Formal Synthesis of Control Functions with Supremica

Knut Åkesson, Martin Fabian and Hugo Flordal
Department of Signals and Systems
Chalmers University of Technology
Gothenburg, Sweden
Email: {knut,fabian,flordal}@chalmers.se

***Abstract*—Supremica is a tool for formal synthesis of discrete-event control functions based on discrete event models of the uncontrolled plant and specifications of the desired closed-loop behavior. By using formal synthesis of control functions the need for formal verification is reduced since the control functions are computed to automatically fulfill the given specifications, that is, they are "correct by construction". Formal synthesis is especially useful for applications where the system-requirements or the available hardware is changing frequently. The modeling framework in Supremica is based on finite automata extended with variables, guard conditions and action functions. In order to handle large scale problems Supremica uses compositional abstraction techniques, and binary decision diagrams. Since the modeling language and algorithms are generic these techniques are useful in many application areas including control functions implemented in hardware. Supremica has been used in a number of industrial research projects to synthesize control functions for industrial robots and flexible manufacturing systems.**

I. Introduction

Embedded computers are often used to implement control functions for reactive systems. Formal verification techniques, like model checking, may be used to guarantee that the control functions behave as expected in all circumstances. However, an alternative approach is to automatically synthesize control functions from high-level descriptions that are correct by construction. While formal verification techniques have been developed mainly by the computer science community, formal synthesis of control functions has been developed in the control community where reactive systems are commonly referred to as *discrete event systems* and their control functions is named *supervisor*.

The supervisory control theory (SCT) [1], [2] is a general framework for verification and synthesis of discrete event supervisors that has shown promising results. However, in order for SCT to be accepted in industry, user friendly tools able to solve large problems are critical. Supremica [3], [4] is an attempt to build an integrated development environment that is able to solve large scale supervisor verification and synthesis problems.

The purpose of this paper is to briefly present Supremica and some of the main ideas of the SCT for synthesizing control functions that are correct by construction. Supremica is constantly evolving but the latest release can always be downloaded, free for education and research, from [3].

II. Supervisory Control Theory

Reactive systems have been a research field within computer science and engineering for a long time. However, with no control theoretic background, the main focus is on *verification* of, typically already controlled, reactive systems rather than the *synthesis* of control functions for an uncontrolled system. The *Supervisory Control Theory* (SCT) [1], [2] took a control-theoretic model-based approach, applying formal reasoning on a model of the uncontrolled process, the *plant*, and a model of the desired behavior of the controlled system denoted the *specification*. From the plant and the specification a safety device, called a *supervisor*, can be automatically synthesized. The supervisor controls the plant to always stay within the limits set by the specification, by dynamically disallowing the plant to generate events that might otherwise have been generated.

The SCT proves that given a plant and a specification there will always exist an optimal supervisor guaranteeing that the specification will not be broken, while at the same time allowing the system to always fulfill its defined (sub-)tasks. Optimality concerns here restricting the given plant as little as absolutely necessary. Such a supervisor is said to be *maximally permissive*, since it allows the controlled system the largest possible amount of freedom, in terms of event-generation, within the constraints set by the plant and the specification.

The control theoretic contribution concerns the inclusion of a certain type of "controllability". The supervisor is mainly a safety device that hinders the plant from executing events that would take the controlled system outside the specified behavior. However, not all events can be hindered from occurring, some events are *uncontrollable*, and the supervisor must never (try to) disable any of the uncontrollable events. It is known, [1], that for a given specification and plant, a supervisor that guarantees that the entire specification can be achieved exists if and only if the specification is *controllable*. This means that the specification must be such that it can be enforced without having to (try to) disable any uncontrollable events. If the specification is not controllable, it is further known that a supervisor still exists, but this supervisor can only achieve a sub-behavior of the specification, namely what is known as a *controllable sub-language*. Even more, it is also known that a unique optimal such supervisor exists and

is readily calculatable, and this supervisor will achieve the *supremal controllable sublanguage* of the specification.

In addition to controllability, which is a safety property, it is desired for the supervisor to be *non-blocking*. This is a progress property enforced by the supervisor that guarantees that at least one *marked* state is reachable from any state that it allows the controlled system to reach. Marked states typically represent (sub-)tasks that the system must always be able to finish. Typically, the initial state is a marked state, guaranteeing that under supervision of a non-blocking (and controllable) supervisor the task that the system performs can be performed again and again. As above, it is known that the *supremal controllable and non-blocking sublanguage* of a specification with respect to a given plant, exists.

