Automatic Synthesis of Distributed Systems

Alin Ştefănescu
LFCS, Division of Informatics, University of Edinburgh,
JCMB, King's Buildings, Edinburgh EH9 3JZ
Alin.Stefanescu@dcs.ed.ac.uk

Abstract

Our objective is to develop an automatic synthesis procedure for distributed systems having a flexible specification language and a reasonable computational complexity such that applications to real case studies to be possible. We intend to use asynchronous automata as theoretical models for the systems and distributed versions of temporal logics based on Mazurkiewicz traces for the specification. Target applications of the synthesis procedure are classes of systems such as small distributed algorithms and asynchronous circuits.

1 Introduction

The design of correct distributed systems is a difficult and error-prone task. This is due to the multitude of possible interactions between the concurrent components of the system. Briefly, the design of a system consists in giving a specification and then finding a model for this.

There are two complementary approaches to this problem: verification and synthesis. The verification procedure checks if a given model satisfies the specification, whereas in the synthesis approach the model is directly synthesized from the specification. Regarding verification, if the given model fails to fulfill the specification, this should be checked and iteratively improved by the user. In the latter case of synthesis, the system is correct by construction so that there is no need for further verification. Despite this advantage and the fact that the approaches have similar computational complexity, automatic synthesis has so far been less successful than verification. A possible explanation is the fact that both approaches were mostly studied using temporal logics like LTL or CTL for the specification. A shortcoming of these logics is the fact that they cannot express properties about casual independences between the different actions of the system, hence these properties cannot play a role in the synthesis procedure. This will be discussed further in section 2.

Asynchronous automata are a well-known formal model in which synthesis takes into account an independence relation on actions. Asynchronous automata were proposed by Zielonka [Zie87] as a natural generalization of finite state automata for concurrent systems and, loosely speaking, they are sequential automata communicating through 'rendez-vous'. These automata have been introduced to provide a model of computation for trace languages, which are languages closed under an explicit independence relation between actions. Zielonka gives a construction that accepts as input a regular trace language and a distribution pattern and outputs an asynchronous automaton accepting the given language. The procedure is complicated and computationally involved despite attempts to simplify it. Furthermore, little has been done to use its power in practice and to turn the theory into a reliable computer science tool (to the best of our knowledge no steps towards implementations have been taken).

Objectives of our research The main lines along which our efforts will concentrate are the following:

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

The extended abstract is organized as follows. In the next section we present the synthesis problem and current solutions. In section 3 we outline preliminary ideas and the proposed approach. We give a toy example in section 4 and we continue with more insights in section 5, where we summarize the work done until now. We conclude with section 6.

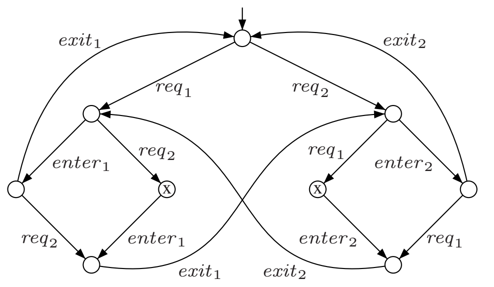
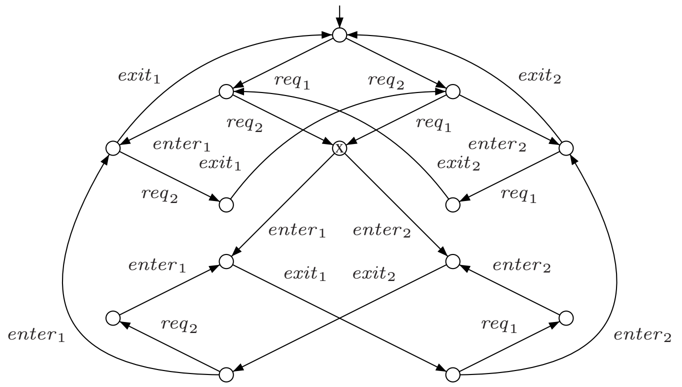
Synthesis from a CTL specification	Synthesis from specification using traces
	

Table 1. Comparing two solutions for mutual exclusion problem

2 The synthesis problem

We study the problem of automatic synthesis of distributed systems. In system synthesis we transform a specification into a system that is guaranteed to satisfy this specification. This is a problem that has been investigated in various frameworks but generally proved to be a rather difficult question. We mention seminal works, which generated the mainstreams in the area, and try to motivate the goals of our research.

