Report from Dagstuhl Seminar 19071

Specification Formalisms for Modern Cyber-Physical Systems

Edited by

Jyotirmoy V. Deshmukh[1], Oded Maler[2], and Dejan Ničković[3]

1 **University of Southern California – Los Angeles, US,** jdeshmuk@usc.edu
2 **VERIMAG – Grenoble, FR,** oded.maler@univ-grenoble-alpes.fr
3 **AIT Austrian Institute of Technology, AT,** dejan.Ničković@ait.ac.at

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 19071 "Specification Formalisms for Modern Cyber-Physical Systems." Specifications play a major role in evaluating behaviors of modern cyber-physical systems (CPS). There is currently no specification language that allows joint description of safety, performance, security, privacy, and reliability aspects of CPS applications. The Dagstuhl seminar brought together researchers and practitioners from formal methods, control theory, machine learning and robotics to discuss the state-of-the-art and open challenges in specifying properties of modern CPS. Special attention was given to exploring the intersection of machine learning and formal specification languages, where formal specifications can serve as a bridge between the world of verification and the world of learning and data-mining.

Seminar February 10–15, 2019 – http://www.dagstuhl.de/19071

2012 ACM Subject Classification Theory of computation → Timed and hybrid models, Theory of computation → Modal and temporal logics, Theory of computation → Formalisms, Computer systems organization → Embedded and cyber-physical systems, Computer systems organization → Real-time system specification

Keywords and phrases Cyber-physical systems, formal specifications, runtime verification and control

Digital Object Identifier 10.4230/DagRep.9.2.48

1 Executive Summary

Jyotirmoy V. Deshmukh
Dejan Ničković

Modern Cyber-Physical Systems (CPS) represent the convergence of the fields of control theory, artificial intelligence, machine learning, and distributed communication/coordination. CPS applications range from small quad-rotor based aerial vehicles to commercial airplanes, from driverless autonomous vehicles to vehicle platoons, from nano-scale medical devices in closed-loop with a human to giga-scale industrial manufacturing systems. While several application domains can claim to be cyber-physical systems, a unique aspect of CPS is a strong focus on model-based development (MBD). The MBD paradigm allows analyzing the system virtually, examining its safety, performance, stability, security, privacy, or resilience. At a certain level of abstraction, a model of a CPS application can be roughly divided into

three parts: (1) the plant model representing an encapsulation of the physical components in the system, (2) the controller model representing the software used to regulate the plant, and (3) an environment model representing exogenous disturbances to the plant.

Given plant, controller and environment models of a system, in order to perform any of the aforementioned analyses, a crucial step is to articulate the goal of the analysis as a formal specification for the system. The analysis problem can then check whether the system implementation is a refinement of its specification. However, the state-of-the-art in industrial settings is that formal specifications are rarely found. Specifications exist in the form of mental models of correctness formed by engineers through their design insights and experience, or visual depictions in the form of simulation plots, and occasionally as legacy scripts and monitors. None of these are formal, machine-checkable unambiguous specifications. In the industry, engineers often use the term requirements instead of specifications. Typical industrial requirements do not arise from principled software engineering approaches to develop CPS software, but rather are summaries of discussions between developers and their customers. While the state-of-the-art for requirements/specifications in industrial settings is far from ideal, in academic settings, there is a problem of having a wide choice between a number of specification formalisms, primarily being developed by the formal methods community. On the other hand, application-specific academic domains such as robotics, biological systems, and medical devices may not always articulate formal system specifications.

The overarching goal of the seminar was thus to address the following question: *Is there a universal specification formalism that can be used as a standard language for a variety of modern cyber-physical systems?* To address this question, this seminar was divided into three broad thrusts:

- State-of-the-art in general specification formalisms,
- Domain-specific needs and domain-specific specification formalisms,
- Expressivity, Monitoring Algorithms and Analysis concerns for a specification language.

Outcome of the seminar

The seminar had a total of 37 participants with a mix of research communities including experts (both theoreticians and practitioners) in formal methods, runtime monitoring, machine learning, control theorey, industrial IoT, and biological systems. The seminar focused on the cross-domain challenges in the development of a universal specification formalism that can accommodate for various CPS applications.

The seminar provided an excellent overview of requirements from various application domains that paved the road for identifying common features in a cross-domain specification language. As another outcome of the seminar, we defined as a community the following next steps:

1. Identification of various benchmark problems for monitoring specifications at runtime, and learning specifications from data.
2. Standardizing syntax for expressing time-series data, such as comma separated values (CSV) with a well-defined header file.
3. Creating a public repository containing traces, specifications, models, and pattern libraries.
4. Coordination with RVComp, a runtime verification competition collocated with the Runtime Verification (RV) conference, and possible coordination with SygusComp (Syntax-guided synthesis competition) and SYNTComp (Synthesis competition) to arrange special tracks on learning specifications.

5. Creating a public repository containing standard parsers for variety of specification formalisms such as variants of Signal Temporal Logic.

Sessions

The seminar was organized as a sequence of open discussions on pre-defined topics of interest. Each session had one or two moderators who introduced the topic and one or two scribes who recorded the proceedings of the discussions. The moderators had a short introduction of the topic, identifying the most important sub-topics for open discussion. The discussions were structured in following sessions:

Day 1 State-of-the-art in general specification formalisms
1. Specification languages in digital hardware
2. Tools perspective
3. Overview of declarative specification languages

Days 2 and 3 Domain-specific needs and domain-specific specification formalisms
1. Specifications in automotive systems
2. Specifications in robotics and perception
3. Specifications in Industry 4.0, EDA and mixed signal design
4. Specifications in smart cities
5. Specifications in bioloty
6. Specifications in medical devices
7. Specifications in security

Days 4 Expressivity, monitoring algorithms and analysis concerns
1. Algorithms for specifications: specifications for learning *versus* learning specifications
2. Streaming languages
3. Runtime monitoring
4. Expressivity

Day 5 Next steps and summary of the seminar outcomes

We also organized on Day 1 a session to honor the memory of Oded Maler, one of the co-organizers of this seminar, and who sadly passed away in September 2018.

2 Table of Contents

3 Summary of Discussions

3.1 Specification Languages in Digital Hardware

Dana Fisman (Ben Gurion University – Beer Sheva, IL)

Property Specification Language (PSL) is an IEEE standard specification language used in the digital hardware domain. PSL is an example of a successful development of a formal and mathematically rigorous language that is widely adopted in the semi-conductor industry. In this session, we recalled the steps that led to this success story.

Before year 2000, there was no standard for hardware specification language. Users of formal verification tools by different companies used different languages (Motorola's CBV, IBM's Sugar, Intel's Forspec and Verisity's Temporal e). Semi-conductor companies and Electronic Design Automation (EDA) tool vendors decided that there was a need for a standard specification language. A committee with 30 persons was created that included industry representatives but also leading formal methods researchers. 70 requirements were collected and 74 examples of industry properties were expressed in the 4 existing languages. It was decided that 1 of the 4 languages should be used as the basis for the standard, and after two rounds of votes IBM's Sugar was selected. The new language was named PSL.

