Preemption in Concurrent Systems

Gérard Berry
Ecole des Mines
Centre de Mathématiques Appliquées
B.P. 207
06904 Sophia-Antipolis CDX, FRANCE
e-mail: berry@cma.cma.fr

Abstract. Process preemption deals with controlling the life and death of concurrent processes. Well-defined preemption mechanisms are essential in control-dominated reactive and real-time programming, and accurate handling of preemption requires a time-dependent model. We first informally discuss what preemption is about and argue for the need of preemption primitives that are fully orthogonal with sequencing and concurrency ones. Then, we formally present the preemption operators of the Esterel zero-delay process calculus, which is a theoretical version of the Esterel synchronous programming language.

1 Introduction

In concurrent systems, one deals with concurrent processes that coordinate with each other. Coordination can result from information exchange, using for example messages circulating on channels with possibly some implied synchronization. It can also result from process preemption, which is a more implicit control mechanism that consists in denying the right to work to a process, either permanently (abortion) or temporarily (suspension). Preemption is particularly important in control-dominated reactive and real-time programming, where most of the work consists in handling interrupts and in controlling computation and input-output using watchdogs.

Process preemption primitives are available in all operating systems, unfortunately often with fairly loose or complex semantics that makes abstract reasoning difficult. As far as concurrent programming languages are concerned, preemption has been far less studied than communication. Most existing languages offer a small number of preemption primitives, often insufficient to adequately program reactive applications, and their semantics is also often kept quite loose.

In this paper, our goal is twofold: first, we shall informally argue that preemption primitives should be provided at first-class level and with full orthogonality with respect to all other primitives, including concurrency and communication; second, we shall formally discuss preemption operators in the Esterel imperative synchronous process calculus. This calculus is not novel: it is simply a

* Work supported by the CONFER Esprit Basic Research Action and by IFCPAR, Indo-French Center for the Promotion of Advanced Research, New Delhi, India.
rewriting of the ESTEREL synchronous programming language [8, 13] in a process calculus style, with the standard behavioral semantics due to Gonthier [8, 18]. Compared with previous papers on ESTEREL, we adopt a classical process calculus presentation style and we put more emphasis on preemption operators design and semantics. We give only a few illustrative example. The reader can refer to [2, 7, 11, 16, 25] for detailed programming examples using ESTEREL.

In Section 2, we present the basic concepts of process preemption. We establish a fundamental difference between two interpretations of preemption, the "may" interpretation and the "must" interpretation. We show that classical time-independent languages can only handle the weaker notion of "may" preemption, instead of the "must" interpretation that is really needed for reactive systems. Must preemption require reasoning about relative timing of events. In Section 3, we present the perfectly synchronous or zero-delay model on which ESTEREL and other synchronous languages are based [6, 15, 19, 20, 21]. This model allows us to reason about relative timings in an abstract way. In Section ??, we present the syntax and behavioral semantics of the basic ESTEREL operators: sequencing, communication, conditional, concurrency. In Section 4, we present the ESTEREL preemption operators and discuss their axiomatization. We distinguishing between weak and strong operators and between immediate and delayed ones. In Section 5, we show that weak preemption can be obtained as an outcome of a general-purpose escape mechanism. We conclude in Section 6, and we summarize the final ESTEREL calculus in appendix.

2 What Preemption is about

One of the first things a UNIX beginner learns is how to abort its current command by typing the "C control character. Without this command, his only way to stop a looping program would be to unplug the computer. Process abortion is the simplest form of preemption. When he gets more experienced, the UNIX user learns a more elaborate control scheme where preemption is only temporary: by typing "Z, he suspends the activity of his process, and he restarts it from the suspended state by typing "fg". In this way, he gains fine control over the activity of his process. Abortion and suspension are the two fundamental kinds of preemption we shall deal with.

We gave the UNIX example only to make the reader familiar with the concept of preemption. Much more serious examples appear in real-time or reactive process control, where a number of concurrent processes are used to control a physical device such as an airplane or a telephone switch. In such system, one makes a major use of watchdogs that cancel or suspend an ongoing activity if some condition is not fulfilled within a given amount of time; one also often handles anomalies by aborting the normal functioning mode and entering a safety mode. Controlling the life and death of computation and input-output tasks appears to be the most difficult problem in control-dominated reactive systems.