数理解析研究所講究録
236巻 1975年 38-45

A Formal Semantics For Algorithmic Languages Based On The Scott's Logic

K. Mukai, S. Sekimoto, and M. Sudo
(Mitsubishi Electric Corp.)

Abstract A semantics of algorithmic languages are given applying the least fixed point theory. The language to be studied in this paper is a subset of ALGOL60, which is almost the full set of ALGOL60, but without goto statements.

Mainly, meanings of mutually defined recursive procedure with parameters are described. A lattice theoretical foundation for this semantics is seen in the appendix.

§1 Introduction

In Scott[1],[3] and Strachey[1], mathematical except (7) and (8) domains are introduced to describe the meanings of the source language. The list of those is (1) T: truth values (2) N: numbers (3) S: states (4) L: locations (5) P: procedures (6) C: commands (7) Id: identifiers (8) Env: environments.

These domains satisfy the next relations. a) $V=T+N+C+P+L$ b) $P=[V\to[S\to V\times S]]$ c) $C=[S\to S]$ d) $Env=[Id\to C+V]$. Here X+Y means direct sum of continuous lattices X and Y, and also X+Y becomes a continuous lattice. $[X\to Y]$ denotes the set of all continuous functions from X to Y.

The syntax of the source language studied in the above papers is divided into two categories, that is, COMMANDS (Cmd) and EXPRESSIONS (Exp). And then, two functions $\mathcal{C}$ and $\mathcal{E}$ are constructed recursively.

(1) $\mathcal{C}$: Cmd $\rightarrow$ [Env $\rightarrow$ [S$\rightarrow$S]]

(2) $\mathcal{E}$: Exp $\rightarrow$ [Env $\rightarrow$ [S$\rightarrow$V $\times$ S]]

Meanings of γ in Cmd and ε in Exp are $\mathcal{C}[\![\gamma]\!]$ and $\mathcal{E}[\![\varepsilon]\!]$ respectively. Detailed explanations of Cmd, Exp, $\mathcal{C}$, and $\mathcal{E}$ are, however, ommitted here. (See [1] or [3]).

§2 Recursive Procedures and Block Structures

The source language $\mathcal{A}$, whose semantics we are going to study, is the largest subset of ALGOL60 such that $\mathcal{A}$ is without (1) goto statements and labels, (2) for statements, and (3) switch, own, and array.

2.1 Procedures

We have to modify b) in §1 into b') because in the language $\mathcal{A}$, a number of parameters of a procedure may be taken to be arbitrarily finite, and procedures are not restricted to function type. That is, also command type procedures are allowed.

b') $P = c + [S \rightarrow V \times S] + [V \rightarrow P]$

2.2 Call by name and call by value

The language $\mathcal{A}$ allows a flexible parameter passing i.e. call by name and call by value. The format of a procedure declaration is:

⟨ identifier ⟩ (a,b,....,c: x,y,....,z) ⟨ procedure body ⟩ ,

where a,b,....,c and x,y,....,z are formal parameters called by value and name respectively. And the format of a procedure statement is:

Call ⟨ identifier ⟩ (e_1, ---, e_m : f_1, ---, f_n) ----- (α)

2.3 A semantics of procedure statements

We describe a semantics of a procedure statement α step by step. For a given σ in S and ρ in Env,

(1) $\mathcal{E}[\![e_1]\!](\rho)(\sigma) = \langle v_1, \sigma_1 \rangle$

- - - -

$\mathcal{E}[\![e_m]\!](\rho)(\sigma_{m-1}) = \langle v_m, \sigma_m \rangle$

(2) $\rho' = \rho[v_1/a, \cdots, v_m/c, \mathcal{E}[\![f_1]\!](\rho)/x, \cdots, \mathcal{E}[\![f_n]\!](\rho)/z]$

(3) finally $\sigma' = \mathcal{C}[\![\langle\text{procedure body}\rangle]\!](\rho')(\sigma_m)$

<u>def. 1</u> $\mathcal{C}[\![\alpha]\!](\rho)(\sigma) \triangleq \sigma'$

We have to suppose that ρ has two facilities: briefly, (i) a facility of "copy rule" for identifiers in a given source program in α , and (ii) a dynamic memory allocation without destructing any necessary area.

2.4 A semantics of function procedures

For simplicity, we consider a function procedure, whose formal parameters are only a and x, and they are called by value and name respectively. We describe a meaning of next (β) as in section 2.3.

