This is a survey of the last year's work of the group of algorithmic logic. Our studies have concentrated on two (not disjoint) tasks:
- design of programming language LOGLAN 77,
- studies of computational complexity.

The questions of data structures implementation and of non-sequential computing processes were the main aims of our research. The work devoted to LOGLAN language has paid back handsomely in discovering possibilities of algorithmic approach to the foundations of set theory. The survey ends with a short presentation of an early stage of research connected with P=NP problem.

§1 LOGLAN 77

Here I am going to present some intuitions and key words rather than LOGLAN language itself.

The language LOGLAN 77 is designed to be a universal programming language containing all necessary programming tools. Its closest relative is SIMULA 67 [2]. LOGLAN has: a small syntax, a precise semantics, possibility of multiprocessor execution, seven ways of transmitting parameters, unified version of assignment instruction, possibility of assigning expressions as the contents of variables, algebra of type expressions with array, class and prefix operations. Types are patterns for objects. By an object we understand a pair:
<valuation of attributes, list of instructions>. A collection of objects may constitute a configuration. Execution of a program results in the sequence of configurations forming a computation. Some examples of LOGLAN programs are given below. Here we shall discuss the relation
in between objects and types. We shall say that an object o is of type T iff 1° declaration of type T contains exactly those attributes which form the domain of the valuation of the object o, 2° values of attributes are in corresponding, declared types, 3° list of instructions of o is compatible with the body of type T declaration. Let r be the relation: type T is prefixed by a type T'. Let the relation \( r^* \) be the transitive closure of r. Then the relation in is defined by the equivalence:

\[
o \in T \Leftrightarrow (E T') T r^* T' \land o \in T'
\]

Examples:

Let T be a

\[
type T : \text{class}(a:T_1, b:T_2); \text{begin variable } c:T_3; c:=a \text{ end}
\]

and o an object

\[
\begin{array}{c|c|c}
  a & b & c \\
  \hline
  o_1 & o_2 & o_3
\end{array}
\]

then o is T iff

\[
o_1 \in T_1 \text{ and } o_2 \in T_2 \text{ and } \left( o_3=\text{none} \land I= \text{c:=a or o}_3=\text{c:=I} \land I=\emptyset \right)
\]

The notion of binary tree can be defined as follows:

\[
type \text{bintree} : \text{class}(l,r : \text{bintree}); \text{ Set of objects of bintree type contains}
\]

\[
o : \left( \begin{array}{c|c|c|c}
  1 & r & \emptyset \\
  \hline
  \text{none} & \text{none} & \emptyset
\end{array} \right)
\]

The first object represents a leaf, the second a two-element tree. Let elem be a type, then the notion of tree with vertices labelled by elem objects is defined as follows

\[
type \text{labelled bintree} : \text{bintree class}(\text{val : elem});
\]

An example of an object of type labelled bintree is

\[
\left( \begin{array}{c|c|c|c|c}
  1 & r & \text{val} & \emptyset \\
  \hline
  t_1 & t_2 & e & \emptyset
\end{array} \right)
\]

where t_1, t_2 are again in bintree and e in elem. This object is labelled bintree object and in bintree type.

The modest syntax of LOGLAN is satisfactory enough to extend the language to the desired size, data structures and functions due to the 1° prefixing, 2° allowing instructions to be parameters.