

# Design and Synthesis from Components

Edited by

Jakob Rehof<sup>1</sup> and Moshe Y. Vardi<sup>2</sup>

1 TU Dortmund, DE, [jakob.rehof@cs.tu-dortmund.de](mailto:jakob.rehof@cs.tu-dortmund.de)

2 Rice University, US, [vardi@cs.rice.edu](mailto:vardi@cs.rice.edu)

---

## Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 14232 “Design and Synthesis from Components” which took place from June 1st to June 6th, 2014. The seminar aimed at bringing together researchers from the component-oriented design community, researchers working on interface theories, and researchers working in synthesis, in order to explore the use of component- and interface design in program synthesis. The seminar program consisted of 6 tutorial talks (1 hour) and 16 contributed talks (30 mins) as well as joint discussion sessions. This report documents the abstracts of the talks as well as summaries of discussion sessions.

**Seminar** June 1–6, 201401 – <http://www.dagstuhl.de/14232>

**1998 ACM Subject Classification** I.2.2 Automatic Programming – Program synthesis, D.2.2

Design Tools and Techniques, F.3.1 Specifying and Verifying and Reasoning about Programs

**Keywords and phrases** Component design, Component-based synthesis

**Digital Object Identifier** 10.4230/DagRep.4.6.29

**Edited in cooperation with** Dror Fried

## 1 Executive Summary

*Jakob Rehof*

*Moshe Y. Vardi*

**License**  Creative Commons BY 3.0 Unported license  
© Jakob Rehof and Moshe Y. Vardi

The purpose of the seminar was bringing together researchers from the component-oriented design community, researchers working on interface theories, and researchers working in synthesis, in order to explore the use of component- and interface design in program synthesis.

The seminar proposal was motivated by a recently developing trend in component-based synthesis, which is seen both as creating a need and providing the potential for a cross-community effort. Traditionally, synthesis has been pursued in two distinct and somewhat independent technical approaches. In one approach, synthesis is characterized by temporal logic and automata theoretic methods, whereas in the other synthesis is characterized by deductive methods in program logics and in type theory considered under the Curry-Howard isomorphism. Recent work in component-oriented design has spurred the idea of *component-based synthesis*, where systems are synthesized relative to a given collection (library, repository) of components, within both technical approaches. Recent results in both communities show that this development allows the two communities to communicate more intensely on the common ground of component-orientation to their mutual benefit. The trend opens the door to a new attack on the great challenges of synthesis (including computational complexity and complexity of specification) by exploiting component design.



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 3.0 Unported license

Design and Synthesis from Components, *Dagstuhl Reports*, Vol. 4, Issue 6, pp. 29–47

Editors: Jakob Rehof and Moshe Y. Vardi



Dagstuhl Reports

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The seminar program consisted of 6 tutorial talks (1 hour) and 16 contributed talks (30 mins) as well as joint discussion sessions. Two slots for joint discussions were pre-planned for each day but were used flexibly and dynamically, depending on the development of discussions and reactions to the talks. It was felt that the mixture of tutorials, talks and joint discussion slots turned out to be an altogether very good instrument for making intensive exchanges among all seminar participants possible. It seems to be the general impression that the seminar was very succesful in meeting the challenge of bringing together researchers from quite a diverse range of technical fields, spanning from software engineering to mathematical logic. The seminar was succesful in generating several concrete cross-community collaboration projects which would not have been likely to have come into existence by way of traditional conferences.

Joint discussions were summarized by Dror Fried (Rice University) who is gratefully acknowledged for undertaking the role of “seminar collector”.

## 2 Table of Contents

### Executive Summary

<i>Jakob Rehof and Moshe Y. Vardi</i> . . . . .	29
---	----

### Overview of Talks

Coordinated Composition of Components <i>Farhad Arbab</i> . . . . .	33
Synthesis for Communication-centred Programming <i>Mariangiola Dezani</i> . . . . .	33
ArchiType: Automatic Synthesis of Component & Connector-Software Architectures with Bounded Combinatory Logic <i>Boris Duedder</i> . . . . .	34
Petri Games: Synthesis of Distributed Systems with Causal Memory <i>Bernd Finkbeiner</i> . . . . .	34
Towards Improving Extensibility and Reuse of Modules Within a Product Line <i>George T. Heineman</i> . . . . .	35
On coercion synthesis for regular expressions <i>Fritz Henglein</i> . . . . .	35
Application-layer Connector Synthesis <i>Paola Inverardi</i> . . . . .	36
Towards Understanding Superlinear Speedup by Distillation <i>Neil D. Jones</i> . . . . .	36
Synthesis of Reactive Systems Components with Data <i>Bengt Jonsson</i> . . . . .	37
Application of Combinatory Logic Synthesizer in Robotics <i>Moritz Martens</i> . . . . .	37
Towards Synthesis of Uniform Strategy against Temporal Epistemic Logic <i>Hongyang Qu</i> . . . . .	38
Beyond Two-player Zero-sum Games: Motivations and Highlights of Recent Results <i>Jean-François Raskin</i> . . . . .	38
Combinatory Logic Synthesis <i>Jakob Rehof</i> . . . . .	39
In search for a strategy to search for a strategy <i>Sven Schewe</i> . . . . .	40
Workflow Synthesis – Concepts and Experience <i>Bernhard Steffen</i> . . . . .	40
What are “good” strategies in infinite games? <i>Wolfgang Thomas</i> . . . . .	41
Automated Synthesis of Service Choreographies <i>Massimo Tivoli</i> . . . . .	41
Component-Based System Design with Interfaces <i>Stavros Tripakis</i> . . . . .	42