There were 3 major changes from Sugar to PSL:

- PSL was partitioned into boolean, temporal, modeling and verification *layers*.
- Different *flavors* were added to PSL, supporting Verilog, VHDL, SystemVerilog, SystemC and GDL flavors.
- Linear-time temporal logic was used instead of its branching-time variant.

Major features in PSL are:

- Regular expressions and trigger operations,
- Reset and abort operators,
- Clock/sampling operators,
- Distinction between strong and weak regular expressions, and
- Local variables.

It took 6 years for PSL to become an IEEE standard and 8 years to complete all the features that are in the language today.

3.2 Tools Perspective

Alexandre Donzé (Decyphir – Moirans, FR) and Akshay Rajhans (MathWorks, US)

There are three key personas in the development of CPS: (1) the academic researcher, (2) the tool developer and (3) the industrial practitioner. The application of specification languages in practice is to a large extent conditioned by tool support. We discussed the potential use of specification languages in the development of CPS from the perspective of two tool vendors:

- A major international corporation that specializes in the development of mathematical computing software, which in particular supports data analysis and simulation.

- A small and medium enterprise that develops the toolbox for the creation of test suites, formal requirements and automatic model falsification and calibration.

The first observation made was regarding the commonly accepted assumption in the formal methods community that engineers have the models and that simulation is easy, which turns out to be wrong in reality. In the industrial practice, model-based design (MBD) is its infancy because modeling is hard and adds an additional burden to the engineers.

Requirements play an important role in the development of complex systems. Requirements can come from:

- Marketing and executive managers,
- Designers,
- Test, quality and verification engineers, and
- Certification and regulation bodies.

Requirements can be about many aspects of the design: function, architecture, implementation, integration, performance, memory consumption and energy consumption. One natural question that is studied in the field of requirement engineering is how to effectively communicate requirements, designs, test and certification results, and other design artifacts. Natural language documents are still the primary certificates that requirements are satisfied, and are commonly managed by tools that are organized around a database. An engineer usually creates a written requirement paper trail in a two step process:

- Actually designs and tests the systems.
- Chooses the nicest curve in the set of simulations/data measurement and attaches it to the report/requirement tool.

This process can be improved by using specification languages to express requirements. Specification languages, such as Signal Temporal Logic (STL), provide readable specifications that are actionable on modeling and simulation tools. There is an inherent conflict between the academic researcher, who aims for an elegant and formal specification language with proofs and guarantees, and the industrial practitioner, who wants a scalable and usable solution that works for all use cases and can be adapted to the existing workflows. The tool developer attempts to resolve this conflict. Specification-based tools are used in some industries to support MDB, and were able to successfully find bugs in production models with legacy code, helped synthesize a controller that is entering production and supported the development of a full suite of requirements for a next generation of automotive production model. However, the adoption of such a methodology is not straightforward and some challenges have been identified:

- Practically all requirements are of the form $\mathsf{always}(A \text{ implies eventually } B)$, where A represents steps, ramps and steady-state conditions, and B represents ranges, overshoots and undershoots– do we then need the full power of a formal specification language?
- A often comes from a noisy environment and is hard to capture with a formal specification language such as STL

There are additional challenges due to the heterogeneity of:

- Domains – automotive, avionics, maritime, medical, etc. Each domain comes with its terminology, standards, regulations and best practices.
- Modeling formalisms – acausal, causal, continuous time, discrete time, event-driven, etc.
- Simulation platform: ODE, DAE, FEM, geometric, etc.

An additional challenge is that the continuous engineering support during system operation becomes increasingly important, requiring a design-operation continuum, with time scales that range from milliseconds to months.

The main lesson learned is that tools should support, and not additionally burden, the modeling effort. Crafting requirements is almost as hard as modeling.

3.3 Declarative Specification Languages – Expressiveness

Thomas Ferrère (IST Austria – Klosterneuburg, AT)

Declarative languages based on temporal logic and regular expressions have been popular specification formalisms for decades. In the context of discrete-time Boolean-valued reactive systems, the properties of different specification languages have been extensively studied. Linear temporal logic (LTL) a popular specification language introduced in the 70s has been shown to be expressively equivalent to start-free regular expressions, monadic first-order logic and counter-free languages. Adding the modulo-counting capability gives Linear Dynamic Logic (LDL), which is expressively equivalent to regular expressions and monadic second-order logic. One of the most important results is that all these formalisms translate to finite automata. Both the complexity and the succinctness are well-studied problems for these classes of specification languages.

Increasing the expressiveness of these languages can be done in several directions. For instance, proper counters and adders can be added, resulting in formalisms that translate to infinite state systems. Another way to increase the expressiveness of classical specification formalisms is to add explicit notion of real-time, resulting in formalisms such as Metric Temporal Logic (MTL), Metric Interval Temporal Logic (MITL), Metric Dynamic Logic (MDL), Metric Interval Dynamic Logic (MIDL), Timed Regular Expressions (TRE), Balanced Timed Regular Expressions (BTRE) and Metric Monadic First-Order logic. Allmost all of these languages can be translated to either timed automata (TA) or alternating timed automata (ATA). The dense time lifting of the discrete time formalisms comes at a cost. For instance, universality of TRE is undecidable, while emptiness is PSPACE. Universality is non-elementary for MTL and MDL and EXPSPACE for MITL and MIDL. There are partial results on the expressiveness of real-time formalisms. TRE and MTL are expressively incomparable, star-free MIDL is more expressive than MITL (assuming constants in $\mathbb{Z}$) and MMFO is equally expressive to MTL (assuming constants in $\mathbb{Q}$).

Dense-time formalism have been used as the basis for specification languages applied to cyber-physical systems (CPS). For instance, Signal Temporal Logic (STL) extends MTL with numerical predicates. In the context of CPS, one of the main extensions that increased the power of the specification languages was the introduction of quantitative semantics that allowed to replace the binary satisfaction relation with a real-valued robustness degree, which indicates how well a behavior satisfies or how badly it violates a specification. Another increase of expessiveness was achieved by adding quantification in Signal value-freezing logic and in Signal First-Order Logic.

Some system aspects cannot be specified with any of the described formalisms, such as controllability, security, multi-agent system properties and spatial relations. There are already existing efforts that develop dedicated formalisms, such as hyper-temporal logics and spatio-temporal logics, which address some of these aspects.

3.4 Specification in Automotive Systems

James Kapinski (Toyota Research Institute North America- Ann Arbor, US) and Jens Oehlerking (Robert Bosch GmbH – Stuttgart, DE)

The session was scribed by Katie Driggs-Campbel and Nikos Aréchiga.

In the automotive domain, formal specifications are slowly being integrated into the model-driven development methodologies. However, there are multiple open challenges that were identified in the seminar.

Existing formalisms are difficult for engineers to use. Cars are designed by mechanical engineers, raising the question what type of formalisms are appropriate for people with this background. An ideal formalism shall not only be powerful and expressive, but also succinct and intuitive. We collected several challenges in using formal specifications:

- Requirements often do not match what the engineer mean, making the translation from natural-language requirements to formal specifications non-trivial.
- Specifications also abstract the expected behavior, raising the question how accurate a specification shall be.
- Describing corner cases often results in the blow-up of the specification size.
- Although it is important to improve the specification languages, it is equally important to broaden the education of the engineers.