Temporal logics Most related work was carried out using temporal logic for the specification. In [EC82], Emerson and Clarke proposed a synthesis procedure using the branching time temporal logic CTL. Of similar nature, but using the linear time temporal logic LTL, is the approach by Manna and Wolper in [MW84]. The main problem with approaches based on (classical) temporal logics is that such logics are not able to express the independences between the involved actions, thus the procedure synthesizes a global transition system and not a distributed concurrent one. Moreover, the constructed transition system might not be distributable (i.e. there is no distributed system exhibiting the same behaviour).

A small example illustrating this claim is the solution for mutual exclusion synthesized in [EC82]. Two processes must share a resource that they can access from a critical section. We consider an alphabet of actions $\Sigma := \{req_1, enter_1, exit_1, req_2, enter_2, exit_2\}$ with the usual intended meanings: request access to a critical section, enter it and exit from it; the indices 1 and 2 specify the process that executes the action. The specification based on CTL consists of three main properties (mutual exclusion, deadlock freedom and absence of starvation) plus a number of properties for local behaviour. The result of synthesis is the transition system shown in the left column of Table 1. The solution is not satisfactory because it is not distributable. The reason is that actions req_1 and req_2 have to be independent (i.e. each process should freely request access to the critical section without having to synchronize in any way), but the given solution fails in this respect. If req_1 and req_2 would be independent, then the sequences $req_1 req_2$ and $req_2 req_1$ should lead to the same global state, but this is not the case (see the states marked with an 'x').

Further research on synthesis using temporal logics was generated by Pnueli and Rosner [PR89], who studied the synthesis of reactive modules (or synthesis of open systems interacting with their environment). In this problem, there is a module and its environment that alternate their moves as in a game and the goal is to find a winning strategy for the module (in other words, the module should satisfy the specification irrespective of how the environment behaves). The problem received further attention (see for example [KV01] for recent acquisitions in the area), but the specification does not yet incorporate the notion of independence of actions and the result of synthesis is a module and not a distributed system.

Petri nets Petri nets are well established models for concurrent systems. In [ER90], Ehrenfeucht and Rozenberg introduced the theory of regions in order to synthesize a Petri net from a given transition system (see [BD98] for a survey on further development of theory). This work has inspired applications in the automatic synthesis of asynchronous circuits [CKKLY00] and synthesis of distributed transition systems [CMT99]. Nonetheless, the approach lacks a flexible specification language, independence between actions is not explicit and it has no abstract notion of action such that different system actions correspond to the same abstract action.

Mazurkiewicz traces and asynchronous automata

A notion of trace was proposed by Mazurkiewicz [Maz77] for the study of concurrent systems and constitutes now a classical subject (see [DR95] for the extensive research over the years on traces). A concurrent behaviour is simply captured enriching a set of actions with the information about independence of actions, denoted $\|$. The notation $\|$ aims at capturing the fact that two actions that are independent could be executed in parallel and could therefore appear in a computation in any order. The executions of a distributed system can then be naturally grouped together into equivalence classes, where two computations are equated in case they are two different interleavings of the same partial order stretch of behaviour. A *trace* is just such an equivalence class of computations. A *trace language* is a languages closed under the independence relation $\|$. It is clear that the behaviour of a distributed system is a trace language. A trace language is *regular* if it is accepted by a conventional finite state machine.

Asynchronous automata were introduced by Zielonka [Zie87] in order to provide a computational model for the regular trace languages. Zielonka's result solved a problem debated for a long time, namely finding a distributed model that has the behaviour given by a regular trace language. The asynchronous automata consist of a set of local automata that periodically synchronize according to a communication structure in order to process the input.

We choose the notions of traces and asynchronous automata as theoretical foundations for the proposed synthesis of distributed systems. There are meaningful results characterizing their relation and they are naturally modeling the behaviour of a concurrent system which involves independence of actions. To support this statement, let us return to the previous example of mutual exclusion and see what is the solution based on traces. We start with the same set of actions as before and we add to the specification the independences between actions such that both processes should independently request access and exit the critical section. Therefore $req_1 \parallel req_2$ is part of the specification. Then we describe the properties of mutual exclusion, absence of starvation, deadlock freedom and local behaviour of processes in terms of regular expressions. For example mutual exclusion condition should forbid from the behaviour sequences like $\Sigma^* enter_1(\Sigma \setminus exit_1)^* enter_2 \Sigma^*$ (second process cannot enter the critical section if the first process was entering the critical section, but did not exit). We obtain the global automaton in the right column of Table 1, which we see that it has a correct behaviour in the situation both processes request access (the sequences $req_1 req_2$ and $req_2 req_1$ lead to the same global state marked with an 'x'). It is now essential that, because of Zielonka's result, the global automaton is distributable, so that there exists an asynchronous automata exhibiting precisely the given behaviour.

