One of the most attractive features and an often-heralded advantage of functional programming languages is that they are not sequential in their nature, as conventional imperative languages are. The parallelism in a functional program is implicit, and it is manifested solely through data dependencies and semantics of primitive operators. This is in contrast to more conventional parallel languages, where explicit constructs are typically used to invoke, synchronize, and in general coordinate the concurrent activities.

Some parallel functional programming languages were building following the idea of concurrent execution of programs without adding new language constructs or detailed program tuning. Many of them were in fact developed simultaneously with work on highly parallel dataflow and reduction machines. In all of these efforts the parallelism in a program is detected by the system and allocated to processors automatically.

An alternative approach is to have a number of processing elements connected together with some kind of network, each independently executing its own program. The problem of dividing the task up into concurrent subtasks, programming these subtasks, and arranging the inter-task communication is left entirely to the programmer. But, often a programmer knows a particularly good, and perhaps probably optimal decomposition and distribution strategy for a program executing on a particular machine, but one can never expect a compiler to find out such optimal mappings always. Although complete user unawareness is a noble goal, we cannot ignore the needs of those who wish to express the decomposition and mapping themselves.

To meet these needs a class of Para-functional programming languages was introduced by Paul Hudak. This class is build to provide an extra degree of expressiveness through annotations to the source program. The term Para-functional is intended to convey not only parallel computation, but also the fact that the annotations provide an operational semantics that is truly extra or beyond the functional semantics of the language. In this research the formal semantics of a Para-functional programming extension of the A_LispKit Lisp is investigated in depth. This extension works on local area networks based on PC computers, on which we have a network of simulators of SECD machines.

**Key words:** Functional programming language, Operational semantics, Parallelism.

REFERENCES