Some of the above challenges can be addressed by providing structured natural languages and domain-specific template languages to bridge the gap between theoretical computer scientists and mechanical engineers. Another way to bridge this gap is to adapt specification languages to the modeling formalisms, such as Simulink, that the engineers are already familiar with. Many concerns do not seem to be fundamental and rather relate to the lack of tools or poor implementations. There is also a vocabulary gap – engineers in the automotive industry usually use the term "specification" to denote a simulation model of the systems. Finally, regarding the education of engineers, it may be only question of time. For instance, it took decades for model predictive control (MPC) to become mainstream, and that was a matter of graduating enough PhDs who took courses on MPC. Perhaps it is the same for formal specifications, which may become widespread after sufficient people with proper training become available on the market.

Incorporating specification formalisms into the existing development processes represents a considerable effort. For production people, they already don't have enough time. Formal specifications are not yet mandated as a necessity, which prevents their widespread use in the automotive domain. In that context, the aerospace domain seems to be more progressive in terms of adoption of formal specifications.

Increasing the adoption of formal specification could be done by facilitating specification activity. There is existing work on learning specifications from traces, but the quality of learned specs is not always satisfactory (e.g., because specs are complex). To improve the situation, specification mining should have a benchmark; there should be a model and some requirements that are known to be true, and then there should be a competition to see who can identify the most parsimonious requirements. The goal shouldn't be learning one single specification, what you really want is a distribution of beliefs over possible concepts.

Lots of resources are currently devoted to OBD (on-board diagnostic) development. It seems like a great application for specification-based monitoring. This requires certification of monitor compilers and qualified monitors.

There are systemic (organizational) roadblocks to implemented formalized specs (e.g., legacy code, dogmatic development philosophies). How can we mitigate these? If you were thinking in terms of development processes, what could we do differently, in terms of developing new formalisms that minimally impact existing processes?

3.5 Specification in Robotics and Perception

Katie Driggs-Campbell (University of Illinois – Urbana Champaign, US) and Georgios Fainekos (Arizona State University – Tempe, US)

The session was scribed by Ezio Bartocci and Marcell Vazquez-Chanlatte.

Robotic and other autonomous (multi-agent) applications are extremely complex and are difficult to analyze. Typically, the decision flow in autonomous systems starts with the sensor measurements, which are fed to the perception module. The perception module detects objects, tracks lanes, etc. Based on this information, the autonomous vehicles makes tactical decisions and plans the trajectory, which is executed by the low lever controller. Verification and validation is often done in a simulation environment before deploying the autonomous system. We identified several requirements for synthesis and analysis of autonomous systems.

Requirements for synthesis:

- Flexible mission specification and expressivity
- Multi-agent systems and interaction
- Multi-objective tasks
- Assumptions vs. guarantees in unstructured environments

Requirements for analysis:

- Safety assessment and validation
- Reasoning about uncertainty
- Safe domain transfer
- Providing guarantees for complex systems like localization and perception

Based on the identified requirements, the following questions naturally arise:

- Some aspects can be formally specified, but how do you formally prove them?
- Some aspects are less evident to specify – how do you formally specify a classifier?
- → What does and does not make sense to formalize?

There have been different approaches to tackle specification, analysis and synthesis of robotic applications. Instead of doing verification and validation, some researchers and companies suggest incorporating heuristics that avoid putting the autonomous vehicle at fault.

The perception module seems particularly problematic with respect to formal specifications. How would one even describe a property of correct perception (or any other classifier)? The state-of-the-art in vision is to compare the algorithms to some ground truth. This does not seem to be sufficient for safety critical applications, as demonstrated by recent works on adversarial testing. One way to tackle this problem is to specify perception properties indirectly – for instance one can define a specification that requires the object (and its label) permanence once that it is detected.

There have been recently new approaches for testing autonomous systems with machine learning components in the loop. Such techniques allow to reason about the impact of machine learning to the overall system safety.

In autonomous and robotic applications, both assumptions and guarantees are likely to be from time to time violated. In that case, the system should aim for minimum violation of its constraints. There are approaches that address this problem by quantifying violations. Reasoning under uncertainty is a related error, where mostly probabilistic models are used. Specifications are often implicitly embedded in the cost function, with a lot of effort going into cost engineering.

Autonomous and robotic applications are good examples of domains with multiple types of specifications. There is first a distinction between soft and hard constraints. Then, there are functional specifications, but also ethical and legal requirements. There are also descriptions of abstract and concrete scenarios. One way to address the multitude of specification classes is to partially order specifications according to their importance and urgency.

3.6 Specification in Industry 4.0, EDA and mixed-signal design

Gustavo Quirós (Siemens – Princeton, US) and Radu Grosu (TU Wien, AT)

The session was scribed by Thomas Ferrère and Niveditha Manjunath.

The goal of this session was to identify challenges of using specification languages in (1) Industry 4.0 and (2) semiconductor industry. The session was started with the introduction of Industry 4.0 and highlighting the new aspects brought with the so-called fourth industrial revolution: (1) more horizontal organization structure and flexible manufacturing, (2) digitalization of the production, (3) use of artificial intelligence and of digital twins.

Specifications are foreseen to play an important role in many aspects – plants already have requirements regarding cost, efficiency, safety, regulations and feasible production. Specifications can support model-driven development methodologies and monitoring various properties of the plant. The state-of-the-practice is to use natural language for specifications. Currently, only specific aspects of a plant a modeled in a simulation environment. For instance, the whole plant can be modeled as a flow (with the queue semantics) to find bottlenecks, while the behavior of an industrial robot can be explored using simulation with geometry. Process control is typically responsible for monitoring different aspects of the plant. There are visual aids and logical alarm systems observed over multiple shifts, which allows automated monitoring of thousands of variables. The human is nevertheless needed in the loop to supervise the process and intervene if necessary.

Using formal specifications in Industry 4.0 raises some basic questions and challenges:

- What are the boundaries of CPS in Industry 4.0?
  - These boundaries are complex, but need to be well defined
- What aspect should be specified?
  - Specification of the overall process, plan automation and timing, communication, etc.
- What properties should be specified?
  - Behavior, timing, safety, security, cost and productivity, availability, etc.
- What is the intention of the specification?
  - Procurement, scheduling, monitoring, etc.

- Who will write and use the specifications?
  - Device manufacturers, plant designers, integrators etc.
- How to compose, update and maintain specifications?

The semiconductor and Electronic Design Automation (EDA) industries were early adopters of specification languages in the context of digital hardware. This situation is however very different when considering mixed signal design. Mixed signal design brings an entirely new set of challenges. These challenges are illustrated by the difficulty to formally specify a spike, a temporal pattern that is found not only in the semiconductor domain, but also in many other fields such as medical electrocardiograms.

What is a spike? There is no satisfactory formal specification of a spike yet. People use an informal procedural description: "follow the signal up, then down, and do not consider spikes for some time following a spike, etc". This procedural description is translated into a program that detects spikes and outputs segments of the observed behavior that match it. It follows that the spike is defined by the monitoring program itself. This is problematic for several reasons:

- The monitoring program is the spike specification → the program is always correct
- Without a formal specification of a spike, one cannot compare two competing monitoring programs that detect spikes

Describing declaratively a spike in time domain seems to be a challenging problem. The noise in the observed waveform may have a similar shape to a spike. The shape of the signal also varies in frequency. As a consequence, some properties seem to be more naturally expressible in the frequency domain, for instance using wavelets. Another possibility would be to use spatial logics in order to describe patterns in 2-dimensional spectrograms.