Inhabitation problems	
<i>Paweł Urzyczyn</i> . . . . .	43
Compositional Temporal Synthesis	
<i>Moshe Y. Vardi</i> . . . . .	43
Compositional Controller Synthesis for Stochastic Games	
<i>Clemens Wiltsche</i> . . . . .	43
Programming with Millions of Examples	
<i>Eran Yahav</i> . . . . .	44
A combinatorial view of module composition for OO programming languages	
<i>Ugo de'Liguoro</i> . . . . .	44
<b>Joint Discussions</b>	
Component Orientation and Complexity (Tuesday 6/3/2014) . . . . .	45
Challenges (Wednesday 6/4/2014) . . . . .	45
Benchmarks (Thursday 6/5/2014) . . . . .	46
Conclusion (Thursday 6/5/2014) . . . . .	46
<b>Participants</b> . . . . .	47

### 3 Overview of Talks

#### 3.1 Coordinated Composition of Components

*Farhad Arbab (CWI – Amsterdam, NL)*

License © Creative Commons BY 3.0 Unported license  
© Farhad Arbab

Modeling components as units of behavior offers a rich framework where composition operators can coordinate the behavior of such constituents into the behavior of arbitrarily more complex systems. Our work on Reo, its semantics, and tools serves as a concrete instance of such a framework. The emphasis in Reo is on the externally observable behavior of components and their coordinated composition into more complex concurrent systems. This emphasis highlights the eminent role of protocols in concurrent systems of components and services, and makes concurrency protocols the central focus in Reo.

At its core, Reo proffers an interaction-based model of concurrency where more complex protocols result from composition of simpler, and eventually primitive, protocols. In Reo, combining a small set of user-defined synchronous and asynchronous primitives, in a manner that resembles construction of electronic circuits from gates and elements, yields arbitrarily complex concurrency protocols. Semantics of Reo preserves synchrony and exclusion through composition. This form of compositionality makes specification of protocols in Reo simpler than in conventional models and languages, which offer low-level synchronization constructs (e.g., locks, semaphores, monitors, synchronous methods). Moreover, the high-level constructs and abstractions in Reo also leave more room for compilers to perform novel optimizations in mapping protocol specifications to lower-level instructions that implement them. In on-going work we currently develop code generators that produce executables whose performance and scalability on multi-core platforms compare favorably with hand-crafted, hand-optimized code.

#### 3.2 Synthesis for Communication-centred Programming

*Mariangiola Dezani (University of Turin, IT)*

License © Creative Commons BY 3.0 Unported license  
© Mariangiola Dezani

**Joint work of** Coppo, Mario; Dezani, Mariangiola; Venneri, Betti

**Main reference** M. Coppo, M. Dezani-Ciancaglini, B. Venneri, “Self-Adaptive Monitors for Multiparty Sessions,” in Proc. of the 22nd Euromicro Int’l Conf. on Parallel, Distributed, and Network-Based Processing (PDP’14), pp. 688–696, IEEE, 2014; pre-print available from author’s webpage.

**URL** <http://dx.doi.org/10.1109/PDP.2014.18>

**URL** <http://www.di.unito.it/~dezani/papers/cdv14.pdf>

The increasing number of heterogeneous devices interacting in networks claims for a new programming style, usually called communication-centered programming. In this scenario possible participants to choreographies expose their interfaces, describing the communications they offer. Interaction protocols can be synthesised through a phase of negotiation between participants, in which different pieces of code can be composed in order to get the desired behaviour. At run time some unexpected event can make the current choreography no longer executable. In this case the participants should be able to adapt themselves in order to successfully continue the interaction. In this adaptation both new interfaces and new codes of participants could need to be synthesised.

### 3.3 ArchiType: Automatic Synthesis of Component & Connector-Software Architectures with Bounded Combinatory Logic

*Boris Duedder (TU Dortmund, DE)*

License  Creative Commons BY 3.0 Unported license  
© Boris Duedder

Joint work of Duedder, Boris; Moritz, Martens; Rehof, Jakob

Combinatory logic synthesis is a new type-based approach towards automatic synthesis of software from components in a repository. In this talk we demonstrate how the type-based approach can naturally be used to exploit taxonomic conceptual structures in software architectures and component repositories to enable automatic composition and configuration of components, and also code generation, by associating taxonomic concepts to architectural building blocks such as, in particular, software connectors. Components of a repository are exposed for synthesis as typed combinators, where intersection types are used to represent concepts that specify intended usage and functionality of a component. An algorithm for solving the type inhabitation problem in combinatory logic – does there exist a composition of combinators with a given type? – is then used to automate the retrieval, composition, and configuration of suitable building blocks with respect to a goal specification. Since type inhabitation has high computational complexity, heuristic optimizations for the inhabitation algorithm are essential for making the approach practical. We discuss particularly important (theoretical and pragmatic) optimization strategies and evaluate them by experiments. Furthermore, we apply this synthesis approach to define a method for software connector synthesis for realistic software architectures based on a type theoretic model. We conduct experiments with a rapid prototyping tool that employs this method on complex concrete ERP- and e-Commerce- systems and discuss some results.

### 3.4 Petri Games: Synthesis of Distributed Systems with Causal Memory

*Bernd Finkbeiner (Universität des Saarlandes, DE)*

License  Creative Commons BY 3.0 Unported license  
© Bernd Finkbeiner


Joint work of Finkbeiner, Bernd; Olderog; Ernst-Rüdiger

We introduce Petri games as a new foundation for the synthesis of distributed systems. The players of a Petri game consist of the system processes and the external environment, all represented as tokens on a Petri net. The players memorize their causal history and communicate it to each other during each synchronization.

