ISSN 2186-7437

NII Shonan Meeting Report

No. 205

Formal Method Extensions to Support Domain Theories

Marc Frappier
Fuyuki Ishikawa
Régine Laleau

November 13–16, 2023

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Formal Method Extensions to Support Domain Theories

Organizers:
Marc Frappier (University of Sherbrooke, Canada)
Fuyuki Ishikawa (National Institute of Informatics, Tokyo, Japan)
Régine Laleau (University of Paris-Est Créteil, France)

November 13–16, 2023

Background and introduction

The development of complex systems usually requires multi-view modelling when the systems involve different scientific disciplines and skills [1]. For instance, in the case of autonomous systems, modelling behaviours and interactions of different systems may require control theory concepts, communication protocols, resource allocation, access control rules, time constraints, etc.

However, handling critical complex models related to different engineering domain areas involves modelling concepts that are not explicitly available nor supported by a single formal method. Examples of such domain-specific concepts are reals and continuous functions, classes, and more generally new types with associated operators and properties. Their formal definitions require to start from the core concepts of formal methods i.e. from discrete maths, set theory, category theory, logic and basic arithmetic. This modelling chain could be possible with most of the existing formal methods but it is cumbersome and makes the proof difficult to carry, because of all the details of the encoding of domain-specific concepts in their underlying theory. Moreover, the explicit semantics of these domain-specific concepts are lost by this modelling [2], which makes their reuse not directly possible.

A solution for specifying domain theories consists in using extension mechanisms available in most of existing formal methods. We can cite for instance contexts or theories or species like in Event-B [3], COQ[4], Isabel/HOL[5], Dedukti[6], Nuprl[7], ASM[8], etc. Thus defining a domain theory may consist in specifying data types and their properties, operators to manipulate them with well-definedness conditions that ensure their correct use, axioms and proved theorems. Consequently, in a system to be developed, the domain theory is used to type specific concepts that can be manipulated only by its operators, provided that the well-definedness conditions are satisfied. Theorems defined in the domain theory can be reused in proofs related to the system to be developed. Domain theories offer a more faithful and encapsulated representation of domain concepts, streamlining the reuse of these concepts when building system models, and facilitating their specification, validation and understanding.

Objectives

The objective of the meeting was to discuss the state of the art advances in the definition of domain theories in formal methods. It builds on the results obtained since the 2016 Shonan Meeting 90 on "Implicit and explicit semantics integration in proof based developments of discrete systems" [1]. We wanted to evaluate the main achievements and identify outstanding challenges. Indeed, although the approaches discussed in Meeting 90 made notable impacts, some limitations are identified. The

[1] https://shonan.nii.ac.jp/docs/No-090.pdf

previous approaches supported modeling by the means of ontology or annotations, thus focusing on understanding and description. They did not support domain-specific verification such as proof reuse and recurring proof patterns. This is critical since there is an increasing demand for dependability of cyber-physical systems, which requires complex verification over continuous systems. Then a better understanding of how to formalise domain knowledge in order to efficiently exploit it in comprehensive verification mechanisms is needed.

The following topics were addressed during the presentations and discussions.

- Experiments on using extension mechanisms of formal methods to define domain theories.
- Experiments on using domain theories, defined using extension mechanisms, in the formal development of complex models. Note that formal development not only includes system specification but also verification of system properties. Among others, case studies showing the reuse of domain theories in the development of various complex systems, the feasibility of multi-view modelling in a single formal framework, etc.
- How to handle the semantic heterogeneity that can occur if different domain theories have to be integrated to develop a complex system? In particular, domain knowledge can be twofold: domain knowledge related to scientific data types, like differential equations and time representation, and domain knowledge related to application domains of systems under design, for example imperial or metric units, railway protocols or International Civil Aviation Organization regulations. It may happen that theories related to scientific domain knowledge are formalised in a logic different from the one used in theories related to application domain knowledge. How can these different semantics be aligned?
- How to ensure that the extension mechanism used to import specific domain theories is sound?
- How to handle domain theory validation?
- How to address proof certification issues as heterogeneous proofs may appear due to the use of different proof systems adapted to specific theories?