The problem of detecting spikes opens another natural question – can we use machine learning to infer a model of spikes from previous observations and labeled data and use it as a classifier to detect newly observed spikes? In principle yes, but we lose guarantees about what we detect and what we miss. The added value of a formal specification with respect to machine learning is that specifications can be understood, interpreted and hence debugged. However, formal specifications and machine learning can be complementary, exploiting the strengths of both approaches.

3.7 Specifications in Biology

Paul Bogdan (USC – Los Angeles, US) and Thao Dang (VERIMAG – Grenoble, FR)

The session was scribed by David Šafránek.

The problem of multi-scale modeling is becoming increasingly important in the areas of intra-cellular processing, cell-to-cell communication as well as communication between groups of heterogeneous cells and populations of different cell types (e.g., microbiome to brain cell interactions).

The key features highlighted in the presentation were the following:

1. Time-dependent signature of biological systems;
2. Stochasticity signature of biological systems;

3. Hybrid modelling (hybrid systems) of biological systems
4. Fractal dynamics (correlated growth) signature of biological systems;
5. Specifying patterns targeting collective behaviour (organisation) at non-equilibrium based on geometric metrics such as orientation angles;
6. Specifying performance of the biological system – typical metrics are not the ones we use in traditional engineering such as delay and throughput, but rather need to quantify and cover learning, adaptation, robustness, and evolution;
7. Need for better metrics to quantify degree of emergence, self-organisation, missing information, intelligence, and complexity;
8. Performance optimization: for a swarm of agents – a performance envelope will imply that the collective system has or exhibits a certain range of degree of emergence, self-organization, robustness, and complexity;
9. Model validation against data: here, formal specifications are hypotheses based on an experimental observation.

Some of the highlights of the discussion were as follows:

- Geometric metrics associated with patterns seem to provide a cost function that is optimised in every step of the dynamics depending on the local neighbourhood (local rules) - indeed these geometric metrics can also be linked with statistical physics concepts such as entropy and free energy.
- Specification of patterns can be probably reduced to the constraints on a cost function, however, find the correct cost function may not be always possible either due to the complexity introduced by the system cardinality, system heterogeneity or unpredictability in reaction behavior to targeted environmental perturbations (the environment does not affect all individuals in the same way).
- The form of hybridisation used for the models (discrete switches vs. ramps) can have implications on optimization, e.g., ramps are better for purposes of optimization.
- Can spatio-temporal behavioral patterns be considered as requirements? The answer was, "In general , yes," but in the context of the discussion, there were only four different collective behaviours considered. The patterns discussed were just representative examples, and there might be richer patterns in reality for testing mathematical formalisms for quantifying emergence, self-organization and complexity of collective biological systems and swarms.
- The problem has a counterpart in synthetic biology – can we design a synthetic population that follows the selected patterns, where patterns can be used to measure system complexity (specification therefore targets the system's complexity)?
- Specifications can often be viewed as probabilistic distributions, so patterns generally describe distributions.
- Specification frameworks should allow to determine critical points that distinguish the patterns through mean-field approximation. There is a need to cope with emergent behaviour – new behaviour not known in advance. Emergence is defined as being proportional with the rate of change of multiscale entropic functionals, which can quantify the seen and unseen.
- Sometimes a specification can be an approximation of the desired property, this does not mean the specification is bad.

Summary

Biological systems that have to be considered and studied in 3D space present a number of mathematical characteristics and challenges for behaviour specification such as time dependent, multiscale, multifractal / hierarchical, heterogeneous, emergent, self-organizing, robust and complex / intelligent behavior. So, how do we mathematically express all these characteristics? Uncertainty quantification is a major challenge. We are required to deal with uncertainty in modeling because it is not guaranteed that all we observe represents all variables that are intrinsic the signature variables of the complex system, the mode structure is uncertain and implicitly also there is uncertainty of the inferred parameters. Models are uncertain (the need for parameter synthesis wrt the given constraints including performance metrics). During the evolution of the population the information entropy and the degree of emergence / self-organisation / robustness / intelligence and complexity change therefore the specification must deal with such time varying collective system metrics and allow to capture emerging knowledge (evolutionary specification). Spatio-temporal reasoning needs to be combined with the ability to describe time varying multi-fractal probabilistic distributions. There is the need to capture time-dependence and higher-order variability (difference between distributions).

3.8 Specifications in Medical Devices, Cyber-Human Systems and Smart Cities

Houssam Abbas (Oregon State University – Corvallis, US), Lu Feng (University of Virginia – Charlottesville, US) and Katie Driggs-Campbell (University of Illinois – Urbana Champaign, US)

This session was scribed by Caleb Stanford.

Medical devices

We discussed some of the issues in specifying CPS specific to medical devices. Some of these issues are:

1. Lack of standardized criteria: Different manufacturers are using different criteria on their devices, and everyone says their algorithm is right. Typically, the overall detection algorithm is structured as a *decision tree* of several different criteria.
2. Lack of a standardized language for different devices: For example, one particular heart monitor vendor uses regular expressions for programming them, but the others don't. It was also discussed how different types of devices (e.g. heart monitor vs. brain EEG) can be very different: although a heart monitor only has 2-3 sensors, EEGs have 100s. And a heart monitor is implanted whereas an EEG is not (so different performance concerns). Can temporal logic support what is needed in all different device types, as well as different device manufacturers for the same type?
3. For high-risk medical devices, resistance to huge changes due to difficulty of the approval process. Each change has to go through clinical trials (small-size, then medium size, then full scale), which takes 2-3 years, millions of dollars, and multiple doctors on staff, as well

as dealing with some legal preparations. The result of this is that incremental changes are preferred, and manufacturers want to be confident of the changes before going through the approval process.

4. Interpretability is key! For medical devices to be useful in practice, doctors need to know what the device is doing and be able to modify a small number of parameters to tune to the specific patient (who will go to regular check ups to make sure the device is working well). There was a discussion about whether neural networks (or some other opaque machine learning model) would be able to overcome this barrier. Would it be possible to adapt the algorithm to a particular patient if it was an opaque machine learning model?
5. Energy consumption is the most important performance concern. If you can make it last 3 more months, over 9 years, that's worth it. Secondary concerns include memory (which is very cheap) and decision time (so medical problems can be detected faster).

