l)\&out(n)\}$ for output stream,

$\{\}\ HEX\ \{D(BASE, 16)\}$ for system variables,

$\{S(l\ n)\&D(DP, a)\}\ ,\ \{S(l)\&D(DP, a + \mathrm{cell})\&D(a, n)\}$ for "comma" allocation,

$\{S(l\ n)\&instream(XXX, \mathrm{bl})\}\ CONSTANT\ \{S(l)\&indict(\ \{S(k)\}\ XXX\ \{S(k\ n)\}\ )\}$

The following four deduction rules are general

$$\frac{\mathtt{P} \supset \mathtt{P}';\ \{\mathtt{P}'\}\ Prog\ \{\mathtt{Q}\}}{\{\mathtt{P}\}\ Prog\ \{\mathtt{Q}\}} \qquad \text{Precondition strengthening}$$

$$\frac{\{\mathtt{P}\}\ Prog\ \{\mathtt{Q}'\};\ \mathtt{Q}' \supset \mathtt{Q}}{\{\mathtt{P}\}\ Prog\ \{\mathtt{Q}\}} \qquad \text{Postcondition weakening}$$

$$\frac{\{\mathtt{P}\}\ Prog1\ \{\mathtt{R}\};\ \{\mathtt{R}\}\ Prog2\ \{\mathtt{Q}\}}{\{\mathtt{P}\}\ Prog1\ Prog2\ \{\mathtt{Q}\}} \qquad \text{Sequential composition}$$

$$\frac{\{\mathtt{P}\}\ Prog\ \{\mathtt{Q}\};\ \{\mathtt{P}'\}\ Prog\ \{\mathtt{Q}'\}}{\{\mathtt{P}\&\mathtt{P}'\}\ Prog\ \{\mathtt{Q}\&\mathtt{Q}'\}} \qquad \text{Conjunction rule}$$

The *instream*-predicate is used for "scanner"-words

$$\frac{\{\mathtt{P}\&instream(string, delimiter)\}\ scanner\ \{\mathtt{Q}\}}{\{\mathtt{P}\}\ scanner\ string\ delimiter\ \{\mathtt{Q}\}} \qquad instream \text{ elimination rule}$$

Example 1. Derivation tree for the program 8 A ! , where A (— a)

$$\frac{\dfrac{\{S(l)\}\ 8\ \{S(l\ 8)\};\ \{S(l\ 8)\}\ A\ \{S(l\ 8\ a)\}}{\{S(l)\}\ 8\ A\ \{S(l\ 8\ a)\}}\ ;\qquad \{S(l\ 8\ a)\}\ !\ \{S(l)\&D(a,8)\}}{\{S(l)\}\ 8\ A\ !\ \{S(l)\&D(a,8)\}}$$

Forth colon definitions are introduced by rule

$$\frac{\{\mathtt{P}\}\ Prog\ \{\mathtt{Q}\}}{\{\mathtt{R}\}\ :\ Name\ Prog\ ;\ \{\mathtt{R}\&indict(\{\mathtt{P}\}\ Name\ \{\mathtt{Q}\})\}}$$

Control structures are also introduced by deduction rules.

$$\frac{\{\mathtt{P}\&S(l)\}\ Prog1\ \{\mathtt{Q}\};\ \{\mathtt{P}\&S(l)\}\ Prog2\ \{\mathtt{R}\}}{\{\mathtt{P}\&S(l\ flag)\}\ IF\ Prog1\ ELSE\ Prog2\ THEN\ \{\mathtt{Q}\&\mathrm{flag}\vee\mathtt{R}\&\neg\mathrm{flag}\}}$$

$$\frac{\{\mathtt{P}\}\ Prog1\ \{\mathtt{Q}\&S(l\ flag)\};\ \{\mathtt{Q}\&S(l)\}\ Prog2\ \{\mathtt{P}\}}{\{\mathtt{P}\}\ BEGIN\ Prog1\ WHILE\ Prog2\ REPEAT\ \{\mathtt{Q}\&S(l)\&\neg\mathrm{flag}\}}$$

"flag" in postcondition only gives information about the condition which generated $flag$ to the stack, formally we do not need this part added to the postconditions.

The definition of $DO...LOOP$ costructs looks more complicated.

$$\frac{\{\mathtt{P}(i-1)\&S(l_{i-1})\&RS(k\ sys\ b+1\ i)\&\ a\le i\le b\}\ Prog(i)\ \{\mathtt{P}(i)\&S(l_i)\&RS(k\ sys\ b+1\ i)\}}{\{\mathtt{P}(a-1)\&S(l_{a-1}\ b+1\ a)\&RS(k)\&\ a\le b\}\ DO\ Prog(i)\ LOOP\ \{\mathtt{P}(b)\&S(l_b)\&RS(k)\}}$$

$$\{\mathtt{P}\&S(l)\&RS(k\ sys\ b+1\ i)\&\ a\le i\le b\}\ I\ \{\mathtt{P}\&S(l\ i)\&RS(k\ sys\ b+1\ i)\}$$

$$\{\mathtt{P}\&S(l)\&RS(k\ sys\ b+1\ i)\&\ a\le i\le b\}\ LEAVE\ \{\mathtt{P}\&S(l)\&RS(k)\}$$

$\mathtt{P}(i)$ is an informal notation again (saying that $\mathtt{P}$ holds after execution of program $Prog(i)$ for the given value of loop index i). $\mathtt{P}(x)$ denotes $\mathtt{P}[x/i]$, i.e. the result of substitution of x for all occurances of i in $\mathtt{P}(i)$. It has to be emphasized that we do not have classical "variables" here (a, b, i are just notations only) and we cannot be precise in defining occurances, substitutions, etc.