Overview of the meeting

The meeting had three types of sessions. The first type of sessions had short presentations where each participant introduced their research work, and listed some ideas/challenges/research directions that they would like to discuss during the meeting. Such presentations have been scheduled during the first session of the first day of the meeting.

The second type of sessions consisted of invited talks on some of the above described topics, each of them has given rise to stimulated, often animated, discussions between the participants. This type of sessions has been scheduled during the following sessions of the first day and the first session of the second day.

The last type of sessions consisted in intensive discussions among sub-groups of participants. The topics of the discussions have been decided at the meeting among those proposed by participants. A debriefing plenary session was set up after the end of each sub-group discussion session. These debriefing sessions consisted in two steps 1) first, a summary of the discussions of each sub-group and 2) free discussions among all the participants of the seminar.

Finally, we sorted out a research agenda for formal method extensions to support domain theories, including a planning of a book.

Overview of Talks

App Design with Alloy

Alcino Cunha, INESC TEC and Universidade do Minho Braga Portugal

In his book "The Essence of Software", Daniel Jackson introduced a new theory of software design, where applications are viewed as a collection of interacting so-called concepts, self-contained units of functionality with a clear purpose. This talk showed how Alloy can be used both in the formal design of individual concepts and of applications. For the former, the Alloy module system and a simple state machine idiom can be used to formalize, validate. and verify reusable concepts. For the latter, the desired interaction between concepts can be formalized with action synchronization assumptions specified using Alloy's temporal logic. The talk concluded with a brief discussion about the current limitations of Alloy for formal design in this particular domain.

Generic context instantiation for modeling pure maths

Laurent Voisin, Systerel, France
Main work performed by Jean-Raymond Abrial and Dominique Cansell

A new plugin has been developed within the Event-B-Rodin Plus project (grant ANR-19-CE25-0010 from Agence Nationale de Recherche) which allows proving mathematical theorems within the Rodin platform. It is named the Generic context instantiation plugin. The idea of this plugin is to allow users to enter definitions of new concepts, to prove theorems about these definitions (e.g., interesting properties of the introduced concept) in an Event-B context. Then, both the definitions and their associated properties can be reused in another Event-B context by instantiating the defining context. This permits users to create more and more elaborate concepts and to prove important theorems in pure mathematics. Therefore, this plugin is a good means to formalize pure mathematics and reason about them.

For instance, the plugin has been successfully used to prove the following theorems:

- Zermelo's theorem (every set can be well-ordered)
- Goodstein's theorem (every Goodstein sequence eventually terminates at 0)
- Construction of the real numbers from integers (Eudoxus reals)

Other case studies have been attempted with this plugin, but it did not work as well as for maths. The main reason is that in computer science, in their models, one often uses inductive datatypes, which are awkward to model with the generic context instantiation plugin. The Theory plugin is then more appropriate for these other domains.

Using Deep Ontologies in Formal Software Engineering

Burkhart Wolff, Université Paris-Saclay, LMF, France

Isabelle/DOF is an ontology framework on top of Isabelle. It allows for the formal development of ontologies as well as continuous conformity-checking of integrated documents annotated by ontological data. An integrated document may contain text, code, definitions, proofs, and user-programmed constructs supporting a wide range of formal methods. Isabelle/DOF is designed to leverage traceability in integrated documents by supporting navigation in Isabelle's IDE as well as the document

generation process. In this talk, we extend Isabelle/DOF with annotations of λ-terms, a pervasive data-structure underlying Isabelle used to syntactically represent expressions and formulas. Rather than introducing an own programming language for meta-data, we use Higher-Order Logic (HOL) for expressions, data-constraints, ontological invariants, and queries via code-generation and reflection. This allows both for powerful query languages and logical reasoning over ontologies in, for example, ontological mappings. Our application examples cover documents targeting formal certifications such as CENELEC 50128, or Common Criteria.

Formal Semantics for Statecharts (a Case for Domain Theory)

Son Hoang, University of Southampton, UK

The increased complexity of high-consequence digital system designs with intricate interactions between numerous components has placed a greater need on ensuring that the design satisfies its intended requirements. This digital assurance can only come about through rigorous mathematical analysis of the design. In this presentation, we provide an approach to formalise statecharts semantics using Event-B, with a discussion about the potential application of using the Theory plugin to develop a domain-theory for statecharts.

