Automatic Synthesis of Distributed Systems

Final Report on EPSRC Grant R64322/01

1 Introduction

In this project we studied the problem of automatically synthesising a distributed system starting from a formal specification.

1.1 Motivation

Automatic synthesis can be seen as a complementary approach to verification. Both approaches were proposed in the early 80s in the seminal works [10, 14]. The verification procedure needs beside a specification also a candidate model. If this candidate fails to fulfill the specification, the designer must iteratively improve the model and check it again. In automatic synthesis, the system is automatically derived from the specification, and, moreover, it is correct by construction. Despite these advantages, and the fact that the two approaches have similar computational complexity for logics like LTL or CTL, automatic synthesis has been far less successful than verification, both from a theoretical and a practical point of view. The project is based on the following hypothesis: Specification languages like LTL and CTL are not adequate for synthesis of distributed systems, because they cannot express properties about distribution of actions among processes, and so the designer has no control about the distribution aspect of the problem. Specification languages that take this into account, and for which the synthesis problem has reasonable complexity, will lead to successful practical applications.

1.2 Original goals and methodology

The original goal of the project was to investigate a synthesis framework with *asynchronous automata* as abstract implementation model. Asynchronous automata were introduced by Zielonka as a natural and very general model for finite-state distributed systems. They can be seen as a tuple of cooperating finite automata. Given an alphabet A of actions, a set P of processes and a distribution D of actions among processes (i.e., a function $D\colon P \to 2^A$) Zielonka's landmark result [19] characterises the regular languages over A that admit a distributed implementation as asynchronous automaton with D as distribution. It also provides a procedure to construct an asynchronous automaton for a given implementable language L. Unfortunately, the construction is complex and the resulting asynchronous automaton can be exponentially larger than the original automaton accepting L. While complexity results show that this exponential blow-up is unavoidable in the worst case, we expected to find simplifications leading to small asynchronous automata for practical cases.

The original objectives of the project were:

1. to develop a specification language based on a distributed version of temporal logic, which is able to express properties about independence of actions,
2. to design a synthesis procedure for the language based on simplifications/improvements of the algorithms for asynchronous automata [1],
3. to implement the new procedure efficiently, and
4. to apply the above procedure to case studies in areas such as small distributed algorithms (e.g. mutual exclusion, communication protocols) and asynchronous circuits.

1.3 Changes in goals and methodology

The objectives had to be changed during the course of the project. The problem of simplifying Zielonka's construction to obtain smaller asynchronous automata (objective 2) turned out to be much harder than expected. After one year of investigating many different roads no substantial progress had been made[2]. Therefore, we concentrated on efficient implementations of the procedure and on its application to case studies (objectives 3. and 4.).

At the same time, we decided to replace objective 2. The project had fixed an implementation model (asynchronous automata) and a particular correctness relation between specification and implementation (language equivalence/language inclusion). Since this particular choice was proving to be very hard, we decided to broaden our scope,

[1] Here we wished to use well-known results linking temporal logic to finite automata.

[2] Actually, we now think that no substantial simplifications are possible.

and investigate the complexity of automatic synthesis problems for a larger spectrum of implementation models, less powerful than asynchronous automata (i.e., fewer specifications can be implemented), and a larger spectrum of correctness relations. We followed two different paths:

(a) We investigated the synchronous products of transition systems of [8] as an alternative to asynchronous automata. This model is very widespread (it can also be described as finite automata with CSP-like communication), and has two advantages:

   – it admits a conceptually simple synthesis procedure [18], and
   – the distributed implementation, if it exists, is guaranteed to be linear in the size of the specification.

(b) We investigated a concrete synthesis problem suggested by our intended beneficiaries in the area of asynchronous circuit design. In cooperation with Alex Yakovlev's group at the University of Newcastle, we investigated the complexity of and efficient procedures for the automatic synthesis of asynchronous circuits from specifications in the form of *Signal Transition Graphs*. The implementability problem can be seen as a very particular case of the implementability problem for asynchronous automata. The asynchronous circuit community had characterised the implementable specifications, but the complexity of checking implementability was not known.

This two paths became our two new objectives 1' and 2'. The price to pay for extending the old objective 2 to the new objectives 1' and 2' was that objective 1 had to be abandoned.

The revised objectives of the project are:

(1') to investigate the computational complexity of the synthesis problem for

   – transition systems with explicit information about the distribution of actions as *specification language*,
   – asynchronous automata or synchronous products of transition systems as *implementation model*, and
   – different *correctness relations* (isomorphism, language equivalence, bisimulation) between specification and implementation.

(2') to investigate the computational complexity of the synthesis problem for specifications in the form of Signal Transition Graphs,

(3,4) As in the original proposal.