3 Proposed approach

The aim of the proposed research is to develop an automatic synthesis procedure for distributed systems having a flexible specification language and a reasonable computational complexity such that applications to real case studies are made possible. The different stages of this research are given below.

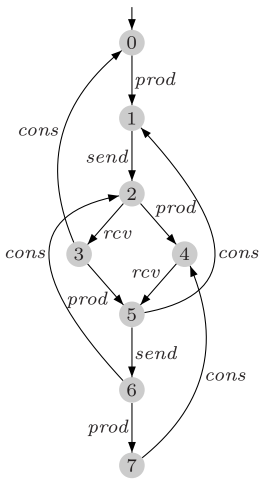
Specification language The final goal is to have a specification language based on temporal logics enriched with information about independences. The behaviour and the independences are initially specified separately. The behaviour will be given either as a regular expression or an LTL formula (depending on the studied model), whereas the independence will be given as a binary relation. An important problem is that of consistency between these two parts: if the specification enables a computation of the system, then all the computations equivalent with respect to the independence should be enabled by the specification.

In a unified manner, we would like to have a unique logic that is able to express both behaviour and independences. We will profit from the current efforts to find distributed versions of temporal logic capable of expressing concurrent behaviour (see in [TH98] a recent survey on distributed versions of linear time temporal logic LTL) and as a candidate we will consider monadic second order logic on strings.

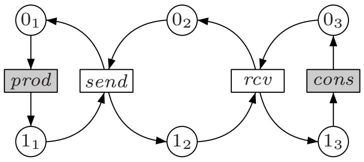
Synthesis procedure Zielonka's procedure is a strong theoretical result, but unfortunately its complexity is superexponential, hence it outputs very large asynchronous automata. This is not surprising as state space explosion is a common problem for concurrent systems. A challenge is to find simplifications, running-time improvements and heuristics for the algorithm. Also, we are looking for nice classes of distributed systems, which are rich enough to include interesting examples for which we can develop simpler versions of the algorithm.

Unfoldings are a successful approach based on branching time partial order semantics (see [ERV96], for example). Given a language and an independence relation, we can easily construct a (usually infinite) unfolding. The problem, however, is to find a good criterion of identifying states that can result in a finite machine. Our experience on some examples indicates unfoldings as a promising idea in our framework.

Implementation The procedure, which will eventually emerge from the presented research, should receive machine support enabling users to deal with a large state space. Implementation of this should include procedures for

<table>
<tr><th>Global automaton</th><th>Distribution</th><th>Result of synthesis</th></tr>
<tr>
<td>
</td>
<td>independences
$prod \parallel cons$
$prod \parallel rcv$
$send \parallel cons$

processes
$\{P_1, P_2, P_3\}$

<table>
<tr><th></th><th>exec. by</th></tr>
<tr><td>$prod$</td><td>P_1</td></tr>
<tr><td>$send$</td><td>P_1, P_2</td></tr>
<tr><td>rcv</td><td>P_2, P_3</td></tr>
<tr><td>$cons$</td><td>P_3</td></tr>
</table></td>
<td>Synthesized asynchronous automaton


Translated distributed algorithm:

[**Producer**]:
repeat forever
produce
if (*buffer*=empty) **then**
buffer := filled
end repeat

[**Consumer**]:
repeat forever
if (*buffer*=filled) **then**
buffer := empty
consume
end repeat</td>
</tr>
</table>

Table 2. Synthesis at work for the Producer-Consumer example

checking the consistency of the specification, closing regular behaviours under a given independence relation, synthesizing distributed systems using the improved version of Zielonka's algorithm as well as the original construction (to compare the results).

Applications The developed theoretical and practical tools should prove their qualities on real case studies like small distributed algorithms and asynchronous circuits.

