Chapter 2

Modeling Real Concurrency

An essential element of the generic model used in this book is that concurrency is represented by *interleaving*. This means that, according to the formal model, two parallel processes never execute their statements at precisely the same instant, but take turns in executing atomic transitions. Formally, when one of them executes an atomic transition, the other is inactive. This model of computation is very convenient for the formalization, analysis, and manipulation of concurrent programs.

However, actual concurrent systems are usually composed of several independent processors, each of them executing a program of its own (a *process* in our terminology). In such systems, the execution of statements in different processors usually overlaps rather than interleaves. We refer to this behavior as *overlapped execution*.

A crucial question is how to reconcile the formal notion of interleaved computations, defined in our models, and the notion of overlapped executions, as realized on actual systems.

Two problems have to be resolved to achieve this reconciliation. The first problem is that of interference, and the second is independent progress.

- **Interference**
  Interleaved execution provides a higher degree of protection from interference than is available in overlapped execution. This is because interleaved execution requires that when a transition is taken, all other transitions are inactive, so no interference during a transition is possible.

  Consider, for example, the following statement:

  \[
  \text{when } y = y \text{ do } S. 
  \]

  In interleaved computations, the condition \( y = y \) is tested in one atomic step, so it always yields the value \( T \). In comparison, overlapped execution (under naive implementation, with no optimization) of this statement will reference \( y \) twice.
If, precisely between these two references, an overlapped statement performed by a concurrent processor changes $y$, the testing processor may find the condition to be false.

There are two possible solutions to the interference problem: we can either admit more interference in the interleaved computations or require more protection in the overlapped executions. These solutions will be discussed in the following sections.

- **Independent Progress**

The problem of independent progress is that, in an overlapped execution, the computation of each process keeps advancing, since each processor is independently responsible for its own progress. In an interleaved computation, the only requirement is that enabled transitions be continuously chosen and executed. There is nothing to disallow a computation in which only transitions from one process are ever chosen. Such a computation ensures progress of the preferred process, but keeps all other processes stagnant.

The solution of this problem is to impose more restrictions on the basic model, guaranteeing progress for all processes in the interleaved computations. Such restrictions will be presented in the sections on fairness.

### 2.1 Interleaving and Concurrency

In this section, we will compare the behavior of programs under overlapped execution with their interleaved computations.

**Overlapped Execution**

Consider the simple program $A_1$ in Fig. 2.1. The overlapped execution of this program on a system consisting of two independent processors sharing a common memory yields $\{0, 1, 2\}$ as the set of possible outcomes for $y$.

```
out y: integer where y = 1

P_1 :: [\ell_0: y := y + 1 : \ell_0] \quad || \quad P_2 :: [m_0: y := y - 1 : m_0]
```

Fig. 2.1. Program $A_1$. 