Specifications for Cyber-Human Systems

We discussed some of the problems specific to cyber-human systems:

1. Device failure due to human error: Unlike in purely autonomous systems, cyber-human systems can fail due to no fault of the system itself, but due to the fault of the human component. It is an open research challenge how to specify and deal with this kind of error. Error seems to arise especially when humans are new to the system – see plane example, where crashes start out extremely high on each new type of aircraft, but then decay off as time goes on. It turns out this is NOT due to software bugs, but due to human error of the pilots. In the diabetes patient example, patients were ignoring the amount of insulin to inject and instead just doing the maximum, or none at all, in some cases. To deal with human error, traditional research models the *reachable space* of human behaviors.
2. The level of trust that the human has in the system becomes a relevant variable that we have to model. Lots of discussion on why measuring and/or controlling the human trust might be useful, and what approach should be taken. If the goal is to make the human trust the system, research shows humans should be provided with just the right amount of detail – not too little, and not too much – about how the system works. Examples: alarm fatigue; info about how a self-driving car works. There is a logic, PRTL* that models an agent's trust over another. How does trust change over time? A comment was that trust may be related to knowledge about the system (see epistemic logic, common knowledge). But is the goal always to make the human trust the system more? We agreed that if the system may fail, the goal may be to make the human trust it less. So you want to avoid undertrust and overtrust. For measuring trust: heart rate and other physiological signals were suggested. Finally, who should really be trusted in case of disagreement – the human or the machine? Who is the certified agent? This may change over time as systems reach super-human performance; for now in domains like autonomous driving, we trust the human driver more.

Smart Cities

Smart cities represent urban environments where there is a combination of sensor networks gathering real-time data about various aspects of the city, and control algorithms that seek to regulate certain characteristics. Examples include sensor networks to monitor air-quality, noise-level, traffic, wait-times for emergency response, etc. Typical requirements are spatio-temporal, requiring specific geographic locations in the city to have certain restrictions on

the behaviors of constituents of the city such as cars, construction crews, energy consumers, etc. These requirements often require encoding aggregate statistical properties. Certain approaches have sought to control violations of smart-cities requirements through dynamic monitoring and enforcement approaches. In large-scale, distributed cyber-human systems: properties have to be checked locally *distributed monitoring*, not globally. Specifically we discussed smart cities. It's too expensive to aggregate for the whole city, so sensor measurements are aggregated locally.4. In large-scale, distributed cyber-human systems: synchronization of measurements? In current smart cities, the measurements are all at the same rate (1 per minute, e.g.) but not necessarily synchronized due to clock drift or small clock differences. Still, they may be able to be treated as synchronized.

3.9 Specifications for Security

Borzoo Bonakdarpour (Iowa State University – Ames, US) and Bernd Finkbeiner (Universität des Saarlandes, DE)

The session was scribed by Ana Oliveira da Costa.

In the domain of cyber-physical security, use of formal specification languages is fairly new. Inspired by work in cyber-security for software systems, the formalism of *hyperproperties* has been gaining traction. The motivation for hyperproperties comes from information flow and security. The basic idea is that there are two classes (public and secret) for input and output ports, and we want to avoid public outputs to depend on the secret inputs. Checking any property of a single trace does not typically reveal any leakage of secret information; instead we need to typically inspect a set of traces. A hyperproperty is thus equivalent to a set of sets of traces. A system satisfies a hyperproperty if the set of traces generated by the system is an element of the hyperproperty. Traditional formalisms such as LTL and CTL cannot express such properties. HyperLTL solves this limitation by extending LTL with quantification over traces. There are many hyper-logics: HyperLTL, HyperCTL, HyperCTL*, HyperPCTL, HyperPCTL*, HyperSTL (which respectively augment LTL, CTL, CTL*, PCTL, PCTL* and STL with universal and existential quantifiers over trace variables).

We discussed a number of questions related to hyper-properties and adapting them to the domain of cyber-physical systems. As the study of hyperproperties is a relatively nascent field, initial discussions revolved around specific use cases for hyperproperties. In particular, the discussion identified the following system properties that can be expressed using different hyper-property formalisms (such as HyperLTL, HyperCTL, HyperCTL*, HyperSTL, HyperPCTL, etc.):

1. *Observational Determinism*: A general definition of observational determinism is that if the public data in an input $u(t)$ is identical to that of another input (say $u'(t)$) for all time t, then the system observations for $u(t)$ and $u'(t)$ should be identical. A particular definition of this property in HyperLTL was discussed, where the public inputs were required to be identical only initially.
2. *Non-interference*: Various versions of non-interference can also be expressed as hyper-properties, that can also be captured with HyperLTL.
3. *Non-inference*: This is another important property that can be expressed using HyperLTL.

4. *Fault-Tolerance*: The property that the output of a program always produces a string that is a guaranteed Hamming distance away from some other run of the program is a hyper-property that is related to fault-tolerance.
5. *Robustness*: Robustness defined as a specific form of continuity is a hyper-property. Abstractly, this property states that for two inputs to a program at a certain distance d, the outputs of the program are within some factor of the distance d. This hyper-property can be expressed in HyperSTL.
6. *Differential Privacy*: Differential Privacy is a property that says that if we remove any one user's data from a given database, then the results of computing certain kind of statistical properties of the database should not change significantly. The logics HyperPCTL and HyperPCTL* can express such properties.
7. *Causality*: The logic HyperPCTL*, which allows expressing conditional probability was an extension of HyperPCTL that was introduced to express causality properties. One form of causality, known as Granger causality states that a random variable x is believed to be a cause for a random variable y if the correlation of x and y is stronger than the auto-correlation of y. Computing autocorrelations and correlations require expression of conditional probability.
8. *Opacity*: Opacity is a condition used in imperfect information games, where the attacker is required to never know some secret condition of the defender. This is a hyperproperty, and there was a brief discussion on quantifiers in strategy logic, and its relation to the opacity condition for imperfect information games. There was limited discussion on how such questions to be tied to zero-knowledge proofs.

We discussed several theoretical issues related to hyper-properties, many of these questions do not have answers yet, but we are hopeful that the seminar participants will investigate these questions in future research. We enlist the prominent questions raised below:

- Can HyperLTL express input-output automata, which can be useful to specify asynchronous systems?
- What kinds of equivalence relations hold true for models satisfying the same hyperproperties? Logics such as CTL have the property that bisimilar models satisfy the same CTL properties. Is there an analogous property for HyperLTL or other Hyper logics? Linear time properties may be preserved by trace equivalence, what about branching structure?
- What kinds of tools exist for monitoring and model checking hyper-properties? Model checking is not decidable in general, but there are decidable fragments. Is there a taxonomy of Hyper-logics vis-à-vis complexity of satisfiability or model checking?
- Can certain kinds of HyperSTL properties be recast as STL properties? This is always possible for properties which can be checked by creating several copies of the system model and running these several copies independently (on possibly different inputs). However, we discussed some HyperSTL properties where this is not possible. Examplex include properties relating to incremental stability, Lyapunov stability, etc.
- Is there any natural way to have a state-based/automata based verification for hyper-properties? If we prefer automata over LTL formulas, we can use automata as with LTL but we then need to deal (explicitly) with the prefix of the hyper-formula. An automaton that builds a relational formula would be required, where states of the automaton have some explicit relation. For certain fragments, it is possible to build automata that represent self-compositions, but this may lead to an exponential blow-up in the number of quantifiers.

