A Formal Framework for the Analysis of Recursive-Parallel Programs

O. Kushnarenko and Ph. Schnoebelen

1 IRISA, Univ. Rennes I, Campus de Beaulieu, 35042 Rennes Cedex France
e-mail: O1ga.Kouchnarenko@irisa.fr

2 Lab. Spéciﬁcation & Vérifcation, ENS de Cachan & CNRS URA 2236,
61, av. Pdt. Wilson, 94235 Cachan Cedex France
email: phs@lsv.ens-cachan.fr

Abstract. RP programs are imperative programs with parallelism and recursion and only a limited way of synchronizing parallel processes. The formal framework we propose here combines (1) a formal operational model of abstract programs (or RP schemes), (2) a set of decision methods for the analysis of RP schemes, (3) a formal operational model for the interpreted programs, and (4) translation results stating how some behavioural properties of the concrete programs can be correctly checked on the corresponding scheme.

Introduction

Automated (and computer-assisted) veriﬁcation and analysis of parallel systems are a fast-growing application ﬁeld for formal methods and techniques in computer science. This is because parallel systems are notably more diﬃcult to understand for human designers and users while formal analysis is not necessarily more diﬃcult from a structural complexity viewpoint.

Today there exists a very successful approach [BCM+92] to some of these questions: systems are commonly modeled by various types of transition systems. In this framework, most problems of system analysis reduce to various kinds of reachability problems on these models. Unfortunately, this classical approach still has many (widely acknowledged) limitations, making it only well-suited to speciﬁc kinds of systems. In particular, its main characteristic is the use of ﬁnite-state transition systems as a foundation.

Recently, many research groups are working to remove, possibly partially, this limitation. They investigate speciﬁc fragments (BPP, PA, ...) of general process algebra, speciﬁc subclasses of communicating automata, pushdown automata, special classes of Petri nets, etc. See [Mol96, Esp96] for a survey.

Keywords: semantics of concurrency, automated veriﬁcation of programs, inﬁnite state systems.

Thanks: This research was mainly done while the 2 authors were at the Leibniz-IMAG Lab. in Grenoble.
In [KS96a, Kou97], we proposed and studied a new infinite-state model of concurrent systems with a good balance between expressive power and decidability of behavioral questions. This formal model, called RP schemes, was developed as an abstract semantic foundation for the RP programming language [VEKM94]. The analysis of RP programs was in fact the real motivation behind the development of a theory of RP schemes. Now a bridge must be built, linking practical questions about real RP programs and theoretical problems about abstract RP schemes.

In this paper, we present a formal framework linking the two viewpoints. This framework provides for a systematic treatment of how results can be transferred from one viewpoint to the other. The formal framework relies on a general Preservation Theorem between two semantics. Today, it is used as a formal semantical foundation in the development of software tools for the analysis of RP programs. These tools are connected to the RP compiler currently developed by a research team of the Theor. Comp. Sci. Dept. at Yaroslavl State University.

This paper is organized as follows: Sections 1 and 2 succinctly recall the basic concepts of RP programs, RP schemes, and their abstract behavioral semantics $\mathcal{M}$. Section 3 summarizes our main decidability results. In Section 4 we define $\mathcal{M}^I$, the formal model we use for the full interpreted language and give our Preservation Theorem linking $\mathcal{M}$ and $\mathcal{M}^I$. We conclude with several examples, illustrating how specific questions can be transferred from $\mathcal{M}^I$ to the abstract $\mathcal{M}$.

1 Recursive-parallel programs and their schemes

The recursive-parallel (RP) style of programming is a specific approach to the organization of parallel computations. This section gives a short introduction to the basic concepts and notions which will be needed throughout. For a more detailed treatment we refer to [KS96a].

1.1 RP programs

RP programs are written in an imperative programming language supporting parallel coroutines (with recursion) and following a precise discipline for handling parallelism. The language has been developed around the parallel machine of the IPTC Institute in Yaroslavl [MVVK88] and it assumes a shared global memory.

An RP program is essentially a set of nested procedures with the possibility of recursive calls. Fig. 1 contains an example of an abstract RP program, "abstract" because we used abstract action names $a, b, c, \ldots$ from some uninterpreted countable alphabet $A$ instead of the usual basic actions from imperative languages: "$x:=y+2$", \ldots