DFG Research Training Group "Algorithmic Synthesis of Reactive and Discrete-Continuous Systems (AlgoSyn)"

DFG-Graduiertenkolleg „Algorithmische Synthese reaktiver und diskret-kontinuierlicher Systeme (AlgoSyn)"

Wolfgang Thomas in collaboration with Kai Bollue, Dominique Gückel, Gustavo Quirós, Michaela Slaats, and Michael Ummels, RWTH Aachen

Summary While methods of software validation and verification are by now well established, the approach of automatic synthesis of software (and hardware) is as yet only developed in quite rudimentary form. Algorithmic program synthesis is possible in restricted scenarios, in particular in reactive multi-agent systems with low data complexity and in control systems. Central issues are the establishment of system models that support algorithmic solutions, the combination of discrete and continuous parameters (in hybrid systems), and the exploration of applications. The aim of the Research Training Group AlgoSyn is to unify the expertise from computer science, mathematics, and four engineering disciplines (processor architectures, automatic control, process control engineering, train traffic systems), in order to push forward the desired integration of methods. ►►► **Zusammenfassung** Während Methoden der Softwarevalidierung und -verifikation inzwischen gut etabliert sind, ist der Ansatz der automatischen Synthese von Software (und Hardware) erst rudimentär entwickelt. Algorithmische Programmsynthese ist in eingeschränkten Szenarien realistisch, insbesondere für reaktive (Multi-Agenten-) Systeme mit eingeschränkter Datenkomplexität sowie in Leit- und Steuerungssystemen. Zentrale Probleme sind die Etablierung von System-Modellen, die algorithmische Lösungen unterstützen, die Kombination von diskreten und kontinuierlichen Parametern (hybride Systeme) und die Erschließung neuer Anwendungen. Ziel des Graduiertenkollegs AlgoSyn ist es, diese Forschungen durch Bündelung der Expertise aus Informatik, Mathematik und vier Ingenieurdisziplinen (Prozessorarchitekturen, Regelungstechnik, Prozessleittechnik sowie Schienenverkehrswesen) voranzutreiben und zu der notwendigen Methodenintegration beizutragen.

Keywords D.2.2 [Software: Software Engineering: Design Tools and Techniques]; Algorithmic game theory, reactive systems, automatic control, hybrid systems, multi-agent systems, formal methods, model-based software development, process control engineering, automatic control ►►► **Schlagwörter** Algorithmische Spieltheorie, reaktive Systeme, hybride Systeme, Multiagenten-Systeme, Modell-basierte Softwareentwicklung, Steuersysteme

1 Introduction

The construction of correct and efficient hardware and software systems is one of the central challenges in computer science. Formal methods have by now a long tradition and a record of convincing success in verification. In particular, the method of model-checking has reached a status where large-scale industrial applications are possible. However, verification always has a flavour of "a posteriori analysis", in which a possibly faulty program or system (for a given specification) is the starting point, and subsequent development steps are devoted to the elimination of bugs.

The main objective of the research training group AlgoSyn is to overcome the paradigm of verification a posteriori, but rather to devise methods for the algorithmic synthesis of systems from specifications. This highly ambitious approach involves much more complex tasks than verification, and in a large range of application domains one faces immediately well-known phenomena of undecidability or high complexity which prohibit a clean algorithmic solution. Nevertheless, the approach of algorithmic synthesis has turned out to be possible in a number of restricted scenarios which are interesting from a methodological viewpoint and significant in real applications. In particular, this is true for all systems in which the flow of control is more dominant than the transformation of data, as in control systems, communication protocols, and process engineering.

A methodology of automatic synthesis exists so far only in quite rudimentary form. There are prominent special examples where an algorithmic solution of the synthesis problem is known. For example, automatic synthesis is possible for reactive finite-state programs with specifications in terms of regular properties (which cover, for example, properties expressible in linear time temporal logic). These results belong to automata theory and are part of the vast and fastly increasing field of algorithmic game theory. In real-life applications, this approach has to be integrated with other methodologies, among them the idea of hierarchical system structure and, as a related aspect, techniques of stepwise refinement. Another crucial feature of application scenarios is the combination of discrete with continuous components of models (or specifications), leading to the concept of hybrid system. Here the aspect of continuous data can enter in a variety of ways: in terms of probabilistic system properties, by including timed specifications, or by modelling behaviour using differential equations rather than automata. In this wider context, the task of algorithmic system construction often involves some kind of optimization problem (where the system is required to minimize or maximize parameters like probabilities, time bounds, or thresholds of other continuous physical values like pressure or temperature).