- There was a detailed discussion on the robustness semantics for HyperSTL, specifically the need for an existential quantifier in the robustness semantics, especially when we have stochastic hyper-properties.
- One commenter observed that notions such as bias and fairness in machine learning algorithms can also be expressed as a hyper-property.
- Finally, there was some discussion on testing for hyper-properties. Particularly, it is possible that test coverage could itself be expressed as a hyper-property. Monitoring of a set of set of traces may be useful, and given a set of traces, the goal is to quantify how good this set is. If the notion of robustness is added to the system, then it can be used so check whether the tests have proper coverage.

3.10 Specifications for Learning/AI vs. Learning Specifications

Rupak Majumdar (MPI-SWS – Kaiserslautern, DE) and Marcell Vazquez-Chanlatte (University of California – Berkeley, US)

This session was scribed by Marcello M. Bersani.

Some of the questions guiding the discussion were: (1) How do techniques from formal methods such as reactive synthesis compare against reinforcement-learning based control synthesis? (2) What are specific challenges in CPS that need to be addressed for learning specifications?

Learning Specifications

Learning specifications from data is inspired from automata learning from examples. Inputs to learning algorithms are typically positive and negative examples or unlabeled examples. In automata learning, typically, we learn only from positive examples. In learning specifications, we can either output formulas in some logic, or automata. There was some discussion on whether the result of learning should be an automaton or a logical formula. There were arguments that logic is better for learning because it is better to understand than automata, while there were counter-arguments that automata are actually easier to understand. There was a comment that if there is a fixed structure (either in automata or logic), then it may be useful. Machine learning techniques can, in general, introduce new behaviors in the system, whereas automata-based learning does not permit this.

Specification mining focuses on learning specifications from an implemented system: here, the purpose is to identify a concise way to state how the system behaves. There may need to be a relation between inputs and outputs of the system in such specifications. In contrast, traditional specifications for reactive synthesis focus on which next move the system should make.

There was a discussion on whether STL is a good logic for learning from time-series data. There is a need for benchmark problems to compare the results of learning. A benchmark problem would consist of a well-known system with "ground truth" learned formulae obtained from the system. Using a logical formalism may be helpful to limit exploration in some way. There are many logics where learning remains an open problem. These include spatial, spatio-temporal logics and logics for hyper-properties.

There was a discssion on interpretability of the learned formulas, where it was observed that learning local properties construct a global property might help. There was some discussion on STL where inputs and outputs can be separately labeled, and this could help better learn causality among the events.

There was a discussion on learning abstractions. Based on system traces, we can learn only a partial system. This may not be a right abstraction, but could be viewed as either an over or an under approximation (or a combination thereof). In continuous systems, e.g. linear systems, approaches such as system identification help with learning the system dynamics from data. There does not seem to be a correspondence in logic, possibly because linear systems have many results that come from a frequency domain treatment. Moving to discrete world it's not clear how this can be done.

Specifications for Synthesis vs. Specifications for RL

There was an interesting analogy drawn between specifications based on logical formulas and statistical specifications for system correctness through the use of state-based rewards used in reinforcement learning. With state-based rewards you can play with a dense exploration of the state-space of a system updating rewards closer and closer to the target. We discussed if there is an analogue in the world of logic-based, specification-based synthesis. Some solutions were to use template-based logical formulas, use of counterexamples, and translation of counterexamples into automata to add knowledge to the learning process. Similarities drawn between rewards-based reinforcement learning (RL) and formula-based reactive synthesis were discussed. RL is typically successful because the global behavior is captured with local structures.

There was some discussion on why RL algorithms scale so well vs. reactive synthesis algorithms. An observation was that in reinforcement learning, the reward function used for searching is clear and its memory footprint is known, wheras it is difficult to predict the size of a BDD used for verification during reactive synthesis. An interesting aspect in RL is that multiple rewards can be combined. In reactive synthesis, we can use parity games to synthesize winning strategies. Strategies guaranteeing satisfaction of every ω-regular language can be learned using such techniques.

In RL, during model building there is no assumption on the underlying model and distributions are learned on-the-fly. Aside from learning the main behavior, we can learn also side behaviors that are invariants around that behavior. There are theoretical limits on reactive synthesis based strategy learning: Given a mean payoff game finding an automaton that describes the strategy is an open question.

3.11 Streaming Languages

César Sánchez (IMDEA Software – Madrid, ES)

This session was scribed by Hazem Torfah.

This session was an invited talk by César Sánchez on stream runtime-verification. In stream runtime verification, a formal specification is translated to a stream-based monitor. A stream-based monitor translates streams of input data into streams of output data. Output

streams are used to statistically evaluate and to check assertions over the system. The goal of stream runtime verification is to easily express monitors with formal guarantees that allow for the computation of rich verdicts beyond yes and no.

Stream-based monitors are usually specified using a stream-based specification language. Prominent stream-based specification languages include LOLA Striver and TeSSLa. They provide a simplistic way for defining monitors by defining a set of equations expressing the relation between input and output streams. Stream-based specification languages:

- provide engineering friendly specification languages,
- subsume formal logics like LTL and STL,
- come with a formal semantics that allows for the computation of formal guarantees for a large classes of properties,
- provide an outline approach to runtime monitoring that is independent of the system, and allow for both online and offline monitoring of systems.

Offline vs. Streaming: A key question is how offline monitoring works with future and past, and the answer is that it has the ability to use two-way alternation. In general; however, you want to find algorithms that go in one direction to avoid memory issues if you want to use approaches like map reduce.

Landscape of streaming languages Stream-based specification languages can be classified according to three dimensions:

1. Datatypes
2. Temporal expressivity
3. Time domain

LOLA is a specification language that allows for the definition of monitors for regular properties over rich datatypes. Real-time extensions like RTLola, Striver and TeSSLa allow for the specification of monitors for timed events. Parameterization does not strictly fit into this classification, and requires a fourth dimension. An example of parameterized stream-based languages is LOLA2.0. The advantage of parameterization is that you want the compiler to handle parameterization. Otherwise you have to manage sets in the syntax.

Comparing stream-based verification vs. STL: Streaming languages provide a framework, where STL formulas can be implemented easily. If we talk about future in STL we have to introduce memory bounds. In stream-based languages, one can compute tight memory bounds. Assumptions such as bounded variability ensures bounded memory when considering real-time specifications. Stream-based languages are very controlled and specific languages that allow to encode robustness, but at the price of undefinedness for monitoring. Changing the different versions of robustness is easier in stream languages, and from experience, debugging specifications is easier for engineers. On the other hand, a disadvantage of stream-based languages is that compact formulas in STL need a lot of case distinctions in stream-based languages. Currently, there has been no comparision between expressiveness of real-time stream-based languages and STL. There are also no equivalence results between stream-based languages and a specific class of automata.

3.12 Runtime Monitoring

Caleb Stanford (University of Pennsylvania – Philadelphia, US) and Dejan Ničković (AIT – Austrian Institute of Technology – Wien, AT)

This session was scribed by Rayna Dimitrova.

This session included an invited talk by Caleb Stanford on *Quantitative Runtime Monitoring*, followed by a discussion.

Quantitative Runtime Monitoring

Typical runtime monitoring assumes that we are given a specification that is compiled to a monitor. Quantitative runtime monitoring refers to performing quantitative computations over the input stream. Such computations can include many simple operations such as adding all numbers in the stream, computing an average, or other statistics, but could permit more complicated costs. Applications of quantitative runtime monitoring include medical devices and IoT. There are several types of specification formalisms. LTL-based specification formalisms,including restrictions, extensions of LTL, as well as quantitative extensions such as STL are formalisms based on logic. Formalisms such as RT-LOLA and TeSSLa are more expressive, and are executable specifications (versus declarative specifications such as LTL).

