IGR Report
A System for Parallel Model Checking

1 Background/Context

A distinctive characteristic of reactive concurrent systems is that their sets of local states have descriptions which are both short and manageable, and the complexity of their behaviour comes from highly complicated interactions with the external environment rather than from complicated data structures and manipulations thereon [13]. One way of coping with this complexity problem is to use formal methods and, especially, computer aided verification tools implementing model checking (see, e.g., [4]) — a technique in which the verification of a system is carried out using a finite representation of its state space.

Model checking and net unfoldings

The main drawback of model checking is that it suffers from the *state space explosion* problem. That is, even a relatively small system specification can (and often does) yield a very large state space. To alleviate this problem, a number of techniques have been proposed. They can roughly be classified as aiming at an implicit compact representation (e.g., in the form of a *binary decision diagram*, or *BDD*, see [2]) of the full state space of a reactive concurrent system, or at an explicit generation of its reduced (though sufficient for a given verification task) representation (e.g., *abstraction* [3] and *partial order reduction* [9] techniques). Among them, a prominent technique is McMillan's (finite prefixes of) Petri net unfoldings (see, e.g., [8,11]). It relies on the partial order view of concurrent computation, and represents system states implicitly, using an acyclic net. More precisely, given a Petri net Σ, the unfolding technique aims at building a labelled acyclic net *Pref* (a *prefix*) satisfying two key properties:

- *Completeness.* Each reachable marking (global state) of Σ is represented by at least one 'witness', i.e., a marking of *Pref* reachable from its initial marking.
- *Finiteness.* The prefix is finite and thus can be used as an input to model checking algorithms, e.g., those searching for deadlocks.

A prefix satisfying these two properties can be used for model checking as a condensed representation of the state space of a system (prefixes are often exponentially smaller than the corresponding reachability graphs, especially if the system at hand exhibits a lot of concurrency). In particular, it can be used to verify many relevant behavioural properties, such as deadlock-freeness [12], or progress properties expressed in a variant of linear time temporal logic [6].

Parallelization of model checking algorithms

Typical state-of-the-art verification packages based on model checking result in sequential implementations with centralized representation of the state space. Therefore, the size of specification which can be verified is limited by the amount of main memory available to hold the state space as the cost of swapping pages to the hard disk leads to a rapid degradation in performance. Also, the time taken to perform the verification is determined by the computational power of the uniprocessor on which the model checking algorithm runs. This can make it impractical, or impossible, to verify a specification.

Therefore, the completed project set out to investigate an alternative: the parallelization of model checking algorithms so that they can run efficiently on parallel computer systems. This would take advantage of the potential for parallelization of algorithms for state space manipulation that could lead to a significant performance speed-up and thus extend the applicability of model checking.

2 Key Advances and Supporting Methodology

The original aims of the proposed research were as follows:

1. To investigate the theoretical principles of the design of parallel algorithms for model checking.
2. To develop efficient parallel algorithms.
3. To implement (in a portable manner) and evaluate these algorithms

The work on the project started from a wide ranging investigation of the existing approaches to model checking from the point of view of their parallelization. The results of this initial investigation were highly encouraging and it was concluded that there were two most promising research directions: (i) parallel exploration of the state space on a network of workstations similar to the APP [14]; and (ii) parallelization of the model checking based on McMillan's unfoldings — a partial order technique. Given the resources available within the project, a decision was made to concentrate on the latter approach as there has already been a significant expertise available on the partial order semantics of concurrency. There are two basic advantages of the unfolding approach when compared with other model checking techniques. First, many instances of practically relevant problems can be solved much more efficiently than by using, for example, exhaustive exploration of the state space or BDDs (see, e.g., [15]). Second, by explicitly representing causality, concurrency and choice, unfoldings support verification based on local states, and yield a direct visual representation of complex concurrent behaviours. This is particularly useful for the debugging and development process, especially at its early stages.