Time: It is only Logical!

Frédéric MALLET, Université Côte d'Azur, CNRS, Inria, I3S, France

Logical Clocks play an important role for the design and modelling of concurrent systems. The Clock Constraint Specification Language (CCSL) was built in 2009, as part of an annex of the UML Profile for MARTE, to give a proper syntax to handle logical clocks as first class citizens. The syntax gave rise to a series of different semantic interpretations along with various verification tools. Use-cases are diverse and include languages to express timing requirements, temporal or spatio-temporal logics to capture expected safety properties, meta- languages to give an operational semantics to domain-specific languages. The application domains include avionics, safety-critical transportation systems, self-driving vehicles, systems engineering models, cyber-physical systems. This paper reviews the effort conducted since 2009 on CCSL. Researchers there found inspiration in the heritage left by the different schools working around the world on concurrency theory, including the school of synchronous languages from which CCSL has emerged.

Towards a semantic framework for domain theory

Elvinia Riccobene Università degli Studi di Milano, Italy
Patrizia Scandurra Università degli Studi di Bergamo Italy

This talk targets a better understanding of how to support domain-specific theory in Formal Methods (FMs) taking inspiration from ontology engineering. Ontologies are semantic data models for knowledge representation; a formal ontology consists of a set of axioms in some logical language. Automatic reasoning over ontologies allows inferring new knowledge about the underling domain. Ontological engineering has been developed with the goal of reusability (ontology patterns), integration (merging ontologies from different domains), and interoperability (translating among multiple ontologies from the same domain). A FM consists of a modeling notation (with semantics) and a computational model to capture system behavior. Making similarities with ontology engineering, formal

models are data and behavior models for system representation. Reasoning problems with FMs (such as proofs of properties) incorporate domain theories as well as proof strategies/procedures. However, FM extensions engineering for domain theories to achieve reusability (formal analysis patterns), integration (merging system models from different semantic domains), and interoperability (translating among multiple formal models from the same semantic domain) are needed. Towards the achievement of such ambitious goals, we present two (correlated) results we achieved in the past that could be possible starting points for realizing extension mechanisms of FMs in support of domain theory.

First, we recall the definition of a semantic framework for the definition of the semantics of metamodel-based languages based on the ASMETA toolset for Abstract State Machines (ASMs). Using metamodelling principles, we propose several techniques, some based on the translational approach while others based on the weaving approach, all showing how the ASM formal method can be integrated with current metamodel engineering environments to endow language metamodels with precise and executable semantics. In the past, we have applied this semantic framework and the supported techniques to languages of different application domains. Second, we recall an existing classification of forms of model composition: white- (or model composition), black- (or result composition), and grey- box (analysis composition) composition. The white-box approach is the model composition by language integration (i.e., the definition of a new language from a set of individual languages, for example, by metamodel unification or weaving. Black-box composition is the composition of the analysis results; models internals remain encapsulated, only explicitly defined interoperability interfaces are used to access the target analysis and translate back the results. Grey-box composition is the composition of the analysis techniques by orchestrating the steps of two or more analysis algorithms (e.g., composition by co-simulation); internal knowledge of models may be partially exposed through interfaces to guide the coordination. Based on such preliminary considerations, as future lines of research we envisage the need to explore more the concept of extension mechanism of FMs for explicit domain theories definition and composition, and to understand more how to handle semantic heterogeneity and reuse of domain-specific V&V strategies.

USE OF DOMAINS IN CORRECT-BY-CONSTRUCTION PROCESS

Dominique Méry, Université de Lorraine, LORIA, France

The design of correct software based systems requires mathematical verification techniques. These techniques are based on general proof principles, such as induction, and are supported by tools of varying effectiveness, as model-checkers or proof assistants, or by semantical analysis techniques, such as abstract interpretation. Formal techniques or formal methods are generally applied in a posthoc way on the contrary of Correct-by-construction techniques. Correction-by-construction is a method that aims to design both the system in question and its correctness. This method has been assessed and tested in the Cleanroom technique, and we are pursuing this idea by using the incremental design process by refinement.