Though the SCT traditionally has focused on synthesis of supervisors, verification is a natural step within synthesis. Synthesis can be viewed as a series of verification tasks, where the process model (the plant) allows the automatic alteration of the suggested, and negatively verified, supervisor. In this respect, the original specification can be viewed as a first supervisor candidate; if it is verified to be correct (controllable and non-blocking) then no further processing is necessary. Thus, by construction, a synthesized supervisor will always be verified to be correct.

To summarize, a maximally permissive, controllable and non-blocking supervisor for a given specification and plant always exists but may be expensive to calculate due to the state-space explosion problem. We can also note that [5] showed that supervisory control and multi-agent planning are equivalent problems.

III. Modeling Framework in Supremica

The modeling formalism in the user interface of Supremica is called Extended Finite Automata, EFA [6]. EFA are an augmentation of ordinary automata, as defined in [7], with variables, guard formulas and action functions. An EFA modeling a stick-picking game is shown in Fig. 1. We associate the guards and actions to the transitions in the automaton. The transitions in the EFA are enabled if and only if the guard formula is true. When a transition is taken, updating actions of a set of variables may follow. For example, the event $player2\_remove\_two$ is enabled when the guard-formula $sticks > 1$ is satisfied, and if the event is executed the action function is triggered and the value of the $sticks$ variable is decreased by 2.

On a low level, Supremica currently uses ordinary automata for its computations. In, [6], it is shown how a set of EFA can be compiled into ordinary automata with the same behavior. For lack of space, the definitions of EFAs are left out here in favor of ordinary automata that will be used throughout to define the necessary concepts.

Definition 1 (Nondeterministic finite automaton): An *nondeterministic finite automaton* is a 5-tuple $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ where Q is a *finite* set of *states*; Σ, the *alphabet*, is a nonempty finite set of *events*; $\rightarrow \subseteq Q \times \Sigma \times Q$ is the *transition relation*; $Q^i \subseteq Q$ is the set of *initial states*; and $Q^m \subseteq Q$ is the set of *marked states*.

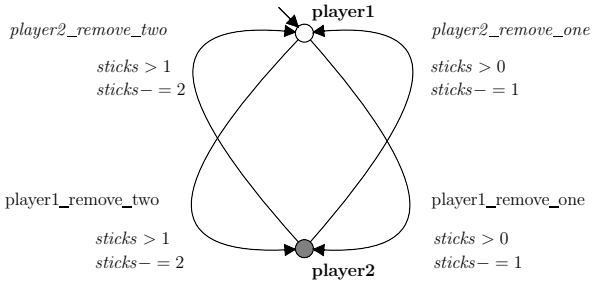

Fig. 1. EFA model of a stick-picking game with two players.

The controllability of an event is a global property and the alphabet Σ can be partitioned into the sets Σ_c and Σ_u of *controllable* and *uncontrollable* events, respectively.

The transition relation is written in infix notation, for example, $p \xrightarrow{\sigma} q$ denotes a *transition* from state p to state q associated with the event σ. This notation is further extended to strings in Σ^* in the natural way. For state sets $Q_1, Q_2 \subseteq Q$, the notation $Q_1 \xrightarrow{s} Q_2$ denotes the existence of some $q_1 \in Q_1$ and some $q_2 \in Q_2$ such that $q_1 \xrightarrow{s} q_2$.

Automata (plants, specifications and supervisors alike) running in parallel interact under lock-step synchronization in the style of [8].

Definition 2: Let $G_1 = \langle Q_1, \Sigma_1, \rightarrow_1, Q_1^i, Q_1^m \rangle$ and $G_2 = \langle Q_2, \Sigma_2, \rightarrow_2, Q_2^i, Q_2^m \rangle$ be two automata. The *synchronous composition* of G_1 and G_2 is

$$G_1 \,\|\, G_2 \;=\; \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, Q_1^i \times Q_2^i, Q_1^m \times Q_2^m \rangle \quad (1)$$

where
$$\begin{array}{l} (p,q) \xrightarrow{\sigma} (p',q') \text{ if } \sigma \in (\Sigma_1 \cap \Sigma_2),\ p \xrightarrow{\sigma}_1 p',\ q \xrightarrow{\sigma}_2 q' \,; \\ (p,q) \xrightarrow{\sigma} (p',q) \text{ if } \sigma \in \Sigma_1 \backslash \Sigma_2,\ p \xrightarrow{\sigma}_1 p' \,; \\ (p,q) \xrightarrow{\sigma} (p,q') \text{ if } \sigma \in \Sigma_2 \backslash \Sigma_1,\ q \xrightarrow{\sigma}_2 q' \,. \end{array}$$

The synchronous composition is also useful when modeling large systems because it allows the user to build multiple sub-models, and the global behavior can then be described using the sub-models and the synchronous composition operator.

