# Formal Semantics for PLEX

Johan Erikson & Björn Lisper
Department of Computer Science and Electronics
Mälardalen University
{johan.erikson, bjorn.lisper}@mdh.se

In any system with shared data and concurrent (or independent) activities, there is a need to guarantee exclusive access to the shared data. A system designed for parallel processing handles this by synchronizing the access to the shared data. But if parallel processing, and synchronization, wasn't an issue at the time of designing the system, non-preemptive execution on a single-processor architecture, automatically guarantees exclusive access to the shared data. We denote the software in the second case as 'sequential software'.

While legacy software systems, developed and maintained over many years, contains large amounts of sequential software (executed on single-processor architectures), there is a development towards different forms of parallel hardware. The problem arises when the single-processor architecture is to be replaced by a multi-processor ditto. At this point, the non-preemptive execution does not protect the shared data any longer, since independent parts, still executed in a non-preemptive fashion, but on different processors, may now access and update the same data concurrently. The question is: *How is such a system to be parallelized?*

A naive solution would be to re-implement the system, but since a legacy software system may contain several million lines of code, this solution is infeasible. A more reasonable solution would be criteria that ensures when functional equivalence, in some central aspect, is preserved for the existing software when it, without any changes, is executed on a parallel architecture. To ensure the correctness of such criteria, the formal semantics of the language in question need to be considered.

Our subject of study is the language PLEX, which is used to program the functionality in the AXE telephone exchange system from Ericsson. All the above properties: independent activities, shared data, non-preemptive execution, and a single-processor architecture are present in the system.

Besides an asynchronous communication paradigm, PLEX is an imperative language with assignments, conditionals, goto's, and a restricted iteration construct (which only iterates between given start and stop values). It lacks some common statements from other programming languages such as `WHILE` loops, negative numeric values and real numbers. A PLEX program file (called a *block*) consists of several, independent *sub-programs*, which can be executed in any order, together with block wise scooped data.

Since the semantics for PLEX previously has been defined through its implementation, the formal semantics of the language has to be defined before any criteria could be stated. In a recent paper [EL05], we have presented two versions of a small steps operational semantics (in the styled used in [NN92]) for the language, and in which the execution of statements is modeled as state transitions. The first version models the existing source code executed by the current single-processor architecture, whereas the second models the execution on an experimental, multi-threaded, shared-memory architecture. The parallel architecture, and its exe-

cution model, is designed to be functionally equivalent with the single-processor architecture in the sense that is should be possible to execute the old software in the parallel environment without any changes in the existing source code. The price to pay in order to achieve this (no changes to the existing code), is a restricted execution model which only allows parallel execution for unrelated activities. Related activities, on the other hand, are forced to execute in the same sequential order as in the single-processor case. Exclusive access to the shared data is ensured by the run-time system, which utilizes a number of binary locks to prevent simultaneous access to the same block.

Since the current parallel implementation is rather restricted, in that it only allows one thread at the time to execute code in a block, the continuation of our work will add primitives for synchronization to PLEX, thereby defining the language "Parallel-PLEX". We will also specify the semantics for this extended language, and from there continue with the criteria that ensures safe parallel execution in a less restricted execution model. Future work will also include a formal definition of the term 'functional equivalence', as well as proving that the property holds between the different semantics.

# References


[EL05] J. Erikson and B. Lisper. Two Formal Semantics for PLEX. In *Proceedings of the 3nd APPSEM II Workshop, APPSEM'05*, Frauenchiemsee, Germany, 13-15 September 2005.

[NN92] H. R. Nielson and F. Nielson. *Semantics with Applications: A Formal Introduction*. John Wiley & Sons, 1992.