We present a proposed implementation of this method, illustrating the use of induction to reduce proof effort and to guide the refinement of Event-B models to derive a correct recursive sequential algorithm based on the call-as-event paradigm. This example shows that induction is an important element of a (problem) domain. Other examples show that the problem domain is an element to be taken into account to facilitate proofs of conditions associated with refinement. A second aspect of this notion of domain is linked to the validation of models integrating domains to model hybrid systems. The role of domain theory as promoted by Dines Bjorner is to provide a communication of

domain knowledge while developing system models and each step of so called refinement should be both verified and validated.

Well-definedness conditions are useful for invariants preservation

Yamine Aït-Ameur, ENSEEIHT - IRIT, Toulouse, France

The objective of the presented work is to show how formal methods can be extended in order to handle externally defined data types formalised as algebraic theories. We rely on the Event-B state-based formal method which proved powerful to formalise complex systems and on its extension feature offered by the capability to define new data-types.

The proposed approach is based on the definition of new data types and the associated operators. Each operator introduces well-definedness conditions which generate specific proof obligations each time this operator is used. In addition, these data-types are enriched by once and for all proved theorems, proof and rewrite rules useful to prove the correctness of the models (state-based transition models) that use these data-types.

This approach has been applied to model hybrid systems, knowledge-based systems and ontologies, interactive systems, railway systems etc.

During this seminar, we focus on the reuse of the proposed approach and give methodological rules that help the designer to make use of externally defined data-types in order to extend the formal modelling language with so called "invited semantics".

Composing Software Behaviors

Marc Frappier, Université de Sherbrooke, Canada

Algebraic State-Transition Diagram (ASTD) is a graphical notation that allows for the combination of extended hierarchical state machines with process algebra operators. Its process algebra operators, inspired from CSP, allows for easy composition of software behavior using traditional operators like sequence, choice, iteration, synchronization (weak, à la AND-state of statecharts, and strong, à la CSP's parallel), and timing operators as in timed CSP. Its Statecharts-like state machines allow for a graphical representation of software behavior, making explicit the control flow of a software. State variables can be locally declared, allowing for shared-variables communication between ASTD components. Actions can be specified on transitions to modify state variables that are locally declared. Actions can be declared at any level in an ASTD, allowing for behavior factorisation. Refinement relations were proposed for ASTD, in the traditional style of refining states or transitions in a state machine. Large case studies (eg, ABZ's Landing Gear, Mercedes ELS and SCS, Mechanical Lung Ventilator) have shown that interleaving composition in ASTD often corresponds to Event-B refinement when new behavior based on introducing new state variables are introduced. A more comprehensive theory for ASTD is needed to handle property proofs (invariant, temporal, time, refinement). Invariant proofs have been recently defined for a subset of operators (choice, sequence, iteration). Global properties should be derivable from local properties, in order to simplify proof construction and re-use.

Formal Modelling of Safety Architecture for Responsibility-Aware Autonomous Vehicle via Event-B Refinement

Tsutomu Kobayashi, Japan Aerospace Exploration Agency (JAXA), Japan
Fuyuki Ishikawa, National Institute of Informatics, Japan

For autonomous vehicles (AVs) to be accepted by society, the explainable and robust guarantee of their safety is crucial. However, constructing rigorous models of AV controllers is complex, particularly because they are expected to be safe, goal-achieving, and highperformance. In this talk, we describe our work [9] on modelling, deriving, and proving the conditions for AVs' safety and goal achievement using the refinement mechanism of Event-B. Our method employs (1) RSS (Responsibility-Sensitive Safety) [10], which defines the responsibilities of vehicles in basic traffic situations, (2) GA-RSS (Goal-Aware RSS) framework [11], which extends RSS to derive conditions necessary for safety and goal achievement, and (3) Simplex architecture [12], which can be used for ensuring safety of controller including AI-based components. Our experience demonstrates that the refinement mechanism can be effectively used for the gradual construction of the complex systems for various traffic scenarios.