The behaviour of a system may, for the purposes of this paper, be represented by its languages, i.e. the sets of strings that the system may generate.

Definition 3 (Languages): Let $G = \langle Q, \Sigma, \rightarrow, Q^i, Q^m \rangle$ be an automaton. The *language* of G, denoted $\mathcal{L}(G)$ and the *marked language* of G, denoted $\mathcal{M}(G)$ are defined as

$$\begin{aligned} \mathcal{L}(G) &= \{ s \in \Sigma^* \mid Q^i \xrightarrow{s} Q \}\,, \\ \mathcal{M}(G) &= \{ s \in \Sigma^* \mid Q^i \xrightarrow{s} Q^m \}\,. \end{aligned}$$

Now the properties controllability and nonblocking can be defined formally, in the definitions below we assume that the plant and the specification are deterministic.

Definition 4 (Controllability): Let G and K be two automata with the same alphabet Σ. K is *controllable with respect to* G if $\mathcal{L}(G \,\|\, K)\Sigma_u \cap \mathcal{L}(G) \subseteq \mathcal{L}(G \,\|\, K)$.

Definition 5 (Nonblocking): Let G be an automaton. G is *nonblocking* if $\mathcal{L}(G) \subseteq \mathcal{M}(G)$.

A supervisor S is nonblocking with respect to a plant G if $G \parallel S$ is nonblocking.

The basic supervisory control problem then concerns the following. Given a plant G and a specification K, calculate a supervisor S such that:

$$\begin{array}{lrcll} \text{i)} & \mathcal{L}(G\,\|S\,) & \subseteq & \mathcal{L}\,(G\,\|K\,) & \\ \text{ii)} & \mathcal{M}(G\,\|S\,) & \subseteq & \mathcal{M}\,(G\,\|K\,) & \\ \text{iii)} & \mathcal{L}(G\,\|S\,) & \subseteq & \overline{\mathcal{M}(G\,\|S\,)} & \\ \text{iv)} & \mathcal{L}(G\,\|S\,)\Sigma_u \cap \mathcal{L}\,(G) & \subseteq & \mathcal{L}(G\,\|S\,) & \\ \text{v)} & \mathcal{L}\,(G\,\|S'\,) & \subseteq & \mathcal{L}\,(G\,\|S\,), & \forall S' \in \mathcal{CNB}\,(G,K) \end{array}$$

Conditions i) and ii) state that the controlled closed-loop behavior must be included in the specified closed-loop behavior. Condition iii) means that the closed-loop system must be non-blocking. Condition iv) states that the supervisor must be controllable with respect to the plant. Finally, condition v) states that all other controllable and non-blocking supervisor candidates must be more restrictive than S, i.e. S has to be the maximally permissive supervisor.

IV. Computational Efficiency

Synthesizing, as well as verifying, a supervisor generally entails enumerating the state-space of a model of the controlled system. A *monolithic* approach to this, where the entire state-space is explicitly enumerated is typically intractable for systems of industrially interesting sizes. For instance, the monolithic model of a of a central-locking system for a modern car, [9], encompasses a global state-space of roughly 1 billion reachable states. Naturally, such a large state-space cannot be efficiently manipulated by explicit enumeration of the states.

One approach to defeat this "state-space explosion problem" for supervisor synthesis uses ideas from symbolic model checking, in particular binary decision diagrams, BDDs [10]. BDDs can efficiently represent and manipulate Boolean functions respresenting for example supervisory control problems.

However, typically, a plant and a specification are given in a *modular* way, as sets of interacting sub-systems. Each of these sub-systems is generally relatively small, but together they span an enormous global state-space. The model of the central-locking system mentioned above is a modular model consisting of 53 automata (18 plant and 35 specifications) the largest of which has 27 states and 80 transitions.