Quantitative Regular Expressions and Cost Register Automata are formalisms that lie between stream-runtime processing languages and declarative temporal logic specifications. We explain a QRE with an example: given an input stream word w and two monitors f1, f2 that produce output on that word, compute max(f1(w),f2(w)). The main idea is to combine regular parsing of the input stream, followed by a computation over that. Example application: input stream of blood pressure measurements, partitioned into episodes of high blood pressure measurements. Match sequence of high blood pressure episodes, then match a whole day. Matching the input stream constitutes the parsing stage, then the computation stage follows. For example, compute the average for each episode, then the maximum over all episodes.

Broadly, different specification formalisms for runtime monitoring can be compared with respect to various criteria such as expressiveness, succinctness, online monitorability, and distributed monitoring. Further, specifications may be discrete-time or dense-time. Formalisms can be low-level or high-level: high-level formalisms need to be interpretable/compilable, low-level formalisms need to be compositional. We now enumerate some of the topics discussed in this session:

1. Expressiveness of QREs: For parsing of the input stream the regular expressions are assumed to be unambiguous. Unambiguous QRE-Past can be compiled to monitors of quadratic size without exponential blow-up. Check for unambiguity is done through type annotations: well-typed QRE areunambiguous. It can happen that the given QRE is not unambiguous, but in practice not that often. QREs support quantitative concatenation: given a word w, for a unique split w=w1w2, compute op(f1(w1),f2(w2)), where op is some operator. QRE supports two types of temporal windows: (1) pattern-based windows: for example, maximum over last 3 episodes, (2) time-based windows: for example, average over last 10 minutes. For each type, there are two variations: tumbling (i.e., "last ...") vs sliding (i.e., "each ..."). One question was why not directly have built-in constructs like

average, but instead have time windows and do computations over them. The reason for this is that the window has to be a regular pattern.F inally, we noted that QRE-Past are efficiently monitorable.

2. Online Monitorability: The classical example of a specification that cannot be checked in a streaming fashion is the LTL property $\mathbf{G}(p \implies \mathbf{F}q)$. However, if $\mathbf{G}$ and $\mathbf{F}$ are replaced with the corresponding past operators, then the specification is streamable (albeit with different semantics). Another option is to consider bounded future operators, which can then be translated to purely past formulas. Determinization incurs exponential blow-up.
3. Overhead of quantitative monitoring: For the Boolean setting, it suffices to do an online subset construction, while in the quantitative setting, with each state there is also an associated cost. One approach: use a table that enumerates all the states and dynamic programming to update cost to be in each state (cost can be infinity).
4. Trade-off between expressiveness of the formalism and compositionality: some streaming algorithms are not compositional, so cannot support arbitrary streaming algorithms.
5. Automata-based approach to monitorability: NFA for classical RE. The benefits are compilation for free, extensibility with additional constructions. Guaranteed streamability and cost of evaluation. Question: How about using RE to sequential circuits construction? Answer: Register automata go in this direction: instead of storing 0 and 1, storing different values.
6. Succinctness: Generally, there is an exponential blow-up in translation of logical formulas, i.e., LTL to Nondeterministic Büchi Automata translation. Translating QREs to cost register automata incurs an exponential blow-up. One solution is to use data transducers (DT) instead. DTs allow for modular compilation and are succinct. DT are expressive, compositional, allow efficient steaming evaluation, and are succinct. Furthermore, every well-typed query α can be translated to a DT of $O(|\alpha|^2)$ size. Evaluation of DT can be done in $O(|\alpha|)$ time per element of input stream (Under the assumption that data values are stored in constant space and operations on data values take constant time). Furthermore, it is feasible to simulate DT in some SRV stream language by simulating the transfer function of the automaton. Encoding LOLA programs into DTs is a bit tricky as at this time, LOLA assumes a single data stream.
7. Distributed online monitoring: The central question in distributed monitoring is whether once can evaluate the monitoring algorithm in distributed fashion instead of sending everything to one location. The high level goal is to optimize. There are many potential applications of distributed monitoring in edge and cloud computing. A key question is whether the distributed monitoring should be done in a bottom up or a top down fashion? Here, bottom-up means that the monitor combine values from different lower-level monitors, while top-down means that there is an explicit global property that is partitioned and distributed to different agents. Top-down is generally hard, as decompositing a global specification is hard in general.
8. QREs and Stream Runtime Verification can be possibly combined. Streaming languages provide the foundation to reason about big data streaming platforms. One issue with QREs and data transducers is that they are not dense time.
9. For monitoring IOT, or during cloud computing, the typical approahc is to use *ad hoc* approaches (e.g., code injected in routers to do filtering/monitoring). On the other hand, for edge computing: – One needs to monitor past.– Typically one needs to monitor sensor data which is slow, even if the edge computer works on much higher frequency. Thus, some latency and low rates are permissible.

10. *Parameter Invariant Monitoring*: Here, the problem is to monitor specifications that may be parametric. This is required, for example, in noisy environments, such as when monitornig patient data, or when we are required to handle inaccuracies of sensors and actuators.
11. *Specification Synthesis*: There are many extant approaches to do specification synthesis. These include learning specifications using a sketching-like approach, where we are given a template specification and are required to learn the parameter valuations. It can be also used to tune partial specifications.
12. *Learning*: Runtime verification also has applications in learning from dynamic environments. It can additionally be used for runtime enforcement and control.

There are several practical applications of runtime verification in various industries. We include a partial list below:

- Inline monitors in Java code.
- Built-in tests in Aerospace: Here, we may have automated tests upon starting the aircraft (check engine), that do runtime monitoring, but may be hand-written.
- Onboard diagnostics in car: These are simple runtime monitors in car, but more applications could be coming with autonomy. Here, some tests may include checking if a given trace that you see is it an execution of some hybrid automaton. Runtime monitors could be used as triggers for data collection, as it is too much to send all the data. Predictive monitors for maintenance such as vibration sensors, temperature sensors, could also be used.
- There may be other approaches required for system-level properties (for example hyperproperties could be used).

3.13 Expressivity extensions to Signal Processing, Spatial Reasoning

Laura Nenzi (University of Trieste, IT) and David Šafránek (Masaryk University – Brno, CZ)

This session was scribed by Necmiye Ozay.

This session discussed extensions to logics such as STL to allow reasoning about dynamical systems, signal processing, spatial reasoning, etc. We discussed three main topics:

Specifications for phase portraits of dynamical systems

Phase portraits of dynamical systems can reveal many interesting properties of the system. Some of these properties include identifying whether a given equilibrium point is a sink, source or saddle, identifying bifurcations, etc. A hybrid logic called HUCTL (Hybrid UCTL) was discussed. The key idea is that elementary patterns describe temporal behavior in states, which could be related to stability, stabilisation, direction of the flow of the vector field, etc. HUCTL is a logic that expresses branching over labeled transitions, future and past state variables. Several interesting patterns were discussed as being expressible in HUCTL; examples include: sinks, sources, 2d-saddle points, states being in nontrivial strongly connected components, etc. Hybrid logics are expressive up to graph isomorphism. State-bindings cause exponential blow up in model checking. One of the examples discussed was expressing monotonicity in biological systems.