The following advancements have been made:

- integer programming based model checking algorithm employing complete prefixes of Petri net unfoldings
- integer programming based algorithms for detection and resolution of encoding conflicts in specifications of asynchronous circuits
- SAT based algorithms for detection and resolution of encoding conflicts in specifications of asynchronous circuits
- algorithm for efficient generation of possible extensions of a prefix
- parallel unfolding algorithm
- theory of canonical prefixes
- unfolding algorithm for high-level Petri nets

It is also important to stress that all the theoretical developments made within the completed project were backed up by prototype tool implementations and extensive experiments carried out on the standard benchmarks. Finally, the project has ended with the submission and successful defence of a PhD Thesis completed by the student supported by the grant [16]. The main advances made within the project are described in more detail below.

Checking behavioural properties of prefixes

In the work carried out within the project, we first addressed the problem of an efficient model checking of a given complete prefix, as at that time property verification was the bottleneck of the entire method.

In [12], the problem of deadlock checking of a Petri net was reduced to a mixed integer linear programming problem (*MIP*). In the papers [17, 25, 26, 33], we presented a further development of this approach. The essence of the proposed modifications is to transfer the information about causality and conflicts between events involved in an unfolding into a relationship between the corresponding integer variables in the system of linear constraints. We adopt the Contejean and Devie's algorithm (*CDA*), developed in [5], for efficiently solving systems of linear constraints over the domain of natural numbers, and refine it by employing specific properties of the systems of linear constraints to be solved and exploiting partial order dependencies between events in the unfolding of a Petri net. The conducted experiments demonstrated that the resulting algorithm achieves significant speedups. We provided theoretical background and implementation details, as well as outlining ways of reducing the number of variables and constraints in the original system

presented in [12]. This integer programming approach is applicable to deadlock detection, checking mutual exclusion, and various kinds of coverability and reachability analysis. We presented some additional heuristics, which can be incorporated into our algorithm. We also consider an on-the-fly version of our algorithm, which allows one to verify deadlock-freeness in a very memory efficient way, without explicitly generating the system of constraints.

The achieved speedups were so significant that model checking of a given complete prefix ceased to be the bottleneck of this verification method, so there was no point in parallelizing the algorithm. As a result, the main effort was then concentrated on the problem of efficient generation of complete prefixes.

Generation of possible extensions of a prefix

Generation of possible extensions of a prefix is the most time consuming part of the unfolding algorithm. Prior to our efforts, this problem was addressed in [7], where the original McMillan's technique [11] was considerably improved. In particular, to build possible extensions of the branching process being constructed, [7] suggested to keep the concurrency relation and provides a method of maintaining it. This approach is fast for simple systems, but soon deteriorates as the amount of memory needed to store the concurrency relation is quadratic in the size of the already built part of the prefix. In [18,27], we proposed another method of computing possible extensions. It is fully compatible with the concurrency relation approach, but can also be used independently. In contrast to the concurrency relation approach, it does not require much additional memory. The essence of the method is to add new transition instances to the prefix being built not by trying all the transitions one-by-one, but several at once, merging the common parts of the work. Moreover, we provided some additional heuristics. Experimental results demonstrated that one can achieve significant speedups if the transitions of the Petri net being unfolded have overlapping parts.

Parallelization of prefix generation

In [20, 28], we proposed a parallel unfolding algorithm. It should be noted that the traditional sequential unfolding algorithms [8,11] are not well suited for parallelization, and we had to develop a new (significantly different) one. The experiments demonstrated that the degree of parallelism is usually quite high and the resulting algorithms achieves significant speedups comparing with the sequential case. It turns out that the new algorithm not only admits efficient parallelization, but also is faster than the traditional ones even in a sequential implementation. In [20, 28], we also propose some additional optimizations for generating possible extensions of a prefix.