2 The Structure of AlgoSyn

The research areas addressed by these questions span a wide spectrum. First, several disciplines of theoretical computer science are relevant, regarding the development of algorithmic approaches and adequate formal models of reactive systems or multi-agent systems. Over this basic layer, the embedding into the software development process has to be realized, using different (and usually much more complex) formalisms for describing system behaviour. A third layer is represented by concrete application areas, which in AlgoSyn come from engineering disciplines like process control engineering, hardware design, or railway engineering. Figure 1 gives an overview.

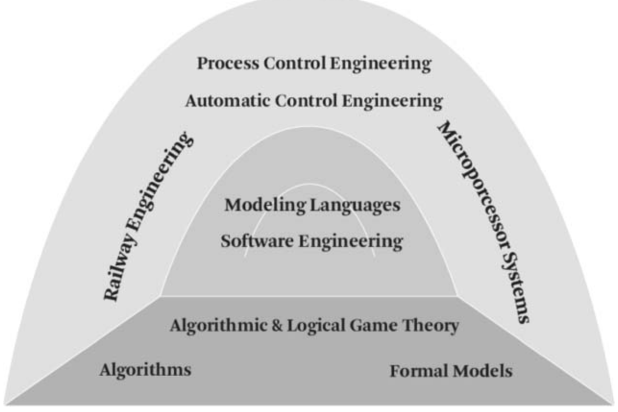

Figure 1 Structure of AlgoSyn.

At RWTH Aachen University, all branches of such a highly interdisciplinary objective are represented, and so it was natural to form a research training group with a potential to help integrating foundational work with approaches from engineering. A group of ten professors was formed, with one professor from mathematics, five from computer science, and four from four different faculties of engineering (civil engineering, mechanical engineering, material science, and electrical engineering/information technology). In the third year of the funding period, a new position of junior professor in the domain of hybrid systems was adjoined to AlgoSyn. In order to have an intensive training "on the spot", the young researchers funded from AlgoSyn are integrated into the respective research groups, but pursue all their joint activities together (in the regular seminar, training in supplementary skills, and special compact courses). These joint activities turned out to be essential since a main issue in the work of the research training school is the development of an understanding between, e. g., theoreticians and engineers.

3 Research Program

The problem of program synthesis is a fundamental question in computer science, and decades of research have treated numerous aspects of it. The focus of AlgoSyn is concerned only with selected topics that are suitable for an algorithmic approach.

In standard programs that transform input data into output data, the behaviour is given by the input-output relation $R(x, y)$, and the problem of synthesis is to design an algorithm A, transforming input x into output $A(x)$, such that for all x we have $R(x, A(x))$. Input and output values usually belong to an infinite data domain and are given and produced, respectively, as a whole.

In many projects of the research training group AlgoSyn we pursue an orthogonal view, as it arises typically in reactive systems. In this scenario, the parameters x and y are successively (and in an interleaving mode) produced by two agents as contributions to a "system run". The same idea underlies the concept of online algorithm. So x and y are streams (formally, infinite words over a finite alphabet) that together have to satisfy a given specification $R(x, y)$. The task of synthesis is to devise a method to produce a stream y in an online mode while x is observed, such that $R(x, y)$ holds. Many reactive systems are modelled in this way, and a prominent framework for describing desired relations R is temporal logic. In recent years, this simple model has been extended to much more complex forms. These extended models are studied in two research areas, numbered 1 and 2 below, and belonging mostly to theoretical computer science. The bridge to applications is built in two stages described in the research areas 3 and 4.

3.1 Research Area 1: Algorithmics for Agent Based, Probabilistic, and Hybrid Systems

The requirements for algorithmic solutions of control synthesis in technical systems usually have to take into account incomplete information on the behaviour of the process under control as well as a discrete-continuous dynamics of the entire system. Available approaches to algorithmic synthesis are built on simplifying assumptions and have to be extended to cope with realistic applications. This sets the task of developing complex system models that merge reactive, stochastic, and hybrid behaviour. In particular, we have to integrate various approaches, such as game theoretic methods in agent based optimization, primal dual algorithms, techniques for the analysis of Markov chains, and stochastic automata. Applications range from distributed resource management to job shop scheduling, and routing in networks. Examples of research topics are approximate solutions of complex optimization problems, synthesis of cost-optimal scheduling strategies, and the synthesis of hybrid systems.

3.2 Research Area 2: Formal Models and Game-Theoretic Methods

A simple but powerful model for basic reactive systems (consisting of the two components "controller" and "environment" with discrete, non-terminating behaviour) is that of an infinite two-person game. An infinite play of the game amounts to a system run that is built up in alternation between these two players "controller" and "environment", and a specification of the system amounts to a winning condition of controller on such infinite plays. In an abstract sense, the winning condition is a language of infinite words. For the class of finite-state systems and the so-called regular winning conditions, one knows how to check whether a specification is realizable by a controller, and in this case it is possible to construct a controller that ensures the specification for any choice of behaviour by the environment. This beautiful classical result (due to Büchi and Landweber) has two essential defects: The complexity of the known procedures is too high to allow applications of interesting size, and the game theoretic model is too simple for many interesting application domains since phenomena like infinite state spaces, incomplete information, the existence of more than two players, real-time requirements, and non-discrete aspects in the winning condition have to be taken into account. The extension of the existing methodology with the aim to overcome these defects is a chief objective of this research area of AlgoSyn.