$\chi(e, f)$ (χ in Id) ----- (β)

For given ρ in Env and σ is S,

(1) $\mathcal{E}[\![e]\!](\rho)(\sigma) = \langle v_1, \sigma_1 \rangle$

(2) $\rho' = \rho[v_1/a, \mathcal{E}[\![f]\!](\rho)/x]$

(3) $\sigma' = \mathcal{C}[\![\langle\text{procedure body}\rangle]\!](\rho')(\sigma_1)$

(4) Contents $(\rho(\chi), \sigma') = v'$, (see [1] or [3]), so we obtain

<u>Def. 2</u> $\mathcal{E}[\![\beta]\!](\rho)(\sigma) \triangleq \langle v', \sigma' \rangle$

2.5 Block structure

Blocks are considered to be procedures with no parameters. For a source program π in $\mathcal{Q}$, let $B(\pi)$ be the set of all occurrences of blocks in π. Each occurrence of a block in π is implicitly supposed to have an identifier in Id. That is, $B(\pi) \hookrightarrow$ Id (inclusion map is, say, i_π). Then for each b in $B(\pi)$ and ρ in Env, $\rho(i_\pi(b))$, or in short, f(b), is in C+V.

It may be more useful to introduce a few notions before we finally describe a meaning of π. Let D be a continuous lattice.

def. 3 $\mathcal{J}(D)$ is the set of all trees over D.

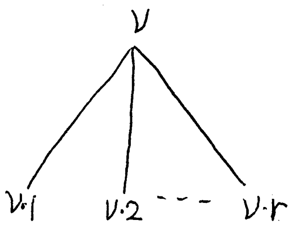
def. 4 For s and t in $\mathcal{J}(D)$, $s \leq t$ is true if and only if dom(s)= dom(t), and $s(\nu) \sqsubseteq t(\nu)$ for all ν in dom(s), where $\sqsubseteq$ is the order relation in D.

def. 5 Let $F = \bigcup_{n=0}^{\infty} [D^n \to D]$, and each element in F is supposed to be assigned a rank in the apparent way.

Hereafter, a tree f over the ranked set F is fixed.

def. 6 $\mathcal{J}(D,f) \triangleq \{ t \in \mathcal{J}(D) \mid dom(t) = dom(f) \}$

def. 7 A transformation $\Phi(f)$ on $\mathcal{J}(D, f)$ is defined as follows,



i.e $t' = \Phi(f)(t)$ is true if and only if $t'(\nu) = f(\nu)(t(\nu\cdot 1), ---, t(\nu\cdot r))$ for each ν in dom(f). (A tree is a function from a tree domain to a set of labels.)

def. 8 By Fix $(\Phi(f))$, the least fixed point of above $\Phi(f)$ is denoted.

2.6 A semantics of a source program π in $\mathcal{A}$

In this section, we give, finally, a definition of a meaning of a source rpogram π in $\mathcal{A}$.

(1) First, transform π into a tree f over F precisely preserving the block structure of π, where $F = \bigcup_{n=0}^{\infty} [C^n \to C]$, and ($C = [S \text{—} S]$).

For given ρ in Env, it is not so difficult to construct a tree f($=f_{\pi,\rho}$) mentioned above. For example, let's consider a block (#).

(#) begin if ε then begin A end else B end end

The rule corresponding to (#) is

λ (u, v) Cond (u, v) * $\mathcal{E}[\![\varepsilon]\!](\rho)$ in $[C \times C \to C]$. We suppose a construction of such $f_{\pi,\rho}$ (=f for brief)

(2) Second, now we are able to get $\mathcal{T}$(c, f), Φ(f), and Fix (Φ(f)) as in def. 6, def. 7, and def. 8 respectively. Let θ_0 in C be the label of the root of Fix (Φ(f)). And, at last, $\mathcal{C}[\![\pi]\!](\rho)$ is defined to be θ_0.

def. 9 $\mathcal{C}[\![\pi]\!](\rho) \triangleq \theta_0$

§3 Conclusion

The method described in §2 is an extension of one used in such an example as follows.

example:

FACT(x):= if x=0 then 1 else x*FACT (x-1)

The least solution of this example is given to be $\lim_{n\to\infty} f^{(n)}$, where $f^{(0)} \triangleq$ totally undefined function, and $f^{(n+1)}(x) \triangleq$ if x=0 then else x*$f^{(n)}$(x-1) for $n \geq 1$.

Without an assumption of memory (or state vectors) of a machine and a stack manipulation, it is, we think, almost impossible to describe a meaning of mutually recursive procedures together with assignment statements. And a detailed description of ρ in Env corresponds to the stack manipulation or the dynamic memory allocation.

One of motivations of this essay is to give a formal description of a source program in $\mathcal{A}$ in a bit more syntax-directed method. The def. 7 in 2.5 reminds us of D. Knuth's semantics theory for context-free languages. We have given in this paper only an outline of semantics of $\mathcal{A}$. More detailed discussions must be done.