The last rule introduces $CREATE...DOES>$ metaconstruction for new defining words.

$$\frac{\{\mathtt{P}_1\&S(l_1\ l_d)\&D(DP,a)\}\ Prog1\ \{\mathtt{Q}_1\&S(l_1)\};\ \{\mathtt{P}_2\&S(l_2\ l_u\ a)\}\ Prog2\ \{\mathtt{Q}_2\&S(l_2\ l_r)\}}{\{\mathtt{R}\}\ :\ Name\ CREATE\ Prog1\ DOES>\ Prog2\ ;\ \{\mathtt{R}\&indict(\{\mathbf{P}\}\ Name\ \{\mathbf{Q}\}\}}$$

where $\mathbf{P} = \mathtt{P}_1\&S(l_1\ l_d)\&D(DP,a)\&instream(XXX,\mathrm{bl})$

and $\mathbf{Q} = \mathtt{Q}_1\&S(l_1)\&indict(\{\mathtt{P}_2\&S(l_2\ l_u)\}\ XXX\ \{\mathtt{Q}_2\&S(l_2\ l_r)\})$

Example 2. Let us define a new defining word $CONST$ and use it to create a constant SIX.

: $CONST$ $CREATE$, $DOES>$ @ ; 6 $CONST$ SIX SIX .

What does the last dot operation print? We will use the "comma"-axiom, the "fetch"-axiom, the number axiom, the "dot"-axiom and the axiom scheme for the *indict*-predicate, also the rules for $CREATE...DOES>$, *instream* and general rules. The complete tree is too big to figure, we give three most important fragments of the proof. We need the following notations ("variables"): $k, l, m \in \mathbf{Z}^*$, $c \in \mathbf{Z}$, $XXX \in \mathbf{T}^*$ and $a \in \mathbf{A}$.

$$\cfrac{\cfrac{\{S(l\ c)\&D(DP,a)\}\ ,\{S(l)\&D(DP,a{+}\mathrm{cell})\&D(a,c)\}}{\{S(l\ c)\&D(DP,a)\}\ ,\ \{S(l)\&D(a,c)\}\ ;} \quad \cfrac{\{S(k\ a)\&D(a,c)\}@\{S(k\ c)\&D(a,c)\}}{\{S(k\ a)\&D(a,c)\}\ @\ \{S(k\ c)\}}}{\{S(m)\}\ :CONST\ CREATE\ ,\ DOES\!>\ @\ ;\ \Big\{S(m)\&indict\Big(\{\mathbf{P_C}\}\ CONST\ \{\mathbf{Q_C}\}\Big)\Big\}}$$

$\mathbf{P_C} = S(l\ c)\&instream(XXX, \mathrm{bl})$
$\mathbf{Q_C} = S(l)\&indict\big(\{S(k)\}\ XXX\ \{S(k\ c)\}\big)$

$$\cfrac{\{S(l)\}6\{S(l\ 6)\};\quad \cfrac{\big\{S(l\ 6)\&instream(SIX,\mathrm{bl})\big\}CONST\big\{S(l)\&indict\big(\{S(k)\}SIX\{S(k\ 6)\}\big)\big\}}{\{S(l\ 6)\}\ CONST\ SIX\ \big\{S(l)\&indict\big(\{S(k)\}\ SIX\ \{S(k\ 6)\}\big)\big\}}}{\big\{S(l)\big\}\ 6\ CONST\ SIX\ \big\{S(l)\&indict\big(\{S(k)\}\ SIX\ \{S(k\ 6)\}\big)\big\}}$$

$$\frac{\big\{S(m)\&S(k)\&indict\big(\{S(k)\}SIX\{S(k\ 6)\}\big)\big\}\ SIX\ \{S(m\ 6)\};\{S(m\ 6)\}\ .\ \{S(m)\&out(6)\}}{\{S(m)\}\ SIX\ .\ \{S(m)\ \&\ out(6)\}}$$

The first fragment illustrates $CREATE ... DOES$>-rule, the postcondition weakening rule is used twice. The second fragment is not complete, there has to be *indict*($...CONST...$) as a precondition. This technique is demonstrated on the third fragment for the word SIX. If we want to build the full tree, we have to use the composition rule many times.

We have answered the initial question now, the last dot operation will print "6". The data stack state does not change after execution of the program and there will be the new specified words $CONST$ and SIX in the dictionary.

These have been quite random ideas about introducing the Floyd-Hoare semantics for the Forth language. The real examples are too complex to be proved by hand, it is assumed that there is some computer aided tool to do the technical work. Sometimes we need fragments of proofs to understand what the program really does, sometimes such rules are the only short and unambigous way to explain new constructs. Author hopes that this technique will be useful also in program construction and synthesis.