3.3 Research Area 3: Software Engineering and Modelling Languages

This research area addresses the interface between the foundational studies in AlgoSyn and engineering disciplines. It is well known, for example from experiences in the development of control software systems, that this is a highly nontrivial task and that the integration of progress in algorithmics and theoretical foundations into the development process takes a long time. Not only a systematic embedding of new methodologies into the development process is lacking. Equally problematic is the discrepancy between the simple and clean formalisms studied in theoretical computer science (transition systems, temporal logics, timed automata, etc.) and the much more complex shape of formalisms used in the industrial context, as found in languages like UML or Esterel. Hierarchical structures and the merge of discrete and continuous aspects are typical reasons for this complexity. We address this problem in selected and relevant cases aiming at algorithmically manageable abstractions that allow – at least in interesting partial domains – the application of synthesis methods developed in the foundational research areas 1 and 2.

3.4 Research Area 4: Applications and Demonstrators

This research area is divided into four topics; for each of them we give a short outline.

Design Tools for Multiprocessor Systems

In the development of embedded systems we currently observe a shift from custom ASICs to programmable application specific processors (ASIPs). Architecture and instruction sets of ASIPs are tuned to specific needs of an application domain; they also represent a good compromise between flexibility and efficiency. Although a number of software tools exists that support ASIP design, the current methodology relies to a large extent on the principle of trial and error. Another challenge is the emergence of heterogeneous multiprocessor systems on chip (MPSoC). For these, the choice of optimal processing elements and communication structures is a dominating problem. As for ASIPs, the state of the art in MPSoC design rests on simulation, profiling/analysis and an ad hoc interaction by the human developer. There is an urgent need to automate this process by the construction of synthesis tools. They should cover, for instance, the allocation of processes to MPSoC components, subject to restrictions on cost, performance, efficiency, and real-time constraints.

Process Control Engineering
The main task of process control engineering is to control chemical engineering processes according to production requirements. The required control functionality is usually implemented as a network of individual control functions. These are designed, implemented, and operated individually for every plant and for every process. Current research topics in process control engineering are:

- integration of digital field devices with their increasing functionality;
- a merger of current production management systems with the system services of process control engineering;
- further development of continuous and sequential control functions into qualitative high-class robust self-configuring components;
- conceptual design of software service agents for supporting operations flexibly by executing extensive testing and diagnostic tasks;
- consolidation of the properties of hardware and software components in product data systems for electronic management and electronic trade;
- formalisation and automation of the design and implementation process itself.

A set of requirements for further development of process control methods and concepts may be derived from these objectives. These requirements refer to the advancement of the way in which control software is itself organised. For instance, the concepts for the integration of function block and sequential descriptions must be further developed. This demands an extension of the communication possibilities as well as the introduction of modern design methods within the functional units. In addition to the requirements concerning the architecture of the automation software itself, requirements for managing process control systems and handling their workflow may also be derived from the objectives mentioned above.

Automatic Control
Complex technical systems are usually built from numerous components that together realize discrete and continuous behaviour. Current design methods handle these two aspects of behaviour separately, i. e., successively, largely ignoring mutual dependencies that may occur. Therefore, a central objective of research in automatic control is an integrated treatment of control, both regarding correctness and optimization of efficiency of the system under consideration. Closely related are the tasks of devising techniques for an adequate decomposition of specifications and systems, the development of interfaces between components, and the design of distributed control systems. While there are well-established and mathematically clean methods for designing systems with continuous behaviour, most available techniques for the construction based on discrete event systems are far from efficient and practicable. We address these challenges in typical scenarios, using the Petri net state space model (in the format according to IEC 61499). A crucial first step is the automatic synthesis of discrete controllers, using this rich formalism for specification and implementation. A model plant at the Aachen Institute of Automatic Control serves as a test environment (see Fig. 2).

Figure 2 Model Plant at RWTH Aachen Institute of Automatic Control.

Railway Engineering
Due to the safety requirements of railway operations a train control system is indispensable. The train control system determines the way how train operations take place on a railway system. The deterministic behaviour of the railway system can be modelled by discrete event systems. For the capacity planning of railway systems these models are fed with worst case assumptions regarding the underlying railway safety system in order to guarantee safe operation. In particular, for large railway nodes of a railway system, these worst case assumptions lead to an underestimation of available railway capacity. Incorporating stochastic models for the capacity planning instead of worst case assumptions is an approach that may lead to a more detailed view on the railway system and thus to exploit the capacity in large railway nodes more efficiently. The focus of our research is the development of a hybrid model, combining the deterministic model for railway operation with a stochastic model for the railway safety system in order to synthesize optimised train timetables that make better use of the available railway capacity. This research is supported by a laboratory test

bed installation that is used for the simulation of railway operation in the Institute of Transport Science of RWTH Aachen University.