Acknowledgements

We are grateful both to Prof. T. Kasami of the Osaka Univ. and Dr. K. Torii at ETL., for their helpful advices to our study.

Reference

(1) D. Scott and C. Strachey "Toward A Mathematical Semantis For Computer Languages." (Computers and Automata, Brooklyn Inst.

(2) D. Scott "Continuous Lattices" (Proc. Dalhousie Conf., Springer Lecture Note)

(3) D. Scott "Mathematical Concept in Programming Languages" (SJCC, 1972)

(4) D. Scott "The Lattices of Flow Diagrams" (Springer Lecture Notes in Mathematics 188, 311-366 (E. Engeler, Ed., 1971))

(5) P. Naur (Ed.) "Report on the algorithmic languages ALGOL60" CACM3 (1960)

Appendix.

We show in this appendix an outline of a proof for a proposition on which our work is mainly based. From here on, a continuous lattice is restricted to such a space whose greatest element is isolated in the sense of topology.

Proposition A. For given continuous lattices P_0, V_0, and S there exist continuous lattices P and V which satisfy the relations a1) and a2):

a1) $V \cong V_0 + P$,

a2) $P \cong P_0 + [S \to V \times S] + [V \to P]$.

Here $X \cong Y$ means that there exists some one-to-one and onto correspondence between continuous lattices X and Y under which X and Y are both isomorphic and homeomorphic in the sense of lattice and topological space, respectively.

The proposition A is a result of a series of lemmas listed below.

Lemma 1. i) If X and Y are continuous lattices, then X+Y, $X \times Y$, and $[X \to Y]$ are also continuous lattices.

ii) If $\langle X_n, j_n \rangle_{n \geq 0}$ is an inverse system of continuous lattices such that for each $n \geq 0$ j_n is a projection, then the inverse limit $\varprojlim \langle X_n, j_n \rangle_{n \geq 0}$ is also a continuous lattice.

Lemma 2. If X and Y are continuous lattices, then both X and Y are projections of X+Y.

Lemma 3. If X_1, X_2, Y_1, and Y_2 are continuous lattices, and X_1 and X_2 are projections of Y_1 and Y_2 respectively, then next holds.

i) X_1+X_2 is a projection of Y_1+Y_2,

ii) $X_1 \times X_2$ is a projection of $Y_1 \times Y_2$,

iii) $[X_1 \to X_2]$ is a projection of $[Y_1 \to Y_2]$.

<u>Lemma 4.</u> If $\langle X_n, j_n \rangle_{n\geq 0}$ and $\langle Y_n, k_n \rangle_{n\geq 0}$ are inverse systems such that for each $n \geq 0$, both j_n and k_n are projections, then next holds.

i) $\varprojlim \langle X_n+Y_n, j_n+k_n \rangle_{n\geq 0} \cong \varprojlim \langle X_n, j_n \rangle_{n\geq 0} + \varprojlim \langle Y_n, k_n \rangle_{n\geq 0}$,

ii) $\varprojlim \langle X_n \times Y_n, j_n \, k_n \rangle_{n\geq 0} \cong \varprojlim \langle X_n, j_n \rangle_{n\geq 0} \times \varprojlim \langle Y_n, k_n \rangle_{n\geq 0}$,

iii) $\varprojlim \langle [X_n \to Y_n], j_n \to k_n \rangle_{n\geq 0} \cong [\varprojlim \langle X_n, j_n \rangle_{n\geq 0} \to \varprojlim \langle Y_n, k_n \rangle_{n\geq 0}]$.

Here, for each $n \geq 0$ j_n+k_n is the canonical projection from $X_{n+1}+Y_{n+1}$ to X_n+Y_n derived from j_n and k_n, and similar is the case with either $j_n \times k_n$ or $j_n \to k_n$.

<u>(An outline of a proof for the proposition A).</u>

First, construct two inverse systems $\langle P_n, j_n \rangle_{n\geq 0}$ and $\langle V_n, k_n \rangle_{n\geq 0}$. This is all right, i.e., P_n $(n \geq 0)$ is defined by an inductive method as follows.

$$P_{n+1} = P_0 + [S \to V_n \times S] + [V_n \to P_n],$$

$$V_{n+1} = V_0 + P_n.$$

Of course, using lemma 1, 2, and 3, for each $n \geq 0$ both j_n and k_n can be taken as projections.

Second, the pair of $P = \varprojlim \langle P_n, j_n \rangle_{n\geq 0}$ and $V = \varprojlim \langle V_n, k_n \rangle_{n\geq 0}$ is easily checked to satisfy the relations a1) and a2) by applying the lemma 4.