Petri games lead to new decidability results and algorithms for the synthesis of distributed systems. Unlike the classic approaches, which are based on a fixed ordering of the relative informedness of the processes, we can synthesize systems with dynamically changing information flow, as in client-server protocols, where the information source moves back and forth between client and server, or in token ring protocols, where the identity of the sender changes as the token moves.

### 3.5 Towards Improving Extensibility and Reuse of Modules Within a Product Line

*George T. Heineman (Worcester Polytechnic Institute, US)*

License  Creative Commons BY 3.0 Unported license  
© George T. Heineman

The principle of modularity is the basis for both object-oriented (OOD) and component-based design (CBD) but there is a tension between extensibility and reuse. OOD often leads to rich frame-works that enable new classes to be designed as extensions to existing classes but there is little reuse of individual classes out-side of the framework. CBD often leads to third party assembly through well-designed interfaces but it is often impossible to extend components. The Model View Controller (MVC) paradigm bridges these two domains because it can be described as both a Design Pattern (in the OOD realm) and an Architectural Pattern (in the CBD realm). While MVC supports rich extensibility in both models and views, it is striking that it seems to naturally lead to controllers that cannot be reused or extended. We combine the MVC paradigm with feature-oriented programming (FOP) to bring reusability back to controllers. We demonstrate the effectiveness of our approach using a product-line example of a solitaire game engine.

### 3.6 On coercion synthesis for regular expressions

*Fritz Henglein (University of Copenhagen, DK)*

License  Creative Commons BY 3.0 Unported license  
© Fritz Henglein

Regular expressions (REs) are usually interpreted as languages. This is, however, an inadequate theoretical basis for programming applications where parsing, extracting and transforming data, not just membership testing, is required.

REs can be interpreted more intensionally as types, each representing a set of parse trees, such that establishing language containment corresponds to finding some coercion: a function transforming parse trees according to one RE to parse trees in the other RE without changing the underlying string. Coercions can be found by constructive interpretation of axiomatizations of regular expression containment.

This puts the particular axiomatization at center stage: We are not only interested in determining whether or not some coercion exists (whether or not a particular RE containment holds), but actually synthesizing its actual code; furthermore, since the synthesized coercions are actually executed, finding (in particular: not ruling out in the axiomatization) coercions with good computational properties – e.g. streaming execution on a bit-serialized representation of syntax trees and always running in worst-case linear time – is important.

The basic questions are then: How can one efficiently synthesize coercions, which themselves need to be efficient, by proof search in an axiomatization of regular expression containment? More basically, what is a good axiomatization for doing this? And why is regular expression containment an interesting synthesis case study for synthesis? We present preliminary results and some thoughts on these questions.

### 3.7 Application-layer Connector Synthesis

*Paola Inverardi (University of L'Aquila, IT)*

**License**  Creative Commons BY 3.0 Unported license  
© Paola Inverardi

**Joint work of** Inverardi, Paola; Massimo, Tivoli; Romina, Spalazzese; Marco, Autili

**Main reference** P. Inverardi, M. Tivoli, “Automatic synthesis of modular connectors via composition of protocol mediation patterns,” in Proc. of the 2013 Int'l Conf. on Software Engineering (ICSE'13), pp. 3–12, IEEE/ACM, 2013.

**URL** <http://dl.acm.org/citation.cfm?id=2486790>

The heterogeneity characterizing the systems populating the Ubiquitous Computing environment prevents their seamless interoperability. Heterogeneous protocols may be willing to cooperate in order to reach some common goal even though they meet dynamically and do not have a priori knowledge of each other. Despite numerous efforts have been done in the literature, the automated and run-time interoperability is still an open challenge for such environment. We consider interoperability as the ability for two Networked Systems (NSs) to communicate and correctly coordinate to achieve their goal(s). In this tutorial, I report the main outcomes of our past and recent research on automatically achieving protocol interoperability via connector synthesis. We consider application-layer connectors by referring to two conceptually distinct notions of connector: coordinator and mediator. The former is used when the NSs to be connected are already able to communicate but they need to be specifically coordinated in order to reach their goal(s). The latter goes a step forward representing a solution for both achieving correct coordination and enabling communication between highly heterogeneous NSs. In the past, most of the works in the literature described efforts to the automatic synthesis of coordinators while, in recent years the focus moved also to the automatic synthesis of mediators. Within the Connect project, by considering our past experience on automatic coordinator synthesis as a baseline, we propose a formal theory of mediators and a related method for automatically eliciting a way for the protocols to interoperate. The solution we propose is the automated synthesis of emerging mediating connectors (i.e., mediators for short).

### 3.8 Towards Understanding Superlinear Speedup by Distillation

*Neil D. Jones (University of Copenhagen, DK)*

**License**  Creative Commons BY 3.0 Unported license  
© Neil D. Jones

**Joint work of** Jones, Neil D.; Hamilton, Geoff W.

Distillation is a transformation method that can yield superlinear program speedups – a feat beyond earlier fully automatic program transformations such as partial evaluation or supercompilation. Bisimulation is a key to correctness of distillation, i.e., that it preserves semantics.

Relation to Dagstuhl 14232: an optimiser synthesises an efficient program from a less efficient one. A first question: In what sense can equivalent programs with asymptotically different runtimes be bisimilar?