4 Five Example Projects

In the research training group, 15 doctoral research projects are carried out. In order to give the reader a concrete impression, we present here five short presentations of ongoing research done in AlgoSyn.

4.1 Synthesis of Controllers with Nested Pushdown Store (Michaela Slaats)

The classical set-up of automatic synthesis of reactive programs is best described by the model of infinite two-person game. We call the two players Adam and Eve, where Adam stands for the (possibly hostile) environment and Eve for the controller. The game arena is a finite transition graph where each state belongs to either Adam or Eve. The game is played by moving a token through the arena along the edges, where Adam and Eve, respectively, chooses an edge to a next vertex from a vertex that belongs to him or her, respectively. An infinite play is won by Eve if it satisfies a "regular winning condition". Several equivalent definitions of regular winning conditions exist (in terms of Büchi automata, logical formulas, or other kinds of expressions); a very useful normal form is the so-called parity condition. It is known that these finite-state games for regular winning conditions can be "solved": One can compute whether Eve has a winning strategy for plays from a given initial vertex of the game arena, and one can compute such a winning strategy in the format of a finite input-output automaton. This solvability result (going back to Büchi and Landweber in 1969) is the starting point of the algorithmic theory of infinite games.

We address the problem how to extend this algorithmic result to infinite game arenas. Usually they arise by attaching some kind of infinite store to a given finite arena. It is known (by work of Walukiewicz) how to do this if the store is a pushdown stack. It is also obvious from the fundamentals of computability theory that the attachment of two pushdown stacks prohibits an algorithmic solution of the corresponding games. In our work we investigate another structure of "multiple stacks", namely nested stacks. A level-1 stack is a standard stack, a level-2 stack is a stack of stacks, and so on. This kind of storage is needed, for example in the implementation of higher-order recursive programs. Even in this quite general framework, it is possible to provide an automatic synthesis of winning strategies and hence of suitable controllers.

The main problem in solving games over arenas that involve nested stacks (or counters) is to develop an appropriate format of controller (again with the same storage structure) that has enough power to implement possible winning strategies. We have developed a synthesis method for such controllers based on generalized concepts of "regular stack language" [1; 2], and we analyze the applicability in several domains, including scheduling problems.

4.2 Synthesis of Nash Equilibria in Infinite Games (Michael Ummels)

The algorithmic theory of infinite two-player zero-sum games has proven very fruitful in synthesizing controllers for monolithic reactive systems. However, a system that consists of several components is best modelled by an infinite game with several players and not necessarily opposing objectives. Consider, e. g., the following situation: There are two processes that share a single resource (e. g., a printer). If the resource is not busy, the two processes are polled alternately. If one of them wants to use the resource, the resource becomes blocked, and it only becomes available again when the process which has blocked it releases it. Moreover, let us assume that every process needs to use the resource infinitely often to complete its task. This situation is naturally modelled by an infinite game with two players whose objectives are not conflicting, which is depicted in Fig. 3.

The classical solution concept offered by game theory for games with multiple players is that of a *Nash equilibrium*. A profile of strategies, one for each player, constitutes a Nash equilibrium if no player can gain from switching to a different strategy (while all other players stick to their strategy). In our example, there are several Nash equilibria. In particular, there is a Nash equilibrium where only the first process (i. e., daemon) has access to the resource infinitely often and ones where both processes do: If each time a process has access to the resource, it blocks the resource for a finite amount of time and then hands back control to the other process, both processes have access to the resource infinitely often. Clearly, the latter solution is preferable to one where only one process meets its specification.

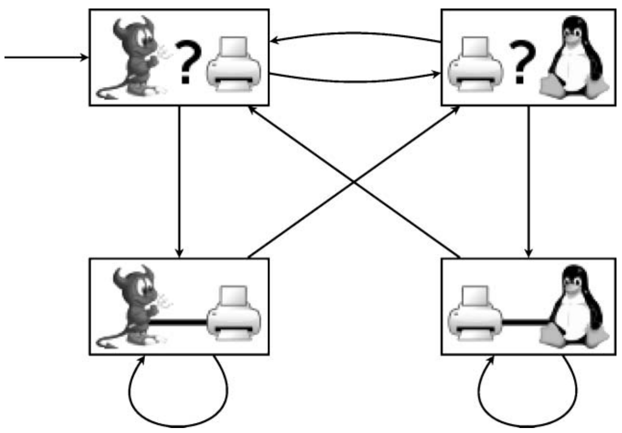

Figure 3 Two processes (penguin and daemon) sharing a common resource. Both the penguin and the daemon want to be connected to the resource again and again.