2 Results

To describe the results of this project, we consider the four revised objectives separately.

2.1 Computational Complexity of Synthesis of Distributed Transition Systems

The distributed implementability problem is defined as follows: Given a specification in the form of a transition system and a distribution of actions between the processes, does there exist a distributed implementation satisfying the initial specification?

We considered two cases, where 'satisfying' is interpreted as: the global behaviour of the implementation is *equivalent to* (Section 2.1.1), respectively *included in* the specification (Section 2.1.2).

We considered two implementation models: asynchronous automata and synchronous products of transition systems. The important overall result is that the complexity of checking implementability has similar complexity for both. Therefore, we have a clear trade-off between the probability that a given specification can be implemented (higher for asynchronous automata), and the probability that an implementation of reasonable size can be obtained (higher for synchronous products of transition systems).

Since the complexity results are similar, we only present the asynchronous automata case.

2.1.1 The Equivalence Case

The exact formulation of the problem in this case becomes:

> Given a specification as a transition system together with a distribution of its actions over a set of processes, does there exist an asynchronous automaton whose global transition system is *equivalent* to the initial transition system?

The term 'equivalent' is interpreted as: (a) isomorphic, (b) language equivalent, and (c) bisimilar. While solutions to the three problems can be found in [18] (except for the bisimilarity case, where only partial results are presented), no exact complexity bounds for computing these solutions existed. The results of this subsection can be found in [**?**] and [4].

Isomorphism We proved that the problem is NP-hard in the general case. Together with results of [16, 17, 18], this shows that the problem is NP-complete.

Experience showed that in practice the specifications are usually not isomorphic with the global state space of an asynchronous automaton. Still, this case is important to consider because we use the isomorphism test in the heuristic of [7] for finding a smaller synthesised solution.

Language Equivalence We proved that the problem is PSPACE-complete and for the deterministic case we gave a simple polynomial algorithm.

Bisimilarity The problem is still open in the general case. If we are looking for a deterministic implementation (the specification can still be nondeterministic), we prove that the problem is polynomial (part of the proof is already implicit in [18]).

2.1.2 The Inclusion Case

Our experiments show that if we take equivalence as correctness criterion, specifications are often not implementable. So it is sensible to relax the correctness relation to inclusion relations.

Language Inclusion We proved in that the problem is undecidable.

Isomorphic Embedding In this case we require that the transition system of the implementation can be isomorphically embedded in the transition system of the specification. We prove that the synthesis problem is NP-complete. We have an implementation for this case (both as heuristic search and complete search, see Section 2.3.2).

2.2 Checking Implementability of Signal Transition Graphs

Signal Transition Graphs (STGs) [9] are Petri nets whose transitions are labelled with circuit signals. They constitute a specification of the behaviour of the circuit which explicitly declares that some signal changes may occur independently of each other, and therefore must be executed by different circuit elements.

Research on the automatic synthesis of asynchronous circuits has characterised the implementable STGs as those admitting a *consistent and complete state coding* (CCSC). From this coding it is possible to automatically produce an implementation at gate level in a series of steps.

Checking the existence of a CCSC for arbitrary Petri net is a PSPACE-complete problem. Khomenko, Koutny, and Yakovlev have developed a procedure that (while requiring exponential time in the worst case) exploits the concurrency of the STG to speed up the check [13, 12]. The procedure is based on *Petri net unfoldings*, a technique introduced by McMillan [15] and further developed by the principal investigator [11]. In cooperation with Khomenko, we have worked on the efficient implementation of the technique. The result has been an efficient model-checker that uses Petri net unfoldings, and can be applied not only to STG problems, but also to more general ones. The results have been published in [6]. This part of the research was carried out by Claus Schröter (see the section on Explanation of Expenditure).

In a second line of research, we investigated the complexity of the implementability problem for the class of free-choice STGs[3], which plays an important rôle in practice. We found the first polynomial algorithm to decide if the STG has a consistent (not necessarily complete) state coding. The result was published in [1], and has been selected for a forthcoming special issue of a journal [2]. It has been implemented in a Master's thesis at the University of Stuttgart [5] supervised by the investigators.

In still unpublished work we have proved, however, that the existence of a consistent *and complete* state coding is NP-complete [3]. Fortunately, there are automatic techniques that, at the cost of losing concurrency, transform an STG with a consistent state coding into another one with a CCSC.

2.3 Efficient Synthesis of Asynchronous Automata

From a practical point of view, one problem of asynchronous automata is that they have a set of *global* accepting states, which in a concurrent implementation would require the existence of a global controller that can observe the global state of the system. This was done for the sake of similarity with the case of finite automata, and because of its elegant mathematical properties. However, it is against the principle of a distributed system involving only local synchronisation. So, when implementing our procedures we must restrict ourselves to the class of *safe* asynchronous automata, where the controller is not necessary. This required to developed some theory characterizing the languages accepted by safe deterministic asynchronous automata, which was published in [7].