The talk describes current work on such questions, partly theoretical and partly computer experiments, on some “old chestnut” programs well-known from program transformation literature (naive reverse, factorial sum, Fibonacci, and palindrome detection).



Using complexity-theoretic tools, we see that a sizable class of first-order exponential-time programs can be converted into second-order polynomial-time equivalents. The effect is to trade time for space, in effect replacing CONS or a Turing machine tape by first-order functions as arguments in a CONS-free program. Finally, a conjecture: that distillation can automatically realise many such superlinear speedups.

### 3.9 Synthesis of Reactive Systems Components with Data

*Bengt Jonsson (Uppsala University, SE)*

License © Creative Commons BY 3.0 Unported license  
© Bengt Jonsson

We consider automated synthesis of reactive components. Synthesis of reactive components is emerging to solve many tasks in software development, in embedded systems, for device drivers, for protocol converters, in service composition, etc. For the synthesis of finite-state components from finite-state specifications, there is currently an elaborate body of theory, which typically performs synthesis by constructing a winning strategy in a two-player game. In this presentation, we consider to extend synthesis of reactive components to take into account also data from potentially unbounded domains. Data from potentially unbounded domains are a natural ingredient in the specification and synthesis of many classes of systems. For example, a protocol mediator must transfer messages and data items correctly between the mediated components, a device driver must deliver the right data in the expected order.

We present techniques for synthesizing reactive components, and show how they can be used to automatically synthesize “missing” glue components in systems of communicating components: Given specification of a collection components, and a specification of the overall system, we can synthesize a most general glue component to satisfy the specification. We also show how the problem of finding a suitable system specification can be automated to a large degree. The techniques are natural when involved component specifications are deterministic. For the nondeterministic case, the synthesis problem is undecidable, and we consider how to restrict the solution space to obtain a decidable synthesis problem.

### 3.10 Application of Combinatory Logic Synthesizer in Robotics

*Moritz Martens (TU Dortmund, DE)*

License © Creative Commons BY 3.0 Unported license  
© Moritz Martens  
Joint work of Martens, Moritz; Döder, Boris; Rehof, Jakob

We present an application of Combinatory Logic Synthesizer, a type based synthesis tool, to the synthesis of workflows. The approach is illustrated by means of synthesis of workflows to control LegoNXT robots.

### 3.11 Towards Synthesis of Uniform Strategy against Temporal Epistemic Logic

*Hongyang Qu (University of Sheffield, GB)*

**License** © Creative Commons BY 3.0 Unported license  
© Hongyang Qu

**Joint work of** Busard, Simon; Pecheur, Charles; Qu, Hongyang; Raimondi, Franco

**Main reference** S. Busard, C. Pecheur, H. Qu, F. Raimondi, “Reasoning about Strategies under Partial Observability and Fairness Constraints,” in Proc. of the 1st Int’l Workshop on Strategic Reasoning (SR’13), EPTCS, Vol. 112, pp. 71–79, 2013.

**URL** <http://dx.doi.org/10.4204/EPTCS.112.12>

In this talk, I will first give an introduction of Interpreted Systems (IS) and epistemic logic. The former provides a formal semantics for modelling multi-agent systems, and the latter specifies knowledge based properties, which are often combined with temporal logics, such as CTL, LTL and ATL. The second part of the talk will focus on difficulties in searching for a uniform strategy for temporal epistemic logic formulas. I will present two semantics for uniform strategy, each of which requires different solutions.

#### References

- 1 Alessio Lomuscio, Franco Raimondi, Model checking knowledge, strategies, and games in multi-agent systems, in: 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), ACM, 2006, pp. 161–168.

### 3.12 Beyond Two-player Zero-sum Games: Motivations and Highlights of Recent Results

*Jean-François Raskin (Université Libre de Bruxelles, BE)*

**License** © Creative Commons BY 3.0 Unported license  
© Jean-François Raskin

Two-player zero-sum games played on graphs is the classical setting for studying the reactive synthesis problem. In this talk, I will report on recent work directions that consider richer settings. To illustrate those new research directions, I will present informally with the help of examples the results contained in three recent publications whose abstract are reproduced below:

1. Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, Jean-François Raskin. *Doomsday Equilibria for Omega-Regular Games*. VMCAI 2014, LNCS, Vol. 8318, pp. 78–97. [http://dx.doi.org/10.1007/978-3-642-54013-4\\_5](http://dx.doi.org/10.1007/978-3-642-54013-4_5)

Two-player games on graphs provide the theoretical framework for many important problems such as reactive synthesis. While the traditional study of two-player zero-sum games has been extended to multi-player games with several notions of equilibria, they are decidable only for perfect-information games, whereas several applications require imperfect-information games. In this paper we propose a new notion of equilibria, called doomsday equilibria, which is a strategy profile such that all players satisfy their own objective, and if any coalition of players deviates and violates even one of the players objective, then the objective of every player is violated. We present algorithms and complexity results for deciding the existence of doomsday equilibria for various classes

of omega-regular objectives, both for imperfect-information games, and for perfect-information games. We provide optimal complexity bounds for imperfect-information games, and in most cases for perfect-information games.