In general, we aim at synthesizing Nash equilibria where certain players win and certain other players lose (and for the rest, we do not care). It turns out that this is a hard task: the corresponding decision problem is NP-hard, even for very simple types of objectives [3]. If one allows more sophisticated models of systems, like systems that can change their state randomly, the synthesis problem may even become undecidable [4].

4.3 Synthesis of Tools for Software Model Checking (Dominique Gückel)

The foremost problem in model checking is to find an appropriate system model. The model has to cover every important aspect of the system. In the case of an embedded system, that may range from the software over the executing hardware to the plant controlled by the device. However, due to the potentially very high number of states encountered during the analysis, the model must not be too fine, or else the analysis will abort due to memory limitations. Having made that decision, the developer has to translate his system into the model checker's input language.

The modelling and translation steps require a lot of expert knowledge and can hardly be automated, hence they are expensive. To make model checking more applicable to real software, one can use an assembly code model checker. An example of such a tool is the [mc]square model checker. [mc]square uses a binary representation of the testee program just as it is created by a compiler. It then simulates program execution on a simulator for the target platform. Starting from the device's reset state, it successively creates all reachable states and verifies the validity of a temporal logic formula. There is no need to perform any preprocessing or provide an environment because the simulator simply assumes that all input from the environment is nondeterministic.

State representation in [mc]square is partially symbolic. This means that machine states can contain symbols which indicate unresolved nondeterminism. Delaying the resolution of nondeterminism reduces the number of states to be stored significantly and helps avoid the state explosion problem. In order to prevent state explosion, [mc]square also supports static analyses. These can be used to figure out useful information about the testee, for example which memory locations are irrelevant and do not need to be stored.

Even though the approach used in [mc]square facilitates model checking from the developer's point of view, it has a severe disadvantage, namely the hardware dependency. For each platform to be supported, the tool requires a simulator. Experience values indicate that it takes about six months to implement such a simulator for a new microcontroller or programmable logic controller. In order to reduce this effort, we conduct research on architecture description languages (ADLs) [5]. Such languages are normally used in design space exploration. Our goal is to use an ADL for algorithmically synthesizing

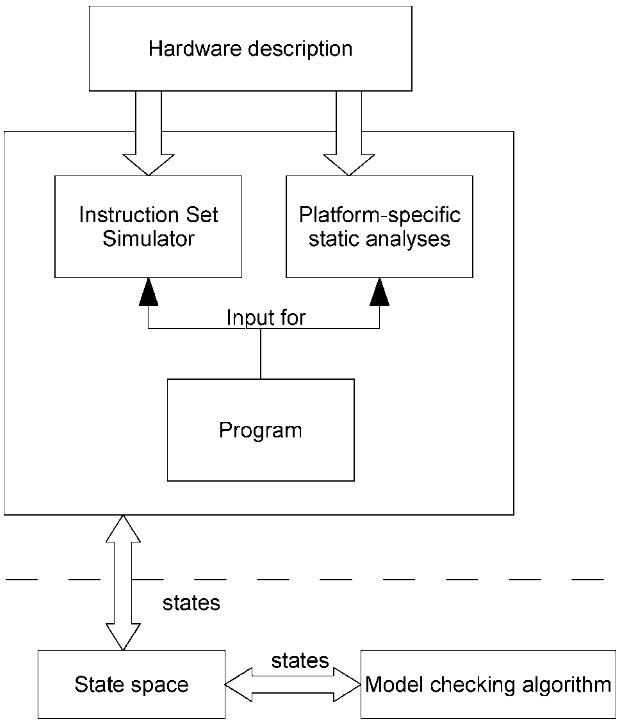

Figure 4 Separating hardware-independent from hardware-dependent constituents of model checking.

both simulators and appropriate static analyses. Figure 4 illustrates the process. The parts of the diagram above the dotted line indicate the hardware-dependent part of the model checking process in [mc]square, whereas the parts below the line are the hardware-independent part. The long term goal of our research is a retargetable assembly code model checker.

4.4 Controller Synthesis for Discrete Event Systems Using Petri Nets (Kai Bollue)

As a benchmark for our approach to controller synthesis, the model plant installed at the Aachen Institute of Automatic Control is used. While the plant's hardware is bench-scale, the control technology including PLCs (Programmable Logic Controllers), programming software etc. complies to current industry standards. As a typical scenario of industrial production, the plant demonstrates the processing of a liquid as well as its bottling and thus the handling of bottles and caps. Hence it includes both continuous dynamics and parts driven by discrete controllers. The first segment of the plant (model station, see Fig. 5) forms a typical example of a discrete event system. Its purpose is to supply the subsequent station with caps in the correct orientation. The station uses two separators, an orientation sensor and a pneumatic gripper (with three degrees of freedom: up or down, open or closed, and turned left or right) to determine and – where neccessary – alter the caps' orientation.