As the number of model-components (modules) grows, the state-space of the monolithic system grows exponentially. Thus it would be very beneficial to be able to perform synthesis (as well as verification) *modularly*; without composing the corresponding monolithic model. Several research groups have conducted research related to various aspects of this. Another way to exploit the modularity is through the *compositional* approach; incrementally calculating abstractions to simplify the problem. Supremica implements monolithic, modular, BDD, and compositional verification and synthesis algorithms for solving non-blocking, controllability, and combined non-blocking and controllability problems.

In Table I, the results of some benchmarks example calculations are shown. The table shows statistics from Supremica using modular, BDD, and compositional algorithms for solving controllability ($\mathcal{C}$) and nonblocking ($\mathcal{NB}$) synthesis and verification problems. The `AGV` example, from [11], models four automated guided vehicles that needs to be coordinated to avoid collisions and `verriegel3` is the central locking model mentioned above. The `philo` examples are models of the classical dining philosophers example with 256, 512 and 1024 philosophers respectively. The number of automata in the models is shown in the "Aut." column, the number of reachable states of the state-space can be found in the "Size" column. For the respective experiments, the columns "States" and "Trans." represent the total numbers of states and transitions examined during the computation and "Time" gives the execution time. All experiments were conducted on a standard desktop PC with a 3.2 GHz processor and 1.5 GB of RAM.

For more detailed presentations of the algorithms used in Supremica as well as further benchmarks see [12]–[17].

V. Applications

Even though the SCT theory is general and can be applied to any system that may be suitably modeled using a finite state representation, most of the applications have so far been within industrial automation. One possible explanation for this is the frequent modifications to both available production equipment and the product to be produced using the production equipment. In our research group we collaborate with both European and North American based automotive companies and with suppliers of industrial automation equipment and tools, in order to develop new techniques and tools that allows the users to faster generate control functions of high quality. Supremica is a vital component in these collaborations. Below is a brief presentation of a few projects where we are cooperating with industrial partners.

A. Industrial Robot Interlocking

In a modern car factory several industrial robots often work concurrently in a shared space in order to fulfill a given task, for example by welding the frame of the car together. In this setting, there is an obvious risk for collisions between the robots. Typically, this is handled by having a programmable logic controller coordinate the movements of the robots. However, manually generating the coordination logic is both time-consuming and error prone due to the number of possible situations that must be taken into account. By using 3D-simulation models of the industrial robots it is possible to automatically extract finite state models of the tasks to be completed and finite state models that model mutual exclusion *zones* where only a single robot is allowed to be at a time. This approach has been implemented as an add-on for a commercial robot programming and simulation environment, the coordination logic synthesis part is done using the Supremica kernel. The approach is presented in [18].

B. Manufacturing Systems

A manufacturing system typically consists of both industrial robots and other sensors and actuators. Large parts of the

TABLE I
BENCHMARK EXAMPLES FROM SUPREMICA.

Benchmark Name	Size Aut.	Size States	Comp. $\mathcal{NB}$ Ver. States	Comp. $\mathcal{NB}$ Ver. Trans.	Comp. $\mathcal{NB}$ Ver. Time	Mod. $\mathcal{C}$ Syn. States	Mod. $\mathcal{C}$ Syn. Trans.	Mod. $\mathcal{C}$ Syn. Time	Mod. $\mathcal{C}$ Ver. States	Mod. $\mathcal{C}$ Ver. Trans.	Mod. $\mathcal{C}$ Ver. Time	BDD $\mathcal{C}$ Ver. Time	BDD $\mathcal{NB}$ Ver. Time
AGV	16	$2.6 \cdot 10^7$	977	2541	0.47s	372	701	0.23s	3183	11626	0.15s	0.43s	0.40s
verriegel3	53	$9.7 \cdot 10^8$	3408	16595	1.4s	381	2005	0.47s	1049	4296	0.22s	0.65s	–
256philo	512	$5.4 \cdot 10^{168}$	14513	36632	8.9s								
512philo	1024	$2.9 \cdot 10^{337}$	29105	73508	29s								
1024philo	2048	$8.5 \cdot 10^{674}$	58289	147224	1.6m								

control logic can be automatically synthesized using high-level application-specific descriptions that allow the user to focus on describing properties of the desired closed-loop behavior. The synthesis procedure then automatically computes the detailed control logic that guarantees that all specifications are fulfilled. Supremica is used in the prototype applications that we have developed for demonstrating the feasibility of the approach. The approach is presented in more detail in [19], [20].