2. Veronique Bruyere, Emmanuel Filiot, Mickael Randour, Jean-François Raskin. *Meet Your Expectations With Guarantees: Beyond Worst-Case Synthesis in Quantitative Games*. STACS 2014, LIPIcs, Vol. 25, pp. 199–213. <http://dx.doi.org/10.4230/LIPIcs.STACS.2014.199>

Classical analysis of two-player quantitative games involves an adversary (modeling the environment of the system) which is purely antagonistic and asks for strict guarantees while Markov decision processes model systems facing a purely randomized environment: the aim is then to optimize the expected payoff, with no guarantee on individual outcomes. We introduce the beyond worst-case synthesis problem, which is to construct strategies that guarantee some quantitative requirement in the worst-case while providing an higher expected value against a particular stochastic model of the environment given as input. We consider both the mean-payoff value problem and the shortest path problem. In both cases, we show how to decide the existence of finite-memory strategies satisfying the problem and how to synthesize one if one exists. We establish algorithms and we study complexity bounds and memory requirements.

3. Romain Brenguier, Jean-François Raskin, Mathieu Sassolas. *The Complexity of Admissibility in Omega-Regular Games*. CoRR abs/1304.1682(2013) – to appear in LICS'14. <http://arxiv.org/abs/1304.1682v3>

Iterated admissibility is a well-known and important concept in classical game theory, e.g. to determine rational behaviors in multi-player matrix games. As recently shown by Berwanger, this concept can be soundly extended to infinite games played on graphs with omega-regular objectives. In this paper, we study the algorithmic properties of this concept for such games. We settle the exact complexity of natural decision problems on the set of strategies that survive iterated elimination of dominated strategies. As a byproduct of our construction, we obtain automata which recognize all the possible outcomes of such strategies.

### 3.13 Combinatory Logic Synthesis

*Jakob Rehof (TU Dortmund, DE)*

License © Creative Commons BY 3.0 Unported license  
© Jakob Rehof

Joint work of Düdder, Boris; Martens, Moritz; Urzyczyn, Pawel

Combinatory logic synthesis has been proposed recently as a type-theoretic research programme in automatic synthesis of compositions from collections of components. Composition synthesis is based on the idea that the inhabitation (provability) relation in combinatory logic can be used as a logical foundation for synthesizing expressions satisfying a goal type (specification) relative to a given collection of components exposed as a combinatory type environment.

It is shown that, under the semantics of relativized inhabitation, already simple types are a Turing-complete logic programming language for computing (synthesizing) compositions, where collections of combinatory types are programs and types are rules in such programs. In order to enhance the ability to express semantic specifications we introduce intersection types

into composition synthesis, and we survey recent results on expressive power, algorithmics, and complexity. It is shown that modal types lead to a natural foundation for introducing meta-programming combinators, resulting in a highly flexible framework for composition synthesis. Based on a prototype implementation of the framework (CL)S, Combinatory Logic Synthesizer, we illustrate with practical examples as time permits.

### 3.14 In search for a strategy to search for a strategy

*Sven Schewe (University of Liverpool, GB)*

**License** © Creative Commons BY 3.0 Unported license  
© Sven Schewe

**Joint work of** John Fearnley, Doron Peled, Schewe, Sven

**Main reference** J. Fearnley, D. Peled, S. Schewe, “Synthesis of Succinct Systems,” in Proc. of the 10th Int’l Symp. on Automated Technology for Verification and Analysis (ATVA’12), LNCS, Vol. 7561, pp. 208–222, Springer, 2012.

**URL** [http://dx.doi.org/10.1007/978-3-642-33386-6\\_18](http://dx.doi.org/10.1007/978-3-642-33386-6_18)

In synthesis, we seek a strategy to control a system to satisfy its objectives. But how do we search? Much research has been done to establish that the problem is hard, and just how hard it is. While this has been used to argue against synthesis, synthesis algorithms exist, but currently those with two arms and two legs have the upper hand. I would like to wonder with you if arms and legs are really necessary for synthesis, or if we can search for a search strategy.

#### References

- 1 B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. of IEEE LICS 2005*, pages 321–330. IEEE Computer Society Press.
- 2 B. Finkbeiner and S. Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5-6):519–539, 2013.
- 3 J. Fearnley, D. Peled, and S. Schewe. Synthesis of Succinct Systems. *Proc. of ATVA 2012*, pages 42–56.

### 3.15 Workflow Synthesis – Concepts and Experience

*Bernhard Steffen (TU Dortmund, DE)*

**License** © Creative Commons BY 3.0 Unported license  
© Bernhard Steffen

**Joint work of** Steffen, Bernhard; Tiziana Margaria, Johannes Neubauer, Stefan Naujokat

**Main reference** T. Margaria, D. Meyer, C. Kubczak, M. Isberner, B. Steffen, “Synthesizing Semantic Web Service Compositions with jMosel and Golog,” in 8th Int’l Semantic Web Conf. (ISWC’09), LNCS, Vol. 5823, pp. 392–407, Springer, 2009.

**URL** [http://dx.doi.org/10.1007/978-3-642-04930-9\\_25](http://dx.doi.org/10.1007/978-3-642-04930-9_25)

The marriage of component-based design and software synthesis is promising, in particular in special scenarios like workflow-design where where components can often nicely be regarded as abstract functions. In this very much service-oriented setting linear time synthesis of chains of interactive activities and automatic processing steps leads to a completely new way of organization and management, which allows one to much better adapt to changing situations simply via appropriate ‘replanning’. Based on an adequate ontology-based domain modelling this can be done in a very situation and process-aware fashion which may in particular also take legal procedural constraints into account.

Another interesting application domain are scientific workflows, where chains of analysis steps need to be arranged in a tailored fashion.

In both cases, linear time synthesis leads to a 'development' style characterized by constraint-driven search of adequate solutions: rather than programming, the user needs to define abstract requirements, and select the best fitting proposed solutions, perhaps after some steps where he refined his requirements to better tailor the search. Combined with an easy pattern-based interface for entering constraints this development style has proven to be very effective for scientist without programming knowledge and in scenarios where the available libraries processing components are continuously evolving.