Our goal is to develop a method to automatically generate discrete controllers for certain parts of the plant out of a model of the uncontrolled plant and "goal

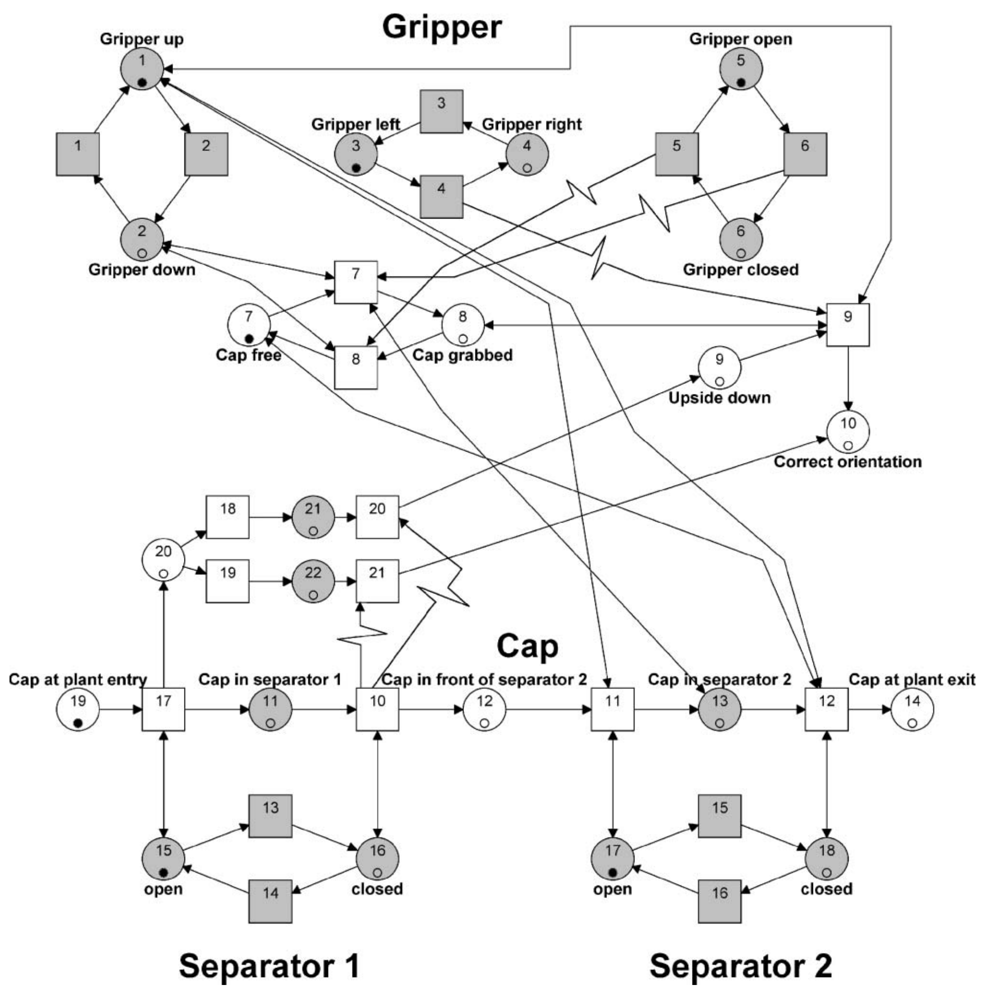

Figure 5 Petri net model of two separators, orientation sensor, and gripper.

specification" (i. e., descriptions of the system's desired behaviour) as well as safety constraints. The resulting controllers can then be translated into code which can directly be used for the PLCs.

In our approach we use Petri nets with certain augmentations for modelling the plant and as a basis to define safety and goal specifications. These augmentations include additional arc types (like event arcs) as well as methods to model the possibilities of the controller to influence the system (controllability) and gain information about it (observability).

First the uncontrolled plant is modelled, i. e., all possible behaviours of the system are considered – not only the desired ones. Here the mentioned augmentations to classical Petri nets provide the possibility to create a modular and intuitive model (see also [6]). Transitions in the model are declared to be either controllable (i. e., the controller can block or enforce the transition) or uncontrollable, while places are declared as either observable (i. e., the controller is able to determine the current marking of the place) or unobservable. Additionally, goal and – if needed – safety specifications are given by linear constraints on the marking of the Petri net.