Canonical prefixes

While proving the correctness of the parallel unfolding algorithm, it has been discovered that generated prefixes enjoy a property akin to canonicity. We expounded this in [21,30,34], where a general framework for truncating Petri net unfoldings has been developed. It provides a powerful tool for dealing with different variants of the unfolding technique, in a flexible and uniform way. In particular, by finely tuning the so-called *cutting contexts,* one can build prefixes which better suit a particular model checking problem. A fundamental result is that, for an arbitrary Petri net and a cutting context, there exists a 'special' *canonical* prefix of its unfolding, which can be *defined without resorting to any algorithmic argument.* This should be contrasted with former approaches, where a prefix was 'defined' as any of the results produced by a non-deterministic unfolding algorithm, which resulted in a very 'inconvenient' theory.

We introduced a new, stronger notion of completeness of prefixes, which was implicitly assumed by many existing model checking algorithms employing unfoldings. We have shown that the canonical prefix is complete w.r.t. this notion.

It turns out that the canonical prefix is exactly the prefix generated by arbitrary runs of the standard unfolding algorithm, as well as our parallel unfolding algorithm (note that both algorithms are non-deterministic). This gives a new correctness proof for them, which is much simpler in the case of the parallel algorithm. As a result, relevant model checking tools can now make stronger assumptions about the properties of the prefixes they use. In particular, they can safely assume that for each configuration containing no cut-off events, all firings are preserved.

Finally, we proposed conditions for the finiteness of the canonical prefix, and presented criteria allowing placing bounds on its size, which should help in choosing problem-specific cutting contexts. It is worth noting that in order to deal with the finiteness problem we proved a version of König's Lemma [10] for branching processes of (possibly unbounded) Petri nets.

Prefixes of high level Petri nets

In [23, 31], we defined branching processes and unfoldings of high-level Petri nets and proposed an algorithm which builds finite and complete prefixes of such nets. We established an important relation between the branching processes of a high-level net and those of its low-level *expansion*, viz. that the sets of their branching processes are the same, allowing us to import results proven for low-level nets. Among such results are the canonicity of the prefix for different cutting contexts, and the usability of the parallel unfolding algorithm proposed in [**?**, **?**]. Our approach is conservative in the sense that all the verification tools employing the traditional unfoldings can be reused with such prefixes. The conducted experiments demonstrated that it is, on one hand, superior to the traditional approach on data-intensive application, and, on the other hand, has the same performance on control-intensive ones.

Applications to asynchronous circuits design

Signal Transition Graphs (STGs) are a Petri nets based formalism widely used for describing the behaviour of asynchronous control circuits. STGs are a form of interpreted Petri nets, in which transitions are labelled with the names of rising and falling edges of circuit signals. One of the stages of circuit synthesis based on STGs involves the detection of coding conflicts, and the elimination of them. In [19, 29], we proposed a solution for the CSC analysis problem, consisting in identifying conflicts which occur when semantically different reachable states have the same binary encoding. We showed that the notion of a coding conflict can be characterized in terms of a system of integer constraints, and developed an efficient technique for solving such a system. It is also worth pointing out that the method allows one not only to find states which are in coding conflict, but also to derive execution paths leading to them without performing a reachability analysis. This algorithm provided a foundation for the unfolding-based framework for resolution of coding conflicts described in [22], which used the set of pairs of configurations representing coding conflicts produced by the algorithm as an input.

The integer programming based algorithm in many cases achieved significant speedups, but on some of the benchmarks its performance was still not entirely satisfactory: on several large instances the test did not terminate within the time limit. In [24, 32] we characterized the CSC problem in terms of boolean satisfiability (SAT). Though being more complicated than the integer programming translation, the SAT one allows more dependencies between the variables to be exploited. State-of-the-art SAT solvers are quite efficient, and have been for long employed in the model checking community. In our experiments, we achieved significant speedups using this method. We show also how the proposed translation can be extended to the USC and normalcy problems. Moreover, by explicitly representing causality, concurrency and conflicts, unfoldings support verification based on local states, and yield a direct visual representation of complex concurrent behaviours [22].

Prototype tools

The following two tools were developed within the project, and were extensively used to obtain experimental result:

- CLP - Linear programming model checker. It uses a finite complete prefix of a Petri net and can check deadlock freeness, reachability or coverability of a marking, or perform an extended reachability analysis, i.e. check if there exists a reachable marking satisfying the given predicate. CLP can be used both as a separate utility or as a part of the PEP tool [1].
- PUNF - Petri net unfolder. PUNF builds a finite and complete prefix of a safe Petri net. The prefixes generated by PUNF can be passed as an input to the CLP model checker.

