Formal Methods
Semantics of Concurrency

Nachum Dershowitz

May 2000

In *interleaving* semantics, one considers the execution of a parallel program $A \parallel B$ to be equivalent to the union of all possible interleavings of the execution sequences $A_0; A_1; A_2; \cdots$ and $B_0; B_1; B_2; \cdots$ of atomic statements of A and B.

This approach is not conducive to compositionality.

For concurrent programs, it is convenient to view the program as a labelled state-transition relation, that is, as a possibly infinite graph, called the *process graph*, with nodes Q called "states" and edges labelled by "actions" A. There is a distinguished *root* node. The "one-step" relation τ is a subset of $Q \times A \times Q$ and describes the possible state transitions and associated actions. A *computation tree* is a possibly infinite tree of state-transitions beginning at the root; a *computation* is a path in such a tree.

A concurrent programming language can be defined in terms of its labelled transitions.

In *branching-time* semantics, one interprets processes as their computation trees. In *linear-time* semantics, one interprets processes as the set of all their computations.

We seek a notion of equivalence of processes which looks at the possible sequences of actions but takes into account the possibility of deadlocks.

Two processes P and R are *bisimular* if for every action $a \in A$ the subtrees of P following a are each equivalent to some subtree of R following a and vice-versa. The (smallest) bisimularity relation exists and is an equivalence.

Complications enter when one allows "stutter" via "silent", action-less transitions.

In the *algebraic* approach to semantics of concurrency, one gives axioms for programming constructs, such as

$$P \parallel (S \parallel R) = (P \parallel S) \parallel R$$

When the number of states is large or infinite, the state-transition τ is often described by a set of formulas that refer to state-variable values and program-statement labels.

In the *computation tree logic* CTL* one has the following operators for describing properties of computation trees: **A** (for all paths), **E** (for some path), **G** (always), **F** (sometimes), **X** (next time), **U** (until), **V** (unless).

Fairness of scheduling excludes certain computation paths from consideration, because of the infinite occurrence of some situation along the path.

More generally, one can use second-order monadic logic to describe temporal properties of concurrent programs.

For compositionality, we want to be able to describe interactions between processes. There are two forms of interaction: synchronous and asynchronous.

There are two means of controlling interaction: via shared registers or via shared communication channels. Thus, we can speak of a *shared memory* models or *message passing* models with shared *ports.*

To model *continuous* time, many new issues arise.