Challenges in Assurance of Automotive Systems

Mark Lawford, McMaster Center for Software Certification, McMaster University, Canada
Joint work with Nicholas Annable, Thomas Chiang, Richard Paige and Alan Wassyng

Automotive systems represent a previously unprecedented challenge in the development of safety critical software. To be competitive in the marketplace automotive companies need to develop a significantly large amount of safety critical, real-time software in a much shorter time frame than any other industry. On a yearly basis automotive companies are delivering advanced autonomy and electrification features that often incorporate Machine Learning (ML) components. Development is further complicated by the numerous product lines and vehicle variants to have to be supported for extended periods of time. In this talk I make the case that while ML based object detection has helped to enabled autonomous driving features, the failure of these components is problematic and the cause of the majority of autonomous vehicle accidents. Therefore we should be developing safety architectures for object detection systems that incorporate a basic deterministic object detection algorithm that is formally verified and then employed as safety monitor for the sensors used in object detection. Another important aspect of assurance of automotive systems is the complex interaction of electrical, mechanical and software to deliver vehicle level functions across a large product line that has to be maintained over years of service. We illustrate the inherent complexity by examining a recall related to vehicle ignition switches. We then propose our recently developed WorkFlow+ model based technique for managing incremental assurance as a method to help manage the complexity of assurance of automotive systems.

List of Participants

- Yamine AIT AMEUR, ENSEEIHT - IRIT, France
- Étienne ANDRÉ, Université Sorbonne Paris Nord, FRANCE, France
- Toshiaki AOKI, JAIST, Japan
- Alcino Cunha, INESC TEC and University of Minho, Portugal
- Clovis EBERHART, National Institute of Informatics, Japan
- Marc FRAPPIER, Université de Sherbrooke, Canada
- Frédéric GERVAIS, Université Paris Est Créteil, France
- Thai Son HOANG, University of Southampton, UK
- Fuyuki ISHIKAWA, National Institute of Informatics, Japan
- Tsutomu KOBAYASHI, Japan Aerospace Exploration Agency (JAXA), Japan
- Olga KOUCHNARENKO, Université de Franche-Comté, France
- Régine LALEAU, Université Paris Est Créteil, France
- Mark LAWFORD, McMaster University, Canada
- Frédéric MALLET, Université Côte d'Azur, France
- Dominique MERY, University of Lorraine, LORIA, France
- Elvinia RICCOBENE, University of Milan, Italia
- Patrizia SCANDURRA, University of Bergamo, Italia
- Jean-Pierre TALPIN, INRIA Rennes, France
- Laurent VOISIN, Systerel, France
- Burkhart WOLFF, Université Paris-Saclay, France

MEETING SCHEDULE

Check-in Day: November 12 (Sun)

- Welcome Banquet

Day1: November 13 (Mon)

- Presentation of the participants: one introduction slide per participant
- Talks and discussions on topics by participants

Day2: November 14 (Tue)

- Talks and discussions on topics by participants
- Group Photo Shooting
- Define priorities on topics discussion for the rest of the seminar
- Discussion on topics in sub-groups
- Plenary session: Summary of sub-group discussions

Day3: November 15 (Wed)

- Discussion on topics in sub-groups
- Plenary session: Summary of sub-group discussions
- Excursion and Main Banquet

Day4: November 16 (Thu)

- Discussion on topics in sub-groups
- Plenary session: Summary of sub-group discussions
- Findings and definition of priorities and collaborations
- Wrap up

SUMMARY OF DISCUSSIONS

A first discussion was proposed on the subject: "What is a domain theory?". Then four discussions have been chosen and the participants have been divided into four sub-groups, each of them dealing with one discussion.

WHAT IS A DOMAIN THEORY?

Participants: all

A domain theory may be seen as a Domain Specific Language (DSL) equipped with a reasoning support, such as a set of theorems and proof strategies, and a methodological support to capture the know-how and reuse it. It may also be seen as a new abstraction made of two parts, a static one and a dynamic one. It also depends on a generic logic type to be replaced with a specific logic type such as FOL, HOL, Fuzzy, ... The static part is composed of datatypes, operators on the datatypes, axioms and theorems and possibly proof strategies. The dynamic part should allow domain theory evolution and could consist of reasoning tools such as deduction (inference rules) or refinement mechanisms.