Both tools can be downloaded from

`http://www.cs.ncl.ac.uk/people/victor.khomenko/home.formal/tools/tools.html`

Research output

Overall, the project has resulted in one PhD Thesis [16] (which comprises and systematizes our work described above), eight peer reviewed conference papers [17–24], eight technical reports [25–32], two submitted journal articles [33, 34], and two tools.

3 Research Impact

The results of the project are relevant to potential users at three different levels:

- The use of unfolding-based model checking in designing new systems, where our ultimate beneficiaries are the designers of reactive concurrent systems in a wide range of industrial applications.
- The use of unfoldings in designing general purpose or application specific computer-aided design, verification and model-checking tools, where our beneficiaries are designers of CAD systems and research demonstrators, developers of industrial model-checkers and verifiers, who can incorporate our unfolding engine or integer programming based parallelized model checker as part of their systems.
- The use of the net unfolding approach as a way to represent concurrent behaviours, where it can benefit both industry and academia in offering courses on concurrency and asynchronous systems and providing efficient ways of semantic model structuring and visualization.

4 Further Research or Dissemination Activities

The dissemination of the results of the completed work has been done via publications in leading international conferences, and through technical reports and the Internet.

References

[1] E.Best and B.Grahlmann: PEP — more than a Petri Net Tool. Proc. of TACAS'96, Margaria T., Steffen B. (Eds.). Lecture Notes in Computer Science 1055 (1996)

[2] R.E.Bryant: Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers 35 (1986)

[3] E.M.Clarke, O.Grumberg, and D.E.Long: Model Checking and Abstraction. ACM Transactions on Programming Languages and Systems 16 (1994)

[4] E.M.Clarke, O.Grumberg and D.Peled: Model Checking. MIT Press (1999)

[5] E.Contejean and H.Devie: An Efficient Incremental Algorithm for Solving Systems of Linear Diophantine Equations. Information and Computation 113 (1994)

[6] J.Esparza and K.Heljanko: Implementing LTL Model Checking with Net Unfoldings. SPIN workshop (2001)

[7] J.Esparza and S.Römer: An Unfolding Algorithm for Synchronous Products of Transition Systems. Proc. of CONCUR'99, Baeten, J.C.M., Mauw, S. (Eds.). Lecture Notes in Computer Science 1664 (1999)

[8] J.Esparza, S.Römer and W.Vogler: An Improvement of McMillan's Unfolding Algorithm. Formal Methods in System Design 20 (2002)

[9] P.Godefroid: Partial-Order Methods for the Verification of Concurrent Systems: an Approach to the State-Explosion Problem. Lecture Notes in Computer Science 1032 (1996)

[10] D.König: Über eine Schlußweise aus dem Endlichen ins Unendliche. Acta Litt. ac. sci. Szeged 3 (1927)

[11] K.L.McMillan: Symbolic Model Checking: an Approach to the State Explosion Problem. PhD thesis, CMU-CS-92-131 (1992)

[12] S.Melzer and S.Römer: Deadlock Checking Using Net Unfoldings. Proc. of CAV'1997, Grumberg O. (Ed.). Springer-Verlag, Lecture Notes in Computer Science 1254 (1997)

[13] A.Pnueli: Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends. Advanced School on Current Trends in Concurrency (1994)

[14] R.K.Ranjan, J.V.Sanghavi, R.K.Brayton and A.Sangiovanni-Vincentelli: Binary Decision Diagrams on Network of Workstation. ICCD (1996)

[15] `http://wwwbrauer.informatik.tu-muenchen.de/gruppen/theorie/pom/`

Project publications

PhD Thesis

[16] V.Khomenko: Model Checking Based on Prefixes of Petri Net Unfoldings. PhD Thesis. School of Computing Science, University of Newcastle upon Tyne (2003)

