The article discusses the application of lambda calculus to the formalization of the semantics of an abstract programming language based on certain constructs of the L2 language. The L2 language has been developed at the Institute of Cybernetics of the Academy of Sciences of the UkrSSR [1] and provides extended facilities for the solution of logical problems, in particular problems in automatic design.

The L2 language and the technique of its application to programming are described in [1-3].

1. Language Description

A program is composed of statements and is divided into blocks which can be imbedded in each other. Each block begins with the statement BEGIN and ends with the statement END. Other statements are subdivided into descriptions and operators. Statements are separated from each other by semicolons. The environment of a statement in a program is defined by a list of all variables declared in blocks containing the given statement. The part of the language which differs from ALGOL is described below.

1.1. Compounds Objects. Consider an abstract ALGOL-like programming language which includes, in particular, compound objects and operators acting on them.

A compound object is a labeled oriented tree. For any compound object are uniquely defined the label \( u \) of its starting node, the arity \( n \) of this label, and the objects \( A_1, \ldots, A_n \) directly subordinate to the object \( A \). Objects of arity 0 are primary objects. The definition of a compound object does not exclude the case when a tree is infinite. As usual in ALGOL-like languages, any compound object \( A \) to be used in a block must be described at the beginning of the block. This is done with the aid of a COMPOUND \( A \) statement.

Programming in the L2 language is based on the use of pointers. A pointer can be either fixed or not. A fixed pointer views the beginning of the compound object on which it is fixed. The value of a pointer is the compound object whose beginning the pointer views. The use of a pointer \( U \) in a block must be described at the beginning of the block by means of the statement COMPOUND POINT \( UR \).

1.2. Language Operators and Operations for Acting on Compound Objects

If \( A \) is a compound object then OPERATION (\( A \)) is a label of its root node and ARITY (\( A \)) is the arity of the label of its root node.

Assignment operator \( A := E \), where \( A \) is an expression with a value in the class of names of compound objects, and \( E \) has a value in the class of compounds.

*First part of article published in Kibernetika, No. 5, pp. 19-23 (1981).

Assignment operator $A := E$, where $A$ is an expression with a value in the class of pointers and $E$ has a value in the class of compounds. The effect of the assignment operator is obvious and is described in [2].

Let $A_1$, $A_2$ be compound objects. Then the expression $A_1 \leq A_2$ has the value TRUE if and only if $A_2$ contains a subobject strictly equivalent to $A_1$ [2].

Consider the analysis operator $\text{ANALYSIS } X = E [, \text{WHERE } B_1, \ldots, B_n]$, where $X$ is a pointer of a compound object or the name of a compound object, and $E$ is an expression of the compound type. The expression $E$ and the conditions $B_1, \ldots, B_n$ can contain the names of compound objects which have no value and also nonfixed pointers. Execution of the operator consists in an analysis of the possibility of assigning values to names which have no value and of fixing nonfixed pointers so that the equality $X = E$ and the conditions $B_1, \ldots, B_n$ are satisfied. If this is found possible, the names are given the respective values and the pointers are fixed so that the variable YES obtains the value 1; otherwise all things remain unchanged and the variable YES is assigned the value 0.

1.3. Operators Acting on Pointers. The effect of each operator is described in [2].

The effect of each operator is described in [2]. Their action is illustrated with a series of drawings. The drawings on the left and right show the states before and after operator execution, respectively. The fact that pointer $V$ views a compound object whose root node is $a$ is denoted by $X \Rightarrow A$. FIX $X$ AT $U$ (here $U$ is a pointer):

\[
\begin{align*}
S & \quad S H I F T [U P W A R D] [P O I N T E R] X \\
S & \quad S H I F T [R I G H T] [P O I N T E R] X \\
S & \quad D E S C E N D X, i \\
S & \quad \text{After execution of the operator REMOVE [POINTER] X, pointer X will be undetermined.}
\end{align*}
\]

For the sake of convenience the language uses standard logical variables out of which we select two: YES and NO [2].

We shall assume that node labels are lambda terms. For the operators mentioned above we do not consider situations in which before the operator SHIFT RIGHT POINTER $X$ is executed, $X$ is located as shown in Fig. 1.

In the following discussion all states before operator execution are assumed to be correct in the sense of providing a unique resultant state.