An instance of synthesis for distributed algorithms is: given all possible runs of an algorithm and a distribution structure, construct a distributed algorithm having the described behaviour. The literature is rich in challenging problems such as mutual exclusion, resource allocation (e.g. dining philosophers), cache coherence and communications protocols (e.g. alternating bit protocol). In addition to the automatic synthesis procedure, further attention will be given to general guidelines on how to specify the problem (which is not always an easy task, for example for extracting information on independences needed for the synthesis).

Asynchronous circuits are an established target in hardware design. We will compare our procedure with the one proposed using Petri nets in [CKKLY00].

4 A simple example

Just to show how the procedure would work, we synthesize a solution for a simplified version of the producer-consumer problem.

We specify the trade between a *producer* and a *consumer* considering the actions of producing, sending, receiving and consuming $\Sigma = \{prod, send, rcv, cons\}$. The regular specification S is given by:

- $S \subseteq$ shuffle$((prod\ send)^*, (rcv\ cons)^*)$, ($S$ is included in the interleaving of the behaviours of the 'producer' and 'consumer')

- $S \subseteq (T^* send T^* rcv T^*)^*$, with $T = (\Sigma \backslash \{send, rcv\})$ (the consumer can only receive something that was previously sent)

In the left column of Table 2 we have the global automaton obtained from the intersection of the previous regular expressions. In the middle column we have a natural choice for independences between actions. We can then distribute the actions to be executed by three processors such that two independent actions are executed by disjoint sets of processors. We apply the synthesis procedure and we obtain the asynchronous automaton depicted in the right column which consists of the synchronization of three local automata with the states $Q_i = \{0_i, 1_i\}$, i=1,2,3 and the global initial state $(0_1, 0_2, 0_3)$.

The result is as we would have expected: the modeling of the trade using a buffer of capacity one modeled by process P_2. If we see the local states of P_2 (0_2 and 1_2) as the states of a buffer, which is initially empty, then we can translate the asynchronous automata in the distributed algorithm in the right column of Table 2. The translation of an asynchronous automaton into a distributed algorithm is not done automatically and further research should be carried out in order to establish if this can be done using a general algorithm.

5 Current stage of research

In this section we will address the state-of-the-art concerning ideas introduced in the previous section. We will follow the same modular structure as before:

Specification language We explored specifications given in terms of a regular behaviour together with an independence structure and we had to deal with the consistency problem:

- when regular behaviour was described as a regular language we studied recent results on closing a regular expression under an independence relation. The closure of a regular language is not regular in general, so we studied a subclass of regular languages (Alphabetic Pattern Constraints, APC) with this closure property proposed in [BMT01].

- when the regular behaviour was described as classic deterministic finite automaton, we have to ensure that the automaton satisfies some syntactic properties called independent and forward diamond rules (for example, the forward diamond rule is: for every state q such that $q \stackrel{a}{\to} q'$ and $q \stackrel{b}{\to} q''$ and $a \parallel b$, there exists a state q''' such that $q' \stackrel{b}{\to} q'''$ and $q'' \stackrel{a}{\to} q'''$). A procedure was developed to safely remove the transitions which made the automaton inconsistent with respect to the independence relation.

Synthesis procedure We invested a great deal of time in the understanding of Zielonka's construction and proof of correctness, which is notoriously hard. We tried various ideas in order to simplify it, but until now only minor improvements have been achieved.

Yet, there are some hopes related to the idea of resolving the problem using unfoldings. The trick is to combine unfoldings with a recent result found by Morin [Mor99], which gives a polynomial test for checking whether a given deterministic automaton together with a distribution structure is isomorphic with an asynchronous automaton. We consider two cases:

- when the behaviour of the distributed system is *finite* we have a method using unfoldings for synthesizing an asynchronous automaton (we combine Morin's result with the fact that a finite behaviour generates a finite unfolding). A possible idea is to use the algorithm in the area of message sequence charts (MSCs), which have a finite behaviour.

- when the behaviour of the distributed system is *infinite* we found a heuristic based on unfoldings using both Morin's test and conditions in Zielonka's construction. The heuristics worked well for many examples and drastic reductions of the state space were obtained. Unfortunately, if the solution requires a large number of iterations, the heuristics blows up the state space.

Implementation We should mention that we are not aware of any implementation for synthesis of asynchronous automata. The following have so far been implemented:

- the procedure of closing an APC under an independence relation from [BMT01]

- Morin's polynomial test

- Zielonka's algorithm

- a heuristic for Zielonka's algorithm.