Conference papers

[17] V.Khomenko and M.Koutny: LP Deadlock Checking Using Partial Order Dependencies. Proc. of CONCUR'2000, Palamidessi C. (Ed.). Springer-Verlag, Lecture Notes in Computer Science 1877 (2000)

[18] V.Khomenko and M.Koutny: Towards An Efficient Algorithm for Unfolding Petri Nets. Proc. of CONCUR'2001, Larsen K.G., Nielsen M. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2154 (2001)

[19] V.Khomenko, M.Koutny, and A.Yakovlev: Detecting State Coding Conflicts in STGs Using Integer Programming. Proc. of DATE'2002, Kloos C.D., Franca J. (Eds.). IEEE Computing Society (2002)

[20] K.Heljanko, V.Khomenko, and M.Koutny: Parallelisation of the Net Unfolding Algorithm. Proc. of TACAS'2002, Katoen, J.-P., Stevens, P. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2280 (2002)

[21] V. Khomenko, M. Koutny, and W. Vogler: Canonical Prefixes of Petri Net Unfoldings. Proc. of CAV'2002, Brinksma E., Larsen K.G. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2404 (2002)

[22] A. Madalinski, A. Bystrov, V. Khomenko, and A. Yakovlev: Visualization and Resolution of Coding Conflicts in Asynchronous Circuit Design. Proc. of DATE'2003, Verkest, D., Wehn, N. (Eds.). IEEE Computing Society (2003)

[23] V. Khomenko and M. Koutny: Branching Processes of High-Level Petri Nets. Proc. of TACAS'2003, Garavel, H., Hatcliff, J. (Eds.). Springer-Verlag, Lecture Notes in Computer Science 2619 (2003)

[24] V. Khomenko, M. Koutny, and A. Yakovlev: Detecting State Coding Conflicts in STG Unfoldings SAT. Proc. of ICACSD'2003, Balarin, F., Lilius, J. (Eds.). IEEE Computing Society (2003)

Technical reports

[25] V.Khomenko and M.Koutny: Deadlock Checking Using Linear Programming and Partial Order Dependencies. Technical Report CS-TR-695, Department of Computing Science, University of Newcastle upon Tyne (2000)

[26] V.Khomenko and M.Koutny: Verification of Bounded Petri Nets Using Integer Programming. Technical Report CS-TR-711, Department of Computing Science, University of Newcastle upon Tyne (2000)

[27] V.Khomenko and M.Koutny: An Efficient Algorithm for Unfolding Petri Nets. Technical Report CS-TR-726, Department of Computing Science, University of Newcastle upon Tyne (2001)

[28] K.Heljanko, V.Khomenko, and M.Koutny: Parallelisation of the Petri Net Unfolding Algorithm. Technical Report CS-TR-733, Department of Computing Science, University of Newcastle upon Tyne (2001)

[29] V.Khomenko, M.Koutny, and A.Yakovlev: Detecting State Coding Conflicts in STGs Using Integer Programming. Technical Report CS-TR-736, Department of Computing Science, University of Newcastle upon Tyne (2001

[30] V.Khomenko, M.Koutny, and W.Vogler: Canonical Prefixes of Petri Net Unfoldings. Technical Report CS-TR-741, Department of Computing Science, University of Newcastle upon Tyne (2001)

[31] V.Khomenko and M.Koutny: Branching Processes of High-Level Petri Nets. Technical Report CS-TR-763, Department of Computing Science, University of Newcastle upon Tyne (2002)

[32] V.Khomenko, M.Koutny, and A.Yakovlev: Detecting State Coding Conflicts in STG Unfoldings Using SAT. Technical Report CS-TR-778, Department of Computing Science, University of Newcastle upon Tyne (2002)

Submitted journal articles

[33] V.Khomenko and M.Koutny: Verification of Bounded Petri Nets Using Integer Programming. Formal Methods in System Design (submitted paper)

[34] V.Khomenko, M.Koutny and W.Vogler: Canonical Prefixes of Petri Net Unfoldings. Acta Informatica (submitted paper)