Grainless Concurrency Semantics for Monitors

Kamal Lodaya[1] and Uday S. Reddy[2]

[1] Institute of Mathematical Sciences, Chennai
[2] School of Computer Science, University of Birmingham

The early work on structuring concurrent programs by Dijkstra, Hoare and Brinch Hansen [2, 5, 6] has emphasized the need for concurrent processes to operate in disjoint portions of memory without interference from each other. Communication between processes is organized through shared resources controlled by mutual exclusion protocols. Monitors, formulated by Brinch Hansen and Hoare [2, 6] form a well-developed mechanism for such sharing of resources. They underlie the concurrency features of practical languages like Concurrent Euclid and Java.

In contrast, the semantic models of concurrent programming languages have been largely devoted to studying shared variable concurrency (e.g., Brookes [3]) using interleaving of traces where the independence of concurrent processes is not represented. In recent work, Brookes [4] has taken a few initial steps in this direction in his modelling of Separation Logic proof rules for conditional critical regions. Building on this work, Reynolds [11] has formulated a semantics for shared variable programs representing the independent operation of concurrent process as long as their operation is independent. He terms such independent operation “grainless concurrency” (to mean essentially what is traditionally meant by “true concurrency.”)

In the present work, we focus on the semantics of concurrent programs with monitors where we try to represent grainless concurrency in a rather direct way. To focus on this aspect, we ignore the issues of heap storage and treat only local variables. This models a language like Concurrent Pascal [2].

To illustrate how the semantics works, consider the concurrent composition of two commands C_1 and C_2 communicating through a shared monitor M. Other than the monitor, we assume that there are no common free identifiers between C_1 and C_2. This is similar to the syntactic control of interference [10] used in Concurrent Pascal. The monitor consists of publicly visible methods, which we assume to be first-order procedures with call by value and/or result parameter passing.

We use a trace-based semantics to denote the externally observable behavior of the monitor, as in Abramsky-McCusker games [1]. For each procedure p in M, there are events (or “moves”) of the form:

$$p.\mathbf{call}(v_1, \ldots, v_n)$$
$$p.\mathbf{return}(v'_1, \ldots, v'_m)$$

where $v_1, \ldots, v_n$ are arguments passed as value-parameters and $v'_1, \ldots, v'_m$ are the values returned as the result-parameters. A **return** event in a trace must be enabled by a prior **call** event.

The meaning of each command C_i (for $i = 1, 2$) is expressed as a set of triples

$$(t_i, s_i, s'_i)$$

where t_i is a trace of events for the shared monitor and s_i and s'_i are states for the portion of the store accessed by C_i (representing respectively the initial and final states of an execution). We are ignoring nontermination in this abstract.

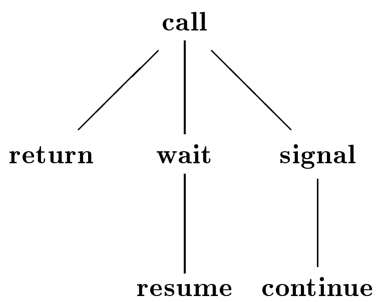
The meaning of concurrent composition is then expressed as follows:

$$\begin{aligned}[\![C_1 \parallel C_2]\!] = \{\, (t,\ s_1 * s_2,\ s'_1 * s'_2) \mid \\ \exists t_1, t_2.\ t \in \mathit{interleavings}(t_1, t_2) \wedge \\ (t_1, s_1, s'_1) \in [\![C_1]\!] \wedge (t_2, s_2, s'_2) \in [\![C_2]\!] \,\}\end{aligned}$$

(Here, $s_1 * s_2$ denotes the join of disjoint states s_1 and s_2.) The following facts may be gleaned from this example:

- The action of each process on its portion of storage is represented as an indivisible state transformation with no need for interleaving.
- Interleaving is only used for the actions on the shared monitor.
- The semantics is a conservative extension of that of sequential programs, i.e., if no shared monitors are used, then the semantics of a process is the same as its semantics in a sequential programming language.

The trace sets of (sequential) objects with first-order procedures can be calculated as in [1, 8, 9]. However, monitors have condition-synchronization achieved using wait and signal statements. To model them, we use additional events in the game tree of monitors, as depicted in:



The **wait** and **signal** events are processed by a scheduler which, in turn, issues **resume** and **continue** events. By composing a monitor with a scheduler, we obtain a trace with only **call** and **return** events, which represents the behavior of a monitor as observable by a client process.

In summary, we define a semantics for a concurrent programming language with monitors and first-order procedures, which represents the independent – "grainless" – operation of the concurrent processes.

References

1. Abramsky, S., and McCusker, G. Linearity, sharing and state. In *Algol-like Languages* [7], ch. 20.

2. Brinch Hansen, P. *Operating Systems Principles.* Prentice Hall, 1973.
3. Brookes, S. D. The essence of Parallel Algol. In *Proceedings, Ninth Annual IEEE Symposium on Logic in Computer Science* (July 1996), IEEE Computer Society Press, pp. 164–174. Reprinted as Chapter 21 of [7].
4. Brookes, S. D. Semantics and Separation Logic for parallel programs with shared mutable data. Manuscript, 2003.
5. Dijkstra, E. W. Hierarchical ordering of sequential processes. In *Operating Systems Techniques*, C. A. R. Hoare and R. H. Perrott, Eds. Academic Press, 1972, pp. 72–93.
6. Hoare, C. A. R. Monitors: An operating system structuring concept. *Comm. ACM 17*, 10 (Oct. 1974), 549–558.
7. O'Hearn, P. W., and Tennent, R. D. *Algol-like Languages (Two volumes).* Birkhäuser, Boston, 1997.
8. Reddy, U. S. Global state considered unnecessary: An introduction to object-based semantics. *J. Lisp and Symbolic Computation 9* (1996), 7–76. (Reprinted as Chapter 19 of [7]).
9. Reddy, U. S. Objects and classes in Algol-like languages. *Information and Computation 172* (2002), 63–97.
10. Reynolds, J. C. Syntactic control of interference. In *ACM Symp. on Princ. of Program. Lang.* (1978), ACM, pp. 39–46. (Reprinted as Chapter 10 of [7]).
11. Reynolds, J. C. Towards a grainless semantics for shared variable concurrency. In *ACM Symp. on Princ. of Program. Lang.* (New York, 2004), ACM. (Abstract of an invited lecture).