Statistical Analysis of Non-Deterministic Fork-Join Processes

Analyzing the state-space of concurrent programs is a notoriously difficult task, if only because of the infamous *state explosion* problem. Several techniques have been developed to “fight” this explosion: symbolic encoding of the state-space, partial order reductions, exploiting symmetries, etc. An alternative approach is to adopt a probabilistic point of view, for example by developing statistical analysis techniques such as [1]. The basic idea is to generate random executions from program descriptions, sacrificing exhaustiveness for the sake of tractability. However, there is an important difference between generating an *arbitrary* execution and generating a random execution according to a known (typically the uniform) distribution. Only the latter allows to estimate the coverage of the state-space of a given analysis.

As a preliminary, I will present a combinatorial interpretations for the fundamental constructions of concurrent programs. To this end, I will introduce a class of programs that uses a *fork-join* model of synchronization, together with loops and a choice construct for *non-determinism.* This is a simple formalism but it is non-trivial in terms of the concurrency features it provides. Based on this combinatorial interpretation, I will derive in a systematic way a combinatorial specification of the possible executions of a program. This specification is a good source of algorithmic investigations and the starting point for the rest of this work.

The first problem I will cover is that of *counting* the number of executions (of bounded length) of the programs. This is the problem one has to tackle to precisely quantify the so-called state “explosion”, and it is also an important building block for the algorithms I will described later. Unfortunately, counting executions of concurrent programs is in fact hard in the general case. It is shown in [2] that even for simple programs allowing *barrier synchronization* (no loops, no choices), counting is a $\sharp P$-complete problem. Fork-join parallelism enables a good balance between tractability and expressivity by enforcing some structure in the state-space.

Of course counting executions has no direct practical application, but it is an essential requirement for two complementary and more interesting analysis techniques I will discuss. First, I will describe an efficient uniform random generator of executions of given bounded length for a given process. Without prior knowledge of the state-space, the uniform distribution yields the best coverage, which makes this random generator a good default candidate for random testing for instance. The second algorithm generates uniform random *prefixes*, which allows the user to introduce some *bias* in the statistical exploration strategy, e.g. towards regions of interest of the state-space, while still giving a good coverage and most importantly still giving control over the distribution. A fundamental characteristics of these algorithms is that they work on the syntactic representation of the program and do not require the explicit construction of the state-space, hence enabling the analysis of systems of a rather large size.

An implementation of all the algorithms presented here can be found at https://gitlab.com/ParComb/libnfj

---

[1] Gaudel, M., Denise, A., Gouraud, S., Lassaigne, R., Oudinet, J., Peyronnet, S.:Coverage-biased random exploration of models. Electr. Notes Theor. Comput. Sci.220(1), 3–14 (2008)

[2] Bodini, O., Dien, M., Genitrini, A., Peschanski, F.: The Combinatorics of BarrierSynchronization. In: International Conference Petri Nets. pp. 386–405. Springer(2019)