2.3.1 The Synthesis Procedure

Recall that Zielonka's result [19] characterises the regular languages that admit a distributed implementation as an asynchronous automaton with a given distribution of actions. It also provides a procedure to construct an asynchronous automaton for a given implementable language. The procedure starts with a deterministic automaton for the language, and proceeds to *unfold* it (in the same way that a graph is unfolded into a tree) until a certain cut-off criterion is met. At that point the tree is "folded back" into a graph, and the asynchronous automaton is extracted.

The problem of the construction is that the result may be much larger than a minimal asynchronous automaton for the

[3]The STG is also required to satisfy some behavioural properties that can be checked in polynomial time.

language. Our experiments confirmed that, even for small examples in which the original automaton has some dozens of nodes, Zielonka's procedure may yield asynchronous automata being two orders of magnitude larger than a minimal asynchronous automaton. In order to palliate this problem we use a result by Morin [16] which characterises the transition systems that are isomorphic with the global state space of an asynchronous automaton. Starting with the specification, we test at each step using Morin's conditions whether the specification can already be transformed into an asynchronous automaton. If it can, we stop with success. Otherwise, we unfold precisely that part of the current automaton that contribute most to the violation of the conditions in Morin's test and repeat the whole procedure. This heuristic is presented in [7], and leads to significantly smaller automata in our case studies.

2.3.2 The Implementation

We implemented (in C) part of the ideas presented above. The input is a regular specification given by a (minimal) deterministic transition system and a distribution.

The first step consists of a sanity check whether the specification admits indeed a distributed implementation (see Section 2.1). For that, it is enough to check that the input transition system satisfies two syntactic constraints, the independent diamond (ID) and forward diamond (FD) rules.

In case they are not satisfied, according to Section 2.1.2, we look for subautomata of the original transition system satisfying ID and FD. The problem is NP-complete and we implemented it in a logic programming framework called SMODELS (`http://www.tcs.hut.fi/Software/smodels/`). At this point we profited from the expertise of Dr. Keijo Heljanko of Helsinki University of Technology who was visiting us in Stuttgart for one year. The results of the collaboration are included in [4]. We also designed a polynomial-time heuristic for finding an ID-FD subautomaton that tries to locally solve the violations of the rules. This may not always find a solution, but when it does, it does it fast (as the experiments showed).

If eventually the specification is implementable, we apply the unfolding approach mentioned in Section 2.3.1. The results obtained were encouraging, at least for the small examples we considered so far (see [7]).

The implementation was made available online at: `http://www.fmi.uni-stuttgart.de/szs/tools/synasync/`

2.3.3 Case Studies

The project has had an important component of experimentation. Case studies have been conducted in three areas:

- Synthesis of asynchronous automata.
  In this are we experimented with parametrized instances of the classical mutual exclusion and deadlock avoidance algorithms. We were able to obtain solutions for languages whose corresponding minimal DFA has up to some hundreds of states. Beyond this point the complexity of Zielonka's procedure made the approach impractical. The experimental results are described in [7, 4].

- Implementability check of STGs.
  For arbitrary STGs we did not conduct any experiments, since our algorithmic contributions were extensively tested by Khomenko in his PhD Thesis [12]. For free-choice STGs we conducted experiments, contained in [5], based on a set of benchmarks provided by Alex Yakovlev and Jordi Cortadella. The results show that the new polynomial algorithm for consistency is extremely fast, requiring only a few seconds for STGs with hundreds or thousands of nodes.

- Model-checking using unfolding techniques.
  Extensive experiments were conducted on examples modelling distributed systems and due to J.C. Corbett, plus some additional academic examples. The results show that the unfolding technique is very competitive for systems exhibiting a high degree of concurrency. The results are described in [6], and can also be found at: `http://www.fmi.uni-stuttgart.de/szs/people/schroeter/CAV/cav.tgz`

3 Project Plan Review

The project was burdened by two facts:

(a) The failure of our attempts to simplify Zielonka's construction.

(b) The move of the principal investigator from Edinburgh to Stuttgart.

We do not think that better planning could have avoided (a). We strongly believed that the simplifications were possible due to our past experiences with other models and to the development of new techniques since Zielonka's original paper, in particular the development of Petri net unfoldings. As mentioned in the Timeliness part of the proposal "we address a fundamental issue . . . and attack a long standing open problem". The problem just turned to be too hard.

The partial overlapping of (a) and (b) delayed the necessary reorganization of the project. Once the new objectives were established (about one year after the project started), the project ran smoothly.