To generate a control algorithm, first the model together with the specifications is converted into a set of so called unified transitions, each of which consists of a set of linear marking constraints as preconditions and a set of marking changes as firing effect. As the safety and goal specifications are also given by linear marking constraints, they seamlessly fit into this framework. Now, beginning at the start marking, possible paths through the marking space are searched for by recursively defining the preconditions of helpful unified transitions as temporary goals and applying the algorithm to these. The selection of considered transitions is made based on the marking changes of the transitions with respect to the constraints currently to be fulfilled. Hence a guided search is implemented, enabling the algorithm to deal with the potentially exponentially large search space and thus also work for non-academic examples. For further details see [7].

4.5 Model-Based Synthesis of Product Flow Path Management Systems (Gustavo Quirós)

A basic and essential operation performed by processing plants is the movement of material, i. e., products, between plant elements. This movement may occur along routes which are given by the structure of the plant. We denote these routes *product flow paths*. In modern plants, process control systems usually fulfill the task of ensuring and monitoring the correct and safe transport

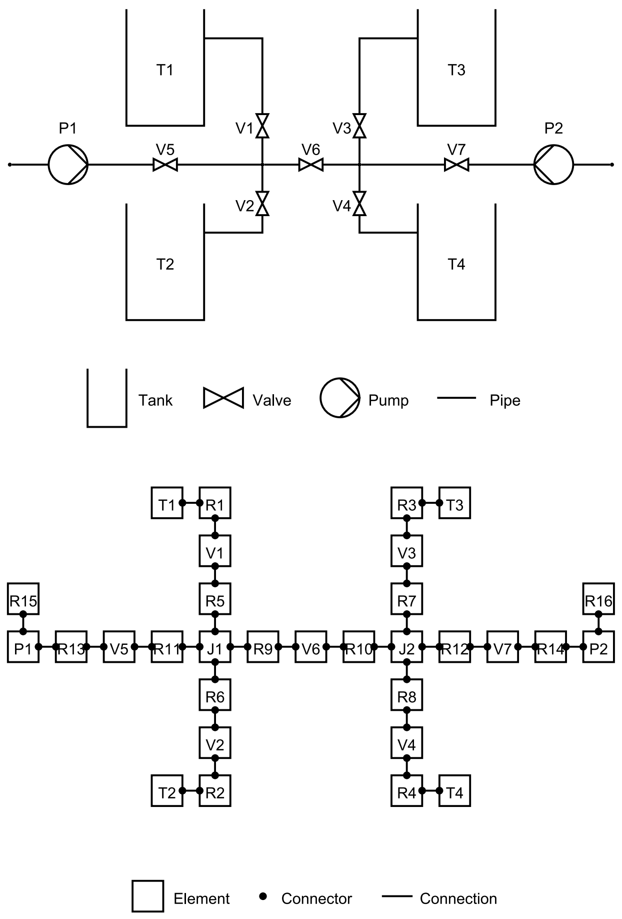

Figure 6 Simple plant and corresponding abstract model.

of material. However, the development of these plant-specific solutions is usually based on informal knowledge, is time-consuming and error-prone, and must be repeated as soon as the plant itself changes. Furthermore, no model for the explicit description of product flow paths is currently known. This motivates the development of automated model-based approaches that may partly or completely replace this engineering work, ensuring its correctness at the same time. With this goal, we have developed a formal and abstract plant model, based on the RIVA model [8] which defines a simplified plant representation that considers the possibility of flow through its components, and is generic enough to represent practically any type of plant and plant device. This model serves then as a base for defining a formal model of product flow paths which may be used to specify and automatically construct related control functionality, which we collectively denote as *product flow path management*.

Using our abstract plant model, the structure of a plant is formally represented based on sets of elements and product connectors. A mapping associates connectors to elements, and a binary relation represents the interconnection of element connectors as is found in the physical plant. Figure 6 shows a simple plant and its corresponding abstract model, in graphical form.

Our model describes the possibility of physical flow of material into and out of every plant connector by means of a *flow allowance setting*, as well as the allowance of flow *among* neighboring plant elements by means of a binary *flow allowance relation* for a given flow allowance setting. A product flow path is defined as a sequence of neighboring plant elements which follows the flow allowance relation, and which may therefore be used by a product to flow from an initial element to a final element.

Systems for product flow path management which operate in a decentralized fashion may be specified using our modeling approach, and may be automatically constructed for a plant given a machine-readable representation of a corresponding abstract plant model and flow allowance model. We follow a decentralized component-based scheme as presented in [9; 10] where every plant element is assigned a component of the system which controls and monitors the element exclusively, and which has connection ports for every corresponding connector. These ports are interconnected through bidirectional communication links in accordance with the connection relation of the abstract plant model. Thus, the structure of the decentralized system is an analogy of the plant layout. Each component interacts with each of its neighboring components by sending and receiving messages, and the components work cooperatively in order to achieve the system's goals. A realization of this decentralization scheme may be accomplished using industry-standard function blocks for the components. Using this approach, such a system may be automatically constructed from a model of the plant by instantiating, parametrising and linking component blocks, all of which are common operations in process control systems. This offers a simple and effective technique for synthesizing flow path management systems.