Two important questions related to their creation must be considered:

- When do we need to create a new domain theory as opposed to reusing an existing one?
- Who can define domain theory: domain experts and/or engineers?

How can we produce domain theory? by instanciation, composition, projection, extension, ...
Why do we need domain theories? It depends on the viewpoint:

- from a system engineering viewpoint
  - to be more efficient to model systems
  - for reuse (of libraries and proofs),
  - improve maintainability and readability
  - to be less error prone
- from (knowledge, reasoning) representation viewpoint
  - for communication with domain experts and to ease validation
  - a common way to explicit and reason on knowledge
  - improve maintainability and readability
  - to undo abstraction of pure logical systems and to relate to domain knowledge. Here, abstraction means that using logical systems allows to specify any kinds of systems whereas domain theories restrict or guide the way specifications are written. We trade expressivity of a formalism against better usability/computability/provability/automation.

DOMAIN THEORIES DEDICATED TO CYBER-PHYSICAL SYSTEMS

Participants: Clovis EBERHART, Étienne ANDRÉ, Mark LAWFORD, Olga KOUCHNARENKO

Some (more or less) recent attempts were made to build domain theories dedicated to cyber-physical systems (CPS) (works by Dupont et al., RSS works by Tsutomu et al., extensions of Ptolemee, etc.). These domain theories have specificities, such as the discrete/continuous interactions (with tools being often good at either of them, but not both), or the fact that they run in “even more

unknown" environments than other systems. Challenges include the heterogeneous nature of discrete/continuous models, the intrinsic infinite nature of CPS (notably due to their continuous nature), but also the reusability of theories (many existing "theories" in model checking for CPSs are rebuilt from scratch). Another challenge is the gap (to narrow as much as necessary) between the proofs (in idealized abstractions of the real system, including perfect knowledge), and the actual limited knowledge of the system subject to sensor uncertainties and computation failures or communication delays in distributed implementations, etc. This may require dedicated abstractions or approximations, including concretization in the actual system of counterexamples given in the abstract formalism.

Engineers developing CPS are typically interested in verifying safety properties such as collision freedom in the example of the RSS specification of adaptive cruise control. Safety properties might also express user experience properties such as passenger comfort in an autonomous vehicle (e.g. bounded acceleration and jerk values). Progress properties are also of interest, such as deadlock/timelock freedom and actual progress (reaching the passenger's desired destination) as well as more advanced properties such as optimization goals (minimizing energy consumption of the vehicle). While such properties are also typical of purely discrete systems, the interaction of the continuous and discrete components of CPS can make verification of these properties much more difficult.

CONNECTIONS BETWEEN PATTERNS AND THEORIES

Participants: Régine LALEAU, Dominique MÉRY, Laurent VOISIN

First we identified different kinds of patterns:

- Modelling patterns that can be used to write specifications by instantiation:
  - Architectural patterns such as controller-environment patterns for designing cyber-physical systems or attack models for security purposes.
  - Properties patterns such as access control patterns or Dwyer patterns.
  - Refinement patterns such as refinement for executable code.
  - Style patterns for writing formulas such as using characteristic function versus set.
- Design patterns that can be used for building systems or software. They provide technical solutions compared to modelling patterns. We can cite Gamma design patterns, fail-safe by construction patterns or anti-patterns.
- Methodological patterns that provide specific ways or approaches to develop systems, such as top down/bottom up or refinement/abstraction or action/reaction patterns.
- Proof patterns. Note that modelling patterns can also define proof patterns. To some extent, reasoning patterns, such as safety cases for certification, can be considered as proof patterns.

Then we tried to establish links between patterns and domain theories. The first thought that immediately came to mind, is that theories can be used to justify or formalize patterns, mainly for properties, refinement or design patterns.

Two other kinds of links can be considered: using patterns for building theories or using theories for defining patterns. In the first kind, the way theories are built can follow methodological patterns. For instance a theory can be gradually built from basic theories or from scratch. A theory can also come from repetition of the same patterns in different models, which can be then captured in a new theory.

