1

Introduction

State space methods are the most popular approach to the automatic verification of concurrent systems. In their basic form, these methods explore the *transition system* associated with the concurrent system. Loosely speaking, the transition system is a graph having the reachable states of the system as nodes, and an edge from a state s to another state s' whenever the system can make a move from s to s'. In the worst case, state space methods need to explore all nodes and transitions of the transition system.

The main problem of transition systems as a basis for state space methods is the well-known *state explosion* problem. Imagine a concurrent system consisting of n sequential subsystems, communicating in some way, and assume further that each of these subsystems can be in one out of m possible states. The global state of the concurrent system is given by the local states of its components, and so the system may have up to m^n reachable states; in fact, this bound is already reached by the rather uninteresting system whose components run independently of each other, without communicating at all. So very small concurrent systems may generate very large transition systems. As a consequence, naive state space methods may have huge time and space requirements even for very small and simple systems.

The unfolding method is a technique for alleviating the state explosion problem. It uses results of the theory of *true concurrency* to replace transition systems by special partially ordered graphs. While these graphs contain full information about the reachable states of the system, their nodes are not reachable states themselves. In particular, the number of nodes of the graph does not grow linearly in the number of reachable states. Since its introduction by McMillan in [84, 85, 86], the unfolding technique has attracted considerable attention. It has been further analyzed and improved [88, 39, 71, 41, 73], parallelized [61, 110], distributed [8], and extended from the initial algorithms, which only allowed us to check the reachability of a state or the existence of a deadlock, to algorithms for (almost) arbitrary properties expressible in Linear Temporal Logic (LTL) [28, 35, 37]. Initially developed, as we shall see below, for systems modeled as "plain" Petri nets, it has been extended to

high-level Petri nets [72, 110], symmetrical Petri nets [29], unbounded Petri nets [2], nets with read arcs [121], time Petri nets [43, 21, 22], products of transition systems [39], automata communicating through queues [83], networks of timed automata [16, 19], process algebras [80], and graph grammars [7]. It has been implemented in several tools [110, 111, 61, 78, 89, 51, 58, 37] and applied, among other problems, to conformance checking [87], analysis and synthesis of asynchronous circuits [74, 76, 75], monitoring and diagnosis of discrete event systems [10, 9, 20], and analysis of asynchronous communication protocols [83].

The goal of this book is to provide a gentle introduction to the basics of the unfolding method, and in particular to give a detailed account of an unfolding-based algorithm for model checking concurrent systems against properties specified as formulas of Linear Temporal Logic (LTL)[1], one of the most popular specification formalisms in the area of automatic verification. Our intended audience is researchers working on automatic verification, and in particular those interested in grasping the algorithmic ideas behind the method, more than the details of true concurrency semantics.

An important question when planning the book was which formalism to choose as system model. The unfolding method requires a formalism having a notion of concurrent components; in particular, the formalism should allow us to determine for each action of the system which components participate in the action and which ones remain idle. For historical reasons, most papers on the unfolding method use Petri nets. We decided to deviate from this tradition and use *synchronous products of labeled transition systems* (*products* for short), introduced by Arnold in [4]. Loosely speaking, in this formalism sequential components are modeled as transition systems (one could also say as finite automata). Components may execute joint actions by means of a very general synchronization mechanism, containing as special cases the mechanisms of process algebras like Milner's Calculus of Communicating Systems (CCS) [90] and Hoare's Communicating Sequential Processes (CSP) [64]. There were three main reasons for choosing products. First, an automata-based model makes clear that the unfolding method is applicable not only to Petri nets. The unfolding method is not tied to a particular formalism, although its details may depend on the formalism to which it is applied. Second, products provide some more information than Petri nets about the structure of the system, and at a certain point in the book (Chap. 4) we exploit this information to obtain some interesting results. Finally (and this is our main reason), products of transition systems contain transition systems as a particular case. Since a transition system is a product of n transition systems for $n = 1$, we can present verification procedures for products by first exhibiting a procedure for the case $n = 1$, and then generalizing it to arbitrary n. This approach is very suitable for describing and discussing the problems raised by

[1] For the so-called stuttering-invariant fragment of LTL, see Chap. 8 for details.

distributed systems, and their solutions. Moreover, the case $n = 1$ is usually simple, and provides a gentle first approximation to the general case.

The reader may now wonder whether the book covers the unfolding method for Petri nets. The answer is yes and no. It covers unfolding methods for so-called *1-bounded* Petri nets (for definition of 1-bounded nets, see, e.g., [30]). Readers interested in unfolding techniques for more general net classes will find numerous pointers to the literature.

Structure of the Book

Chapter 2 introduces transition systems and their products as formal models of sequential and concurrent systems, respectively. As mentioned above, this makes sequential systems a special case of concurrent systems: they correspond to the tuples of transition systems of dimension 1, i.e., having only one component.

Chapter 3 presents the unfolding of a product as a generalization of the well-known unfolding of a transition system (or just a graph) into a tree. In particular, it explains why unfolding a product can be faster than constructing and representing its state space as a transition system. The chapter also introduces the notion of *search procedure*, and lists the three basic verification problems that must be solved in order to provide a model checking algorithm for LTL properties: the *executability*, *repeated executability*, and *livelock* problems.

These three problems are studied in Chaps. 4, 6, and 7, respectively. All these chapters have the same structure: First, a search procedure is presented that solves the problem for transition systems, i.e., for products of dimension 1; the correctness of the procedure is proved, and its complexity is determined. Then, this procedure is generalized to a search procedure for the general case.

The executability problem is the most important of the three. In particular, it is the only problem that needs to be solved in order to answer reachability questions and safety properties. Chapter 5 studies it in more detail, and presents a number of important results which are not directly relevant for the model checking procedure.

Chapter 8 introduces the model checking problem and presents a solution based on the procedures obtained in the previous chapters.

Chapter 9 summarizes the results of the book and provides references to papers studying experimental questions, extensions of the unfolding method, and implementations.

http://www.springer.com/978-3-540-77425-9

Unfoldings
A Partial-Order Approach to Model Checking
Esparza, J.; Heljanko, K.
2008, XII, 172 p. 51 illus., Hardcover
ISBN: 978-3-540-77425-9