C. Autonomous Vehicle

The SCT is equivalent to multi-agent planning [5]. In autonomous vehicles, and especially for space rovers, it is critical that the software never fails and that all tasks may be successfully completed. By using finite state models of all tasks and resources it is possible to synthesize a supervisor that schedules the tasks such that all tasks can successfully finish while all resource constraints are obeyed. Together with a European company within the space industry the feasibility of using Supremica for generation of parts of the scheduling code has been investigated. The details regarding this project is still unpublished.

VI. CODE GENERATION

The difference between implementing control function in hardware and software is shrinking due to the flexibility of FPGAs and associated tools. Supremica includes functionality to automatically generate control code for a number of languages, including ANSI C, that implements the behavior of the synthesized supervisor. In addition it should be straightforward to generate VHDL-code that could be used to program FPGA circuits in order to implement the control function in hardware.

VII. CONCLUSIONS

A tool, Supremica, for verification and synthesis of discrete event supervisors according to the Supervisory control theory was presented. The tool implements current state-of-the-art algorithms to handle systems of industrially interesting sizes. As modeling formalism Extended Finite Automata are used. Supremica is currently used in a number of research projects together with industrial partners to facilitate the development of control functions. The use of formal methods of synthesizing control functions, that are correct by construction, address an import problem with implications for both how software and hardware control functions are developed.

REFERENCES

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event systems," *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[2] ——, "The control of discrete event systems," *Proc. IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[3] "Supremica." [Online]. Available: http://www.supremica.org

[4] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, Jul. 2006, pp. 384–385.

[5] K.-T. Seow, C. Ma, and M. Yokoo, "Multiagent planning as control synthesis," in *AAMAS'04, New York, USA*, July 2004.

[6] M. Sköldstam, K. Åkesson, and M. Fabian, "Supervisory control applied to automata extended with variables," Dept. Signals and Systems, Chalmers Univ. Tech., Göteborg, Sweden, Tech. Rep. R003/2007, 2007.

[7] M. Fabian, "Control and communication systems—lecture notes," Dept. Signals and Systems, Chalmers Univ. Tech., Göteborg, Sweden, Tech. Rep., 2002.

[8] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. Prentice-Hall, 1985.

[9] *KorSys homepage*, KorSys, Techniche Universität München. [Online]. Available: http://www4.in.tum.de/proj/korsys/

[10] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, 1986.

[11] J. O. Moody and P. J. Antsaklis, *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer, 1998.

[12] K. Åkesson, H. Flordal, and M. Fabian, "Exploiting modularity for synthesis and verification of supervisors," in *Proc. 15th IFAC World Congress*, Barcelona, Spain, Jul. 2002.

[13] H. Flordal, M. Fabian, and K. Åkesson, "Heuristics for verification and synthesis of mutually nonblocking discrete event systems," Dept. Signals and Systems, Chalmers Univ. Tech., Göteborg, Sweden, Tech. Rep. R012/2004, 2004.

[14] H. Flordal and R. Malik, "Modular nonblocking verification using conflict equivalence," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, Jul. 2006, pp. 100–106.

[15] ——, "Supervision equivalence," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, Jul. 2006, pp. 155–160.

[16] A. Vahidi, M. Fabian, and B. Lennartson, "Efficient supervisory synthesis of large systems," *Control Eng. Practice*, vol. 14, no. 10, pp. 1157–1167, Oct. 2006.

[17] M. Byröd, B. Lennartson, A. Vahidi, and K. Åkesson, "Efficient reachability analysis on modular discrete-event systems using binary decision diagrams," in *Proc. 8th Int. Workshop on Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, Jul. 2006, pp. 288–293.

[18] H. Flordal, M. Fabian, K. Åkesson, and D. Spensieri, "Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells," *Control Eng. Practice*, 2007, available online.

[19] K. Andersson, J. Richardsson, B. Lennartson, and M. Fabian, "Synthesis of hierarchical and distributed control functions for multi-product manufacturing cells," in *Proc. 2006 Conf. Automat. Sci. Eng.*, Shanghai, China, 2006.

[20] O. Ljungkrantz, K. Åkesson, J. Richardsson, and K. Andersson, "Implementing a control system framework for automatic generation of manufacturing cell controllers," in *Proc. Int. Conf. Robot. Automat., ICRA '07*, Rome, Italy, 2007.