STL with memory (freeze operators)

A freeze operator stores the past values of a signal at specific time points. STL* is a logic that augments STL with the freeze operator. STL* has both Boolean and quantitative semantics. STL* is highly expressive; however, monitoring becomes expensive with the number of times a freeze operator (in the value domain) is nested. In applications like biology, nesting of freeze operators is not needed much. The discussion centered around possible applications where we may need to nest freeze operators? One possibility is to specify the property that the signal shape is roughly like a "staircase". STL* can perhaps express derivatives so things like ODEs can be expressed using STL*.

Signal Convolution Logic (SCL): handling noise

Traditional robust satisfaction semantics have focused on robustness in the value domain of a signal. A signal that violates the property *always* $x > 0$ at a singular time-point, e.g. $x(2) = -1$, is treated equally violating as a constant signal with value -1. This is a problem when dealing with noisy signals that may rarely violate a given threshold while almost always satisfying the threshold. The idea in SCL is to have custom operators that can filter noise in the logic. This is done by introducing a convolution operator with various kernels such as the flat, exponential, Gaussian kernel etc. SCL has Boolean and quantitative semantics and allows expressing properties such as 95% of the times, $x > 0$. Applications include diabetes monitoring.

Spatio-temporal Logic

A key question is how to combine spatial and temporal operators? One approach is to consider a signal as a spatio-temporal signal. Spatio-temporal operators such as *everywhere* and *somewhere* can be then used. Satisfaction becomes a multi-valued spatial signal, where satisfaction is a property of a node in a given graph, but is not a property of the graph. Spatio-temporal logics can be used to monitor smart city applications.

3.14 Summary and Next Steps

Dejan Ničković and Jyotirmoy V. Deshmukh

The seminar provided an excellent overview of requirements from various application domains that paved the road for identifying common features in a cross-domain specification language. We divided the summary of the seminar into broad areas and challenges and learnings that the participants identified as the most interesting to them as follows:

Specification Engineering

1. How can we communicating specifications to users?
2. How do we introduce different levels of abstractions for specs, e.g., trace-local vs. trace-global vs. system-level?
3. How do we create frameworks for expressing different formalisms based on how the spec is used (spec for monitoring vs. synthesis vs. learning vs. coverage/test-quality)?
4. Frameworks for evaluating succinctness tradeoffs.
5. Evaluating and quantifying interpretability of specs.

Standardization: Process

1. How can we translate lessons learned during PSL standardization to formulate specification languages for CPS?
2. Perhaps we can have a common parser for some of the popular logical formalisms, inspired by the SMT-Lib effort.
3. We should create benchmark problems and domain-specific tutorials.
4. We should have an industrial advisory board that consists of tool vendors to lead the effort.

Standardization: Tools

1. We identified the need for tools in specification mining, tools that allow the logic being considered to be easily extended.
2. Interoperability between tools was identified as sorely missing.
3. Identification of various benchmark problems for monitoring specifications at runtime, and learning specifications from data is needed.
4. Standardizing syntax for expressing time-series data, such as comma separated values (CSV) with a well-defined header file was suggested.
5. Creating a public repository containing traces, specifications, models, and pattern libraries was suggested.
6. It was suggested that we seek for avenues to increase coordination with competitions like RVComp (for runtime verification), SygusComp (Syntax-guided synthesis) and SYNTComp (Synthesis). These could include special tracks for learning specifications.
7. Creating a public repository containing standard parsers for variety of specification formalisms such as variants of Signal Temporal Logic.

Specification formalisms

There were many participants that felt that they learned about a new specification formalisms from a domain that was not very close to their own:

1. Specifications for stability/robustness,
2. Spatio-temporal logics,
3. Trigger-based syntax for STL-like logics (inspired by PSL),
4. Hyperproperties,
5. Streaming Languages,
6. Spatial Dynamic Logics,
7. Mixed-time logics,
8. Information theory based specifications on emergence

The seminar also recognized that certain formalisms such as epistemic logics, knowledge-based specifications were not discussed. A general question of how we can clean-up specification formalisms to make them easily understandable was also identified as a challenge.

Acknowledgments

This report was partially supported by the Productive 4.0 project (ECSEL 737459). The ECSEL Joint Undertaking receives support from the European Union's Horizon 2020 research and innovation programme and Austria, Denmark, Germany, Finland, Czech Republic, Italy, Spain, Portugal, Poland, Ireland, Belgium, France, Netherlands, United Kingdom, Slovakia, Norway.

Participants

- Houssam Abbas
Oregon State University – Corvallis, US
- Nikos Aréchiga
Toyota Research Institute – Los Altos, US
- Ezio Bartocci
TU Wien, AT
- Marcello M. Bersani
Polytechnic University of Milan, IT
- Paul Bogdan
USC – Los Angeles, US
- Borzoo Bonakdarpour
Iowa State University – Ames, US
- Chih-Hong Cheng
fortiss GmbH – München, DE
- Thao Dang
VERIMAG – Grenoble, FR
- Jyotirmoy Deshmukh
USC – Los Angeles, US
- Rayna Dimitrova
University of Leicester, GB
- Alexandre Donzé
Decyphir – Moirans, FR
- Katie Driggs-Campbell
University of Illinois – Urbana Champaign, US
- Georgios Fainekos
Arizona State University – Tempe, US
- Lu Feng
University of Virginia – Charlottesville, US
- Thomas Ferrère
IST Austria – Klosterneuburg, AT
- Bernd Finkbeiner
Universität des Saarlandes, DE
- Dana Fisman
Ben Gurion University – Beer Sheva, IL
- Felipe Gorostiaga
IMDEA Software – Madrid, ES
- Radu Grosu
TU Wien, AT
- James Kapinski
Toyota Research Institute North America - Ann Arbor, US
- Martin Leucker
Universität Lübeck, DE
- Rupak Majumdar
MPI-SWS – Kaiserslautern, DE
- Niveditha Manjunath
AIT – Austrian Institute of Technology – Wien, AT
- Stefan Mitsch
Carnegie Mellon University – Pittsburgh, US
- Laura Nenzi
University of Trieste, IT
- Dejan Ničković
AIT – Austrian Institute of Technology – Wien, AT
- Jens Oehlerking
Robert Bosch GmbH – Stuttgart, DE
- Ana Oliveira da Costa
TU Wien, AT
- Necmiye Ozay
University of Michigan – Ann Arbor, US
- Pavithra Prabhakar
Kansas State University – Manhattan, US
- Gustavo Quirós
Siemens – Princeton, US
- Akshay Rajhans
MathWorks, US
- David Šafránek
Masaryk University – Brno, CZ
- César Sánchez
IMDEA Software – Madrid, ES
- Caleb Stanford
University of Pennsylvania – Philadelphia, US
- Hazem Torfah
Universität des Saarlandes, DE
- Marcell Vazquez-Chanlatte
University of California – Berkeley, US