5 Conclusion

The reader will find more information, including the running seminar program and all relevant references, on the AlgoSyn website www.algosyn.rwth-aachen.de.

Acknowledgements

The present exposition of AlgoSyn would not have been possible without the unfailing, efficient, and expert help of Marianne Kuckertz, secretary of Lehrstuhl Informatik 7, and Helen Bolke-Hermanns, the administrative assistant of AlgoSyn.

References

[1] A. Carayol and M. Slaats. Positional strategies for higher-order pushdown parity games. In: MFCS 2008, vol. 5162 of LNCS, p. 217–228, Springer, 2008.

[2] P. Hänsch, M. Slaats, and W. Thomas. Parameterized infinite games and higher-order pushdown strategies. AutoMathA Conf. 2009, Liège, Belgium, 2009 (to appear).

[3] M. Ummels. The complexity of Nash equilibria in infinite multiplayer games. In: Proc. of the 11th Int'l Conf. on Foundations of Software Science and Computation Structures, FOSSACS 2008, vol. 4962 of LNCS, p. 20–34. Springer, 2008.

[4] M. Ummels and D. Wojtczak. The complexity of Nash equilibria in simple stochastic multiplayer games. In: Proc. of the 36th Int'l Colloquium on Automata, Languages and Programming, ICALP 2009. Springer, 2009. (to appear).

[5] D. Gückel, Retargeting a hardware-dependent model checker by using architecture description languages. In: Proc. of the 4th Int'l Workshop on Systems Software Verification (SSV 09), Doctoral Symposium, Aachener Informatik-Berichte 2009 (to appear).

[6] H.-M. Hanisch, A. Lüder, and J. Thieme. A modular plant modeling technique and related controller synthesis problems. In: 1998 IEEE Int'l Conference on Systems, Man, and Cybernetics, 1998, vol. 1, p. 686–691.

[7] K. Bollue, D. Abel, and W. Thomas. Synthesis of behavioural controllers for discrete event systems with NCES-like Petri net models. In: European Control Conf. 2009 (to appear).

[8] R. Jorewitz, U. Epple, A. Münnemann, R. Böckler, W. Wille, and R. Schmitz. Automation of performance monitoring in an industrial environment. In: PCIC Europe 2005, 2nd European Conf. on Electrical and Instrumentation Applications in the Petroleum and Chemical Industry, p. 116–120, Basel, Switzerland, Oct 26–28 2005.

[9] G. Quirós, M. Mertens, and U. Epple. Function blocks for decentralised analysis of product flow paths. In: ETFA 2008, 13th IEEE Int'l Conf. on Emerging Technologies and Factory Automation, Hamburg, Germany, Sep 15–18 2008.

[10] G. Quirós, R. Jorewitz, and U. Epple. Model-based safety monitoring of product flow paths. In: Breitenecker F., Troch I. (eds): MATHMOD 2009: 6th Vienna Conf. on Mathematical Modelling, Vienna, Austria, Feb 11–13, 2009. AGRESIM Report Nr. 35. ARGESIM/ASIM.

Received: May 8, 2009

Wolfgang Thomas holds the chair of logic and theory of discrete systems at RWTH Aachen. His main interests are automata theory and logic in computer science, with an emphasis on generalized models of automata, infinite games, and applications in verification and synthesis. He is Doctor honoris causa of École Normale Supérieure de Cachan (France) and member of Academia Europaea.

Address: RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Tel.: +49 (241) 80 21700, Fax: +49 (241) 80 22 215,
e-mail: thomas@informatik.rwth-aachen.de

Dipl.-Math. Kai Bollue, Institut für Regelungstechnik (Prof. Dr.-Ing. D. Abel), RWTH Aachen.

Dipl.-Inform. Dominique Gückel, Lehrstuhl Informatik 11 – Software für eingebettete Systeme (Prof. Dr.-Ing. S. Kowalewski), RWTH Aachen.

Gustavo Quiros, M.Sc., Lehrstuhl für Prozessleittechnik (Prof. Dr.-Ing. U. Epple), RWTH Aachen.

Dipl.-Inform. Michaela Slaats, Lehrstuhl Informatik 7 – Logik und Theorie diskreter Systeme (Prof. Dr. W. Thomas), RWTH Aachen.

Dipl.-Inform. Michael Ummels, Lehr- und Forschungsgebiet Mathematische Grundlagen der Informatik (Prof. Dr. E. Grädel), RWTH Aachen.