In the second kind, using theories can influence the modelling methodology and then give patterns. For instance, control theory will guide the design of the system and its modelling. Theories can also define style patterns. For instance, properties can be written according to the used theories. Theories can bring vocabulary (with proof techniques, reasoning modes, ...) that make model specification possible. For instance holomorphic functions are mandatory to compute drag and lift for airplane wings.

FORMAL METHODS EXTENSIONS FOR DOMAIN THEORY

Participants: Yamine AIT AMEUR, Toshiaki AOKI, Alcino CUNHA, Thai Son HOANG, Frédéric MALLET, Elvinia RICCOBENE, Patrizia SCANDURRA, Jean-Pierre TALPIN

The group discussed available extension mechanisms of formal methods for domain theory. The discussion brought to the definition of a classification framework for the extension mechanisms supported by different state-based formal methods (e.g., Abstract State Machine/ASMETA, Alloy, B, Event-B), regardless of a specific domain theory. The elements that characterize an FM extension include: type of extension (of the language and/or of the analysis), the purpose of the extension, domain of the extension (e.g., application-specific, mathematical, etc.), mechanism for implementing the extension (e.g., translations, API-based integrations, encoding, orchestration, etc.), and how the extension is activated and (possibly) reused. Concretely, the purposes of such a classification include: To provide a viable base to help FM users and engineers to better understand, design and communicate about FM extensions, using a uniform vocabulary and at a high-abstraction level To provide an up-to-day map of the state of practice in defining state-based FM extensions. To provide a useful framework for evaluating and comparing FM extensions, specific or not for a domain theory. To provide an evidence-based discussion of the emerging trends and gaps and their implication for future research on engineering FM extensions. To provide a foundation for future methodologies, tools and technologies for developing FM extensions. The proposed classification extracts and generalizes concepts from existing FM extensions and from related literature and our previous work. Many questions remain on the precise definition of this classification framework; here, we just started to define this classification generally, based on cross-FM observations.

PROVING SYSTEM PROPERTIES FOR SPECIFICATIONS THAT USE DOMAIN THEORIES

Participants: Yamine AIT AMEUR, Étienne ANDRÉ, Clovis EBERHART, Frédéric GERVAIS, Marc FRAPPIER, Fuyuki ISHIKAWA, Tsutomu KOBAYASHI, Mark LAWFORD, Burkhart WOLFF

In the construction of formal models for software systems, there are two primary artefacts: the formal system model and the domain-specific theory library used in the system model. Since they are constructed separately in general, the group discussion focused on the responsibility of each side: what is the engineer who constructs the system model responsible for, and what about the one who constructs the theory library?

One of the promising ways to the separation of responsibilities is proposed by Aït-Ameur et al. In their approach, the domain-specific theory library should guarantee certain theorems under the assumption of the well-definedness of expressions and predicates. In the system model, usages of expressions and predicates provided by the library should be well-defined.

For instance, a library of autonomous vehicle systems may include the following operator for an update of the road state:

Expression $roadStateUpdate(r : Road, ego : Car)$

definition $\{r' | isPhysicallyPossibleFutureRoad(r, r') \wedge time(r') - time(r) = dt \wedge ...\}$
well-definedness $isRoadStateUpdateWD(r : Road, ego : Car)$
$= \forall$ c. c $\in$ cars(r) $\Rightarrow$ $(0 \leq position(c) \wedge 0 \leq speed(c))$
$\wedge \forall$ l1, l2 . ({l1, l2} $\subseteq lanes(r) \wedge l1 \neq l2) \Rightarrow \emptyset = carsOnLane(r, l1) \bigcap carsOnLane(r, l2)$
$\wedge \forall$ l . $\in$ lanes(r) $\Rightarrow$ ($\forall$ c. d ({c, d} $\subseteq carsOnLane(r, l) \wedge c \neq d \wedge isBehind(c, d)$
$\Rightarrow safeDistance(c, d)$)
...

The library also has the following theorem about the safety of the update of the road's state, which is guaranteed as long as the well-definedness holds:

Theorem roadStateUpdateIsSafe(r : Road, ego : Car)
isRoadStateUpdateWD(r, ego) $\Rightarrow$ isRoadSafe(roadStateUpdate(r, ego))

The engineer who constructs the library is responsible for proving this theorem. Then the system model can be specified using operators from the library as follows:

Event cruise
Then road, ego_car :| road' = update(road, ego_car)
$\wedge$ ego_car' = runAtConstantSpeed(road, ego_car)

The engineer who constructs the system model is responsible for proving the well-definedness condition for each usage of operators from the library (e.g., isRoadStateUpdateWD).