Because an asynchronous automaton can also be seen as a 1-safe Petri net, another idea is to investigate and implement an algorithm for reducing Petri nets based on the so called place bisimulation (see [SS00] for details).

Applications We only considered examples from the area of small distributed algorithms. For instance, we were able to synthesize two new versions for mutual exclusion problem for two concurrent processes based on the global automaton in the right column of Table 1. In the first version, the processes are communicating using two shared variables that are both able to read and write. In the second (improved) version, we give priority to one of the processes and we are also using two shared variables, but now at each step the processes are reading or writing only one variable. In the draft paper [ŞE] we are presenting the automatic synthesis method developed so far applied on case studies like mutual exclusion and dining philosophers. In section 4 we succinctly presented a synthesis for a simplified version of the producer-consumer problem.

A notable collateral result was obtained while working on the synthesis problem. We exactly characterized the subclass of languages accepted by safe deterministic asynchronous automata [Zie89] as the class of languages accepted by (classic) deterministic automata satisfying the diamond rules ID and FD. These languages are suitable to express global behaviour of distributed algorithms. The details are given in the draft paper [ŞE].

Regarding the expectations about how well the approach will scale to different problems, we hope to have good results when dealing with small examples, but we cannot predict anything for larger problems because we have an exponential blowup in general for the synthesis of asynchronous automata.

6 Final hopes

As a PhD is a combination of experimentation and persistence, it is natural to express what is hoped to be the final outcome of this research:

> Based on the experience gained experimenting with different toy examples for which we were able to synthesize nice solutions, and the attempts using unfoldings, we **hope** to find a consistent and feasible approach for synthesis of distributed systems and to build another necessary bridge between theory and practice.

Acknowledgements The described work is carried out at the Laboratory for Foundations of Computer Science, University of Edinburgh, under the supervision of Professor Javier Esparza and is supported by an EPSRC Grant.

References

[BD98] E. Badouel and Ph. Darondeau. Theory of regions. LNCS 1491 (1998) 529-586.

[BMT01] A. Bouajjani, A. Muscholl, and T. Touili. Permutation Rewriting and Algorithmic Verification. In *Proc. LICS'01, IEEE Computer Society* (2001) 399-408.

[CKKLY00] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Hardware and Petri Nets: Application to Asynchronous Circuit Design. In *Proc ICATPN'00*, LNCS 1825 (2000) 1-15.

[CMT99] I. Castellani, M. Mukund, and P. S. Thiagarajan. Synthesizing distributed transition systems from global specifications. In *Proc. FSTTCS* 19, LNCS 1739 (1999) 219-231.

[DR95] V. Diekert and G. Rozenberg (Eds.). *The Book of Traces*. World Scientific, 1995.

[EC82] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming* 2 (1982) 241-266.

[ER90] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures I and II. *Acta Informatica* 27(4) (1990) 315-368.

[ERV96] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proc. TACAS'96*, LNCS 1055 (1996) 87-106.

[KV01] O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, (2001).

[MW84] Z. Manna and P. Wolper. Synthesis of communicating processes from temporal logic. ACM Transactions on Programming Languages and Systems (TOPLAS), 6(1) (1984) 68-93.

[Maz77] A. Mazurkiewicz. Concurrent program schemes and their interpretations. Technical Report DAIMI Rep. PB-78, Aarhus University, 1977.

[Mor99] R. Morin. Hierarchy of asynchronous automata. In *Electronic Notes in Theoretical Computer Science* 28 (1999) 59-75.

[PR89] A. Pnuli and R. Rosner. On the synthesis of an asynchronous reactive module. In Proc. 16th ICALP, LNCS 372 (1989) 652-671.

[ŞE] A. Ştefănescu and J. Esparza. Synthesis of distributed algorithms using asynchronous automata. *Draft*.

[SS00] Ph. Schnoebelen and N. Sidorova. Bisimulation and the reduction of Petri nets. In *Proc. ICATPN'00*, LNCS 1825 (2000) 409-423.

[TH98] P.S. Thiagarajan and J.G. Henriksen. Distributed versions of linear time temporal logic: A trace perspective. In LNCS 1491 (1998) 643-681.

[Zie87] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. Inform. Théor. Appl.* 21 (1987) 99-135.

[Zie89] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In LNCS 363 (1989) 278-289.