4 Explanation of Expenditure

On 1.3.2003 the principal investigator moved from the University of Edinburgh to the University of Stuttgart. Thanks to an agreement between EPSRC and the DFG, the project could be transferred to the University of Stuttgart. Alin Ştefănescu, who was supported by the project from 1.10.01 to 31.3.03, moved to Stuttgart on 1.04.03. Due to the different structures of the British and German academic systems, the project's funding was not sufficient to support a PhD student in Germany. Fortunately, Alin Ştefănescu could be employed as Wissenschaftlicher Mitarbeiter directly by the University, and he could continue his research work, although he had to combine this with teaching duties. The project's money was used to partially support Claus Schröter, who joined the project to investigate the use of unfolding techniques and its applications to STGs.

5 Conclusions

The project has failed to meet its initial objective of developing an efficient automatic synthesis procedure, based on asynchronous automata, able to deal with the synthesis of middle-size systems. We do not think that future developments will improve this situation, because the difficulties we have identified are of fundamental nature. However, the project has been successful in:

(a) providing an optimized implementation for small systems,

(b) identifying and analysing viable alternatives to the original objective, and

(c) developing efficient techniques in the special field of automatic synthesis of asynchronous circuits using STGs.

Briefly, the outcomes of the project are:

1. a complexity study of the problem of checking the distributed implementability of a regular specification,
2. an efficient polynomial-time test for the distributed implementability of an STG-specification, and
3. a prototype implementation of heuristics for the synthesis of asynchronous automata that was able to automatically synthesise new solutions for classical problems.

The principal investigator wishes to thank EPSRC for the smooth transfer of the project from Britain to Germany. This very much helped him to continue with his lines of research, which otherwise would have been stopped for about one year. Although the different structures of the academic systems required a substantial reorganization of the project and lead to some delays, the overall experience was very satisfactory.

References Emerging from the Project

[1] J. Esparza. A polynomial-time algorithm for checking consistency of free-choice signal transition graphs. In *Proc. of the 3rd Int. Conf. on Application of Concurrency to System Design (ACSD 2003)*, pages 61–70. IEEE Computer Society, 2003.

[2] J. Esparza. A polynomial-time algorithm for checking consistency of free-choice signal transition graphs. *Fundamenta Informaticae*, To appear.

[3] J. Esparza, P. Jančar, and A. Miller. On the complexity of checking the consistent state coding property. Technical report, Universität Stuttgart. In preparation.

[4] K. Heljanko and A. Ştefănescu. Complexity results for checking distributed implementability. Technical Report 05/2004, Universität Stuttgart, 2004. Online at `http://www.fmi.uni-stuttgart.de/people/stefanescu/tr-05-2004.pdf`.

[5] S. Mahadevan. Implementation of a consistency test for free-choice signal transition graphs. Master's thesis, INFOTECH, University of Stuttgart, 2004.

[6] C. Schröter and V. Khomenko. Parallel LTL-X Model Checking of High-Level Petri Nets Based on Unfoldings. In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04)*, volume 3114 of *Lecture Notes in Computer Science*, pages 109–121. Springer, 2004.

[7] A. Ştefănescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In R. Amadio and D. Lugiez, editors, *CONCUR'03*, volume 2761 of *LNCS*, pages 27–41. Springer, 2003.

Other References

[8] A. Arnold. *Finite transition systems and semantics of communicating systems*. Prentice Hall, 1994.

[9] T.-A. Chu. *Synthesis of self-timed VLSI circuits from graph-theoretic specifications*. PhD thesis, MIT, 1987.

[10] E.M. Clarke and E.A. Emerson. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2:241–266, 1982.

[11] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20:285–310, 2002.

[12] V. Khomenko. *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD thesis, School of Computing Science, University of Newcastle upon Tyne, 2003.

[13] V. Khomenko, M. Koutny, and A. Yakovlev. Detecting state coding conflicts in STG unfoldings using SAT. In *In Proc. of the 3rd International Conference on Application of Concurrency to System Design (ACSD 2003)*, pages 51–60. IEEE Computer Society, 2003.

[14] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 6(1):68–93, 1984.

[15] K. L. McMillan. A technique of state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[16] R. Morin. Decompositions of asynchronous systems. In *CONCUR'98*, volume 1466 of *LNCS*, pages 549–564. Springer, 1998.

[17] R. Morin. Hierarchy of asynchronous automata. In *WDS'99*, volume 28 of *Electronic Notes in Theoretical Computer Science*, pages 59–75, 1999.

[18] M. Mukund. From global specifications to distributed implementations. In B. Caillaud, P. Darondeau, and L. Lavagno, editors, *Synthesis and control of discrete event systems*, pages 19–34. Kluwer, 2002.

[19] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. Inform. Théor. Appl.*, 21:99–135, 1987.