### 3.16 What are “good” strategies in infinite games?

*Wolfgang Thomas (RWTH Aachen, DE)*

**License** © Creative Commons BY 3.0 Unported license  
© Wolfgang Thomas

Research on infinite (two-person) games shifted over the past decades from the study of existence problems (in set theory: determinacy) via algorithmic questions (in computer science: construction of automata realizing winning strategies) to a more refined perspective: How to construct strategies that are “good” in terms of (1) efficiency, or (2) appropriate format. We first review results and open problems on item (1), regarding minimization of memory size of controllers, and regarding optimization of behavior (explained in the example of reducing waiting times in “request-response games”). Then we discuss item (2): representations of strategies that are alternatives to transition systems (automata with output), namely strategy machines (which are based on Turing machines, Gelderie 2012-2014), Boolean programs (Madhusudan 2011, Brütsch 2014), and logic formulas (Rabinovich, Ths. 2007). This research can be understood as approaches to the problem stated already at the end of the classic paper of Büchi-Landweber (1969): to obtain a comprehensive view of the space of all winning strategies of a given game.

### 3.17 Automated Synthesis of Service Choreographies

*Massimo Tivoli (University of L'Aquila, IT)*

**License** © Creative Commons BY 3.0 Unported license  
© Massimo Tivoli

**Joint work of** Tivoli, Massimo; Autili, Marco; Inverardi, Paola

**Main reference** M. Autili, D. Di Ruscio, A. Di Salle, P. Inverardi, M. Tivoli, “A Model-Based Synthesis Process for Choreography Realizability Enforcement,” in Proc. of the 16th Int'l Conf. on Fundamental Approaches to Software Engineering (FASE'13), LNCS, Vol. 7793, pp. 37–52, Springer, 2013.

**URL** [http://dx.doi.org/10.1007/978-3-642-37057-1\\_4](http://dx.doi.org/10.1007/978-3-642-37057-1_4)

Modern service-oriented systems are often built by reusing, and composing together, existing services distributed over the Internet. Service choreography is a possible form of service composition aiming at specifying the interactions between the participant services from a global perspective. In this talk, I overview a method for the distributed enforcement of service choreographies via automated synthesis of coordination delegates. When interposed among the participant services, coordination delegates intercept the service interaction and mediate it in order to realize the specified choreography. This method is implemented as

part of a model-based tool chain released to support the development of choreography-based systems within the EU CHOReOS project.

### 3.18 Component-Based System Design with Interfaces

*Stavros Tripakis (University of California – Berkeley, US)*

License  Creative Commons BY 3.0 Unported license  
© Stavros Tripakis

Joint work of Tripakis, Stavros; Lubliner, Roberto

Main reference S. Tripakis, R. Lubliner, “Modular Code Generation from Synchronous Models: Abstraction and Compositionality,” pre-print.

URL <http://www.dagstuhl.de/mat/Files/14/14232/14232.TripakisStavros.Paper.pdf>


Model-based design (MBD) is a design methodology that relies on three key elements: modeling (how to capture the system that we want), analysis (how to be sure that this is the system that we want before actually building it), and synthesis (how to build a “low-level” implementation of the system from a “high-level” model/specification). This talk discusses some of our work on MBD with a focus on compositionality. Compositional methods, which allow to assemble smaller components into larger systems both efficiently and correctly, are not simply a desirable feature in system design: they are a must for building large and complex systems. A key ingredient for compositionality is that of an “interface”. An interface abstracts a component, exposing relevant information while hiding internal details. We give an overview of the many uses of interfaces in MBD, from modular code generation from hierarchical models, to incremental design with interface theories, to Ptolemy simulation and FMI co-simulation, to multiview modeling.

#### References

- 1 R. Lubliner, and S. Tripakis. *Modularity vs. Reusability: Code Generation from Synchronous Block Diagrams*. Design, Automation, and Test in Europe (DATE’08).
- 2 R. Lubliner, and S. Tripakis. *Modular Code Generation from Triggered and Timed Block Diagrams*. 14th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’08).
- 3 R. Lubliner, C. Szegedy, and S. Tripakis. *Modular Code Generation from Synchronous Block Diagrams – Modularity vs. Code Size*. 36th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’09).
- 4 S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee. *A Theory of Synchronous Relational Interfaces*. ACM Transactions on Programming Languages and Systems (TOPLAS), 33, 4, 2011.
- 5 S. Tripakis, C. Stergiou, C. Shaver, and E. A. Lee. *A modular formal semantics for Ptolemy*. Mathematical Structures in Computer Science, 23, 2013.
- 6 S. Tripakis, C. Stergiou, M. Broy, and E. A. Lee. *Error-Completion in Interface Theories*. International SPIN Symposium on Model Checking of Software – SPIN 2013.
- 7 D. Broman, C. Brooks, L. Greenberg, E. A. Lee, S. Tripakis, M. Wetter, and M. Masin. *Determinate Composition of FMUs for Co-Simulation*. Proceedings of the 13th ACM & IEEE International Conference on Embedded Software (EMSOFT’13).
- 8 J. Reineke, and S. Tripakis. *Basic Problems in Multi-View Modeling*. Tools and Algorithms for the Construction and Analysis of Systems – TACAS 2014.