The RSS Verification Study [9] is based on HOL-CSP, a combination of Hoare's and Roscoe's Concurrent Sequential Process (CSP) theory with higher-order logic (HOL). HOL-CSP provides the type $'\alpha$ process. This implies that any mathematical object that can be defined in an HOL type can be used inside events of HOL-CSP, This includes real-time, functions, derivatives of functions, multi-dimensional spaces, etc. Algebraic reasoning over CSP-processes can thus be combined with powerful reasoning over differential equations developed in the HOL-Analysis library.

In HOL-CSP, it is possible to define a "demon"-process that asks non-deterministically the physical state of "actors" (cars, pedestrians, traffic-lights, etc) modeled as timed processes [13]; their local states were combined to a global 'scene'. Actors are modeled as processes that receive knowledge over (a fragment) of the global scene. A 'driving strategy', i.e. a function that chooses a particular acceleration from the set of accelerations specific to the actor depending on the scene, is applied and translated via the 'kinematics' function into the next local state of the actor at the time interval defined by the demon.

Identified issues and future directions

Considering the objectives of the seminar and the topics addressed during the presentations and discussions we can state that several of them are still challenging and need in-depth work. In particular, we all agreed on (i) the handling of semantic heterogeneity between different domain theories; (ii) the soundness of the extension mechanisms used to import domain theories; (iii) the issue of proof certification if different proof systems are used.

Moreover we need to develop more experiments on using extension mechanisms of formal methods to define domain theories and using domain theories, defined using extension mechanisms, in the formal development of complex systems. In particular designing cyber-physical systems requires to take into account not only safety properties but also non-functional properties such as optimization or user-friendly goals, which are more difficult to verify because of the interaction of the continuous and discrete components of such systems.

References

[1] D. Bjørner. Manifest domains: analysis and description. *Formal Asp. Comput.*, 29(2):175–225, 2017.

[2] Y. Ait-Ameur and D. Méry. Making explicit domain knowledge in formal system development. *Sci. Comput. Program.*, 121:100–127, 2016.

[3] M. J. Butler and I. Maamria. Practical theory extension in Event-B. In *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, pp. 67–81, 2013.

[4] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: Coq'Art: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[5] T. Nipkow et al. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002. `http://isabelle.in.tum.de/`.

[6] A. Assaf et al. Expressing theories in the ΛΠ-calculus modulo theory and in the Dedukti system. In *TYPES: Types for Proofs and Programs*, Novi SAd, Serbia, May 2016.

[7] R. L. Constable et al. *Implementing mathematics with the Nuprl proof development system*. Prentice Hall, 1986.

[8] E. Börger and R. F. Stärk. *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, 2003.

[9] T. Kobayashi et al. Formal modelling of safety architecture for responsibility-aware autonomous vehicle via event-b refinement. In M. Chechik et al., editors, *Formal Methods - 25th International Symposium, FM 2023, Lübeck, Germany, March 6-10, 2023, Proceedings*, volume 14000 of *Lecture Notes in Computer Science*, pp. 533–549. Springer, 2023.

[10] S. Shalev-Shwartz et al. On a formal model of safe and scalable self-driving cars. *CoRR*, abs/1708.06374, 2017.

[11] I. Hasuo et al. Goal-aware RSS for complex scenarios via program logic. *IEEE Trans. Intell. Veh.*, 8(4):3040–3072, 2023.

[12] D. T. Phan et al. A component-based simplex architecture for high-assurance cyber-physical systems. In *17th International Conference on Application of Concurrency to System Design, ACSD 2017, Zaragoza, Spain, June 25-30, 2017*, pp. 49–58. IEEE Computer Society, 2017.

[13] P. Crisafulli et al. Modeling and analysing cyber-physical systems in HOL-CSP. *Robotics Auton. Syst.*, 170:104549, 2023.