### 3.19 Inhabitation problems

*Paweł Urzyczyn (University of Warsaw, PL)*

License  Creative Commons BY 3.0 Unported license  
© Paweł Urzyczyn

This talk is about a specific version of synthesis: the synthesis of a proof for a given formula. Under the Curry-Howard Isomorphism, formulas correspond to types and their proofs correspond to terms (programs) of appropriate types. Therefore the provability problem is usually equivalent to an inhabitation problem (is a given type non-empty?) in a corresponding lambda-calculus.

The talk covers the following issues:

- introduction to the Curry-Howard Isomorphism;
- Ben-Yelles inhabitation algorithm for simple types and its complexity;
- extensions to arbitrary propositional connectives; and first order quantifiers;
- the automata-theoretic and the game-theoretic paradigm of proof search.

### 3.20 Compositional Temporal Synthesis

*Moshe Y. Vardi (Rice University, US)*

License  Creative Commons BY 3.0 Unported license  
© Moshe Y. Vardi


Synthesis is the automated construction of a system from its specification. In standard temporal-synthesis algorithms, it is assumed the system is constructed from scratch. This, of course, rarely happens in real life. In real life, almost every non-trivial system, either in hardware or in software, relies heavily on using libraries of reusable components. Furthermore, other contexts, such as web-service orchestration and choreography, can also be modeled as synthesis of a system from a library of components.

In this talk we describe and study the problem of compositional temporal synthesis, in which we synthesize systems from libraries of reusable components. We define two notions of composition: data-flow composition, which we show is undecidable, and control-flow composition, which we show is decidable. We then explore a variation of control-flow compositional synthesis, in which we construct reliable systems from libraries of unreliable components.

Joint work with Yoad Lustig and Sumit Nain.

### 3.21 Compositional Controller Synthesis for Stochastic Games

*Clemens Wiltsche (University of Oxford, GB)*

License  Creative Commons BY 3.0 Unported license  
© Clemens Wiltsche

Joint work of Basset, Nicolas; Kwiatkowska, Marta; Wiltsche, Clemens


Design of autonomous systems is facilitated by automatic synthesis of correct-by-construction controllers from formal models and specifications. We focus on stochastic games, which can model the interaction with an adverse environment, as well as probabilistic behaviour arising from uncertainties. We propose a synchronising parallel composition for stochastic games that



enables a compositional approach to controller synthesis. We leverage rules for compositional assume-guarantee verification of probabilistic automata to synthesise controllers for games with multi-objective quantitative winning conditions. By composing winning strategies synthesised for the individual components, we can thus obtain a winning strategy for the composed game, achieving better scalability and efficiency at a cost of restricting the class of controllers.

### 3.22 Programming with Millions of Examples


*Eran Yahav (Technion – Haifa, IL)*

License  Creative Commons BY 3.0 Unported license  
© Eran Yahav

We present a framework for data-driven synthesis, aiming to leverage the collective programming knowledge captured in millions of open-source projects. Our framework analyzes code snippets and extracts partial temporal specifications. Technically, partial temporal specifications are represented as symbolic automata where transitions may be labeled by variables, and a variable can be substituted by a letter, a word, or a regular language. Using symbolic automata, we consolidate separate examples to create a database of snippets that can be used for semantic code-search and component synthesis. We have implemented our approach and applied it to analyze and consolidate millions of code snippets.

### 3.23 A combinatorial view of module composition for OO programming languages

*Ugo de Liguoro (University of Turin, IT)*

License  Creative Commons BY 3.0 Unported license  
© Ugo de Liguoro

**Joint work of** de Liguoro, Ugo; Tzu-Chun Chen

**Main reference** U. de Liguoro, T. Chen, “Semantic Types for Classes and Mixins,” pre-print.

**URL** <http://www.di.unito.it/~deligu/papers/UdLTC14.pdf>

Taking a lambda calculus with records as the basic model, we discuss the choice of a set of combinators representing various possibilities in composing software modules, designed as components to build class hierarchies (e.g. traits or mixins). We also consider a suitable extension of intersection type discipline to be thought of as a specification language for module properties, aiming at extending Rehof’s method of program synthesis by inhabitation to the case of class based programming languages.

#### References

- 1 Ugo de Liguoro. *Characterizing convergent terms in object calculi via intersection types* Proc. of TLCA’01, LNCS 2044, pp. 315–328, 2001.
- 2 Ugo de Liguoro, Tzu-Chun Chen. *Semantic Types for Classes and Mixins* Proc. of ITRS’14, EPTCS, to appear.



## 4 Joint Discussions

*Dror Fried*

License  Creative Commons BY 3.0 Unported license  
© Dror Fried

### 4.1 Component Orientation and Complexity (Tuesday 6/3/2014)

The major problem that is addressed in this discussion is in fact the core of this seminar: there are different teams, with different models of computation. There is a need to find a common ground of formalism in order to enable models comparison.

A short talk initiated by Prof. Farhad Arbab has presented an idea of the differences between direct and indirect methods of constructions, as well as the role of the components as facilitating the transition between these two models. Then, the components are combined by a protocol that expresses relationship such as: synchrony/asynchrony, exclusion, grouping, etc.

Other ideas that were brought up during this discussion:

- The users eventually care about the behaviour, not about the construction of the system. However, sometimes it is hard to tell what the user is really interested in.
- The cost, including the hidden cost of the components, plays a major role in the actual implementation.
- Eventually every software changes. We want to minimize these changes in a rapidly changing environment. How do we response to changes? Do we need to this response to be in the overall specification or in every component locally?
- How do components influence the computational complexity? In theory component are harder to analyse because they hide information and one has to take all possibles into account. However, practice might show otherwise. Perhaps we should consider only the interaction protocol in terms of complexity, and not be concerned with the insides of the components. For example, as we usually use an existing code as a component (from a library) rather than writing one from scratch, we shouldn't be concerned with the analysis of that specific piece of code.

### 4.2 Challenges (Wednesday 6/4/2014)

The challenges that we should seek, and which of these challenges we focus on, is the topic of this discussion. Some of these challenges are relevant to many fields in computer science, in which scientists are not appreciated as expanding humanity knowledge, but rather as deliverers of tools that work.

- Perhaps it is better to seek a “political challenge”. Something big that will draw a lot of attention. For example, the Automatic Theorem Prover. People will get fascinated, and we will get a lot of appreciation and attention. Most grand-challenges are engineering, not scientific. We should look for an engineering grand-challenge.
- However, such projects can easily go down the drain. Projects that aim too high are often not trusted, thus not approved. Perhaps we should instead be realistic. Realize that we don't have high esteem as mathematicians have, and we are only gray-collar workers who provide actual tools, instead of expanding the knowledge of humanity.
- Another approach is to show people what we have done so far, instead of showing what we plan to do. Society has kept us so far because we deliver goods, and so far we have met society's requests.
- We should do things gradually. Like SMT. It took 50 years for people to notice, and now it is big. At first, SMT was just obsession of a few people, but it advanced well,

and now it works. Perhaps this is an example that there is a strategic research as well. Specifically, we should start with small programs, take one step at a time. On the other hand, the criteria of success is dynamic, so it is hard to strategically aim in advance for a specific goal.

### 4.3 Benchmarks (Thursday 6/5/2014)

We have discussed the possibility of creating benchmarks that will serve as a common ground for to compare different methods. The main problem with this approach is that currently each has his own formalism in his own world. There is not one (or two) specific formats on which we can agree. This seminar is a first step, but we still need to find a way to communicate in the same language. However, one attempt that we can already try is to integrate works in Combinatory Logic with the SyGus format, and to participate in the SyGus competition. However, perhaps it will be beneficial to start with a real world problem, something concrete. Then just try to solve that problem by using various tools. We can get inspiration from the Theorem Proving community that does the same thing.

### 4.4 Conclusion (Thursday 6/5/2014)

In the last discussion we have expressed various ideas that were brought during the seminar:

- In comparison to 5–7 years ago, the Computer Science area uses more components. Therefore we are now more aware that these component problems exist. On the semantic level, it seems that there is a convergence of what these components do. Everybody has a clear notion what they mean by component: philosophically we agree. The differences are “only” technical.
- Still, technicality matters as component are abstract and one need something practical to work with: benchmarks, scenarios. For example – formalize what client-server architecture means. We should then ask questions such as: how does the logical for design aspects the complexity of the synthesis problem? How does experimental knowledge for software engineers affect the synthesis problem?
- From a previous experience: a low-level formalism of a declarative program may lead to a chaos (for example: by defining state as boolean logic). The reason is that the higher the formalism is, the more global mistakes there are. These global mistakes are sometimes easier to fix than local ones. Sometimes it’s easier to nail the big bugs than the small.
- We need to put thought on how to test our tools. We need to test the specification as well. However performing many tests should not be our main objective. What about the option of synthesizing tests with the problem? Is this something that can be considered?
- Let’s be realistic: Formal unified specification is hard. Languages are not user friendly. However, we should start think what people like to write their specification with (counter example: LTL). The industry developed their own languages- there are many, and each for a local use.
- Sometimes we mix architecture with structure. Architecture is far more than that. The attempt to formalize architecture will result in formalizing the structure, not more. It is also related to architecture patterns. You cannot formalize that. Then perhaps architecture pattern can be a part of the synthesis. Think of the architecture as an extra constraint for the synthesis. However, this is not a composition, but rather a construct of the architecture process. For example: we don’t necessarily have a client-server application but we need some of it as constraints.

## Participants

- Farhad Arbab  
CWI – Amsterdam, NL
- Christel Baier  
TU Dresden, DE
- Ugo de'Liguoro  
University of Turin, IT
- Mariangiola Dezani  
University of Turin, IT
- Laurent Doyen  
ENS – Cachan, FR
- Boris Döder  
TU Dortmund, DE
- Bernd Finkbeiner  
Universität des Saarlandes, DE
- Dror Fried  
Rice University, US
- George T. Heineman  
Worcester Polytechnic Inst., US
- Fritz Henglein  
University of Copenhagen, DK
- Paola Inverardi  
University of L'Aquila, IT
- Neil D. Jones  
University of Copenhagen, DK
- Bengt Jonsson  
Uppsala University, SE
- Axel Legay  
INRIA Bretagne Atlantique –  
Rennes, FR
- Moritz Martens  
TU Dortmund, DE
- Hongyang Qu  
University of Sheffield, GB
- Jean-François Raskin  
Université Libre de Bruxelles, BE
- Jakob Rehof  
TU Dortmund, DE
- Sven Schewe  
University of Liverpool, GB
- Joseph Sifakis  
VERIMAG – Gières, FR
- Bernhard Steffen  
TU Dortmund, DE
- Wolfgang Thomas  
RWTH Aachen, DE
- Massimo Tivoli  
University of L'Aquila, IT
- Stavros Tripakis  
University of California –  
Berkeley, US
- Paweł Urzyczyn  
University of Warsaw, PL
- Moshe Y. Vardi  
Rice University, US
- Clemens Wiltsche  
University of Oxford, GB
- Eran Yahav  
Technion – Haifa, IL

