Aalto University
School of Science

Department of Computer Science

Contributions to multi-view modeling and the multi-view consistency problem for infinitary languages and discrete systems

Licentiate thesis

Maria Pittou

Supervisor Full Professor Stavros Tripakis

Espoo, Finland, 2017

Aalto University
School of Science

Aalto University
School of Science

Abstract of Licentiate Thesis

Author

Maria Pittou

Title of Thesis

Contributions to multi-view modeling and the multi-view consistency problem for infinitary languages and discrete systems

Abstract

The modeling of most large and complex systems, such as embedded, cyber-physical, or distributed systems, necessarily involves many designers. The multiple stakeholders carry their own perspectives of the system under development in order to meet a variety of objectives, and hence they derive their own models for the same system. This practice is known as multi-view modeling, where the distinct models of a system are called views. Inevitably, the separate views are related, and possible overlaps may give rise to inconsistencies. Checking for multi-view consistency is key to multi-view modeling approaches, especially when a global model for the system is absent, and can only be synthesized from the views.

The present thesis provides an overview of the representative related work in multi-view modeling, and contributes to the formal study of multi-view modeling and the multi-view consistency problem for views and systems described as sets of behaviors. In particular, two distinct settings are investigated, namely, infinitary languages, and discrete systems. In the former research, a system and its views are described by mixed automata, which accept both finite and infinite words, and the corresponding infinitary languages. The views are obtained from the system by projections of an alphabet of events (system domain) onto a subalphabet (view domain), while inverse projections are used in the other direction. A systematic study is provided for mixed automata, and their languages are proved to be closed under union, intersection, complementation, projection, and inverse projection. In the sequel, these results are used in order to solve the multi-view consistency problem in the infinitary language setting.

The second research introduces the notion of periodic sampling abstraction functions, and investigates the multi-view consistency problem for symbolic discrete systems with respect to these functions. Apart from periodic samplings, inverse periodic samplings are also introduced, and the closure of discrete systems under these operations is investigated. Then, three variations of the multi-view consistency problem are considered, and their relations are discussed. Moreover, an algorithm is provided for detecting view inconsistencies. The algorithm is sound but it may fail to detect all inconsistencies, as it relies on a state-based reachability, and inconsistencies may also involve the transition structure of the system.

Research field **Computer Science**	**Key words** **multi-view modeling, view consistency, infinitary languages, discrete systems**
Supervising professor **Full Professor Stavros Tripakis**	**Pages** **80**
Thesis advisor **Full Professor Stavros Tripakis**	**Language** **English**
Thesis examiner **Assistant Professor Jan Reineke**	**Date** **13.02.2017**

☒ **The thesis can be read at https://aaltodoc.aalto.fi/handle/123456789/27**

Abstract of Licentiate Thesis

Preface

The present thesis was carried out at the Department of Computer Science of Aalto University during the years 2015-2017, as a partial requirement for the Licentiate Degree. The particular studies and research were funded by the School of Science of Aalto University, which is gratefully acknowledged.

First of all, I would like to express my sincere gratitude to Professor Stavros Tripakis for giving me the opportunity to work in his research group, and for supervising my research work. I would like to thank him for the abundance of knowledge I received in a variety of previously unknown topics related to Computer Science. Moreover, I would like to thank Professor Tripakis for his patience and guidance, for the fruitful discussions, and for giving me the freedom to find my own path with the research. I feel honoured that I collaborated with a professor of such repute in the field of Computer Science.

Moreover, I would like to thank the examiner Associate Professor Jan Reineke for accepting to review my thesis, and for his valuable feedback. I would also like to thank my research group, Iulia Dragomir, Georgios Giantamidis, Srivinas Pinisetty and Viorel Preoteasa, for the informative sessions they offered, and for the supportive working environment they created. I am particularly grateful to my colleague and very good friend Georgios for his unconditional help and support. I would also like to thank my colleague and friend Iulia for her suggestions and corrections in an earlier version of this thesis.

Furthermore, I would like to express my gratitude to the supervisor of my Master thesis, Assistant Professor George Rahonis, for shaping me as a researcher, and for his continuous guidance and support.

I would like to thank the Computer Science Department for supporting competitive research, and offering a high level quality of studies. Also, I thank the administration, the study coordinators, and the secretaries of the School of Science and the

Department of Computer Science.

I warmly thank my close friends Maria Pavlidou, Anna Simeonidou, and Evi Stefanidou, and my students Vera and Nefeli Vegiazi for their support and trust in me. I also thank all my friends here in Finland for creating beautiful memories.

Finally, I express my deepest gratitude to my family, my mother, my brother and my grandmother, for shaping me as an individual, and for their unconditional love and guidance in my life. Without their support my studies and research work would not be feasible.

Espoo, March 8, 2017,

Maria Pittou

Contents

List of Publications

This thesis consists of an overview and of the following publications which are referred to in the text by their Roman numerals.

I Maria Pittou and Stavros Tripakis. Multi-View Consistency for Infinitary Regular Languages. In *XVI International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS 2016)*, Samos, Greece, July 2016. IEEE, 2016.

II Maria Pittou and Stavros Tripakis. Checking multi-view consistency of discrete systems with respect to periodic sampling abstractions. In *13th International Conference on Formal Aspects of Component Software (FACS 2016)*, Besançon, France, October 2016, *Lecture notes in Computer Science*. Springer, 2016.

Author's Contribution

Publication I: "Multi-View Consistency for Infinitary Regular Languages"

The author of this thesis was the principal author and wrote the paper. The second author and supervisor of this thesis contributed with comments.

Publication II: "Checking multi-view consistency of discrete systems with respect to periodic sampling abstractions"

The author of this thesis was the principal author and wrote the paper. The second author and supervisor of this thesis contributed with discussions with the main author. The final version of the publication was edited jointly with the second author.

1. Introduction

System modeling has proved to be an essential technology for reasoning about complex systems. It uses high level abstractions for the description of a system under development, in order to analyse the system and predicate its behavior. This allows capturing errors in the early stages of the design, improving performance, and ensuring the correctness of the system. As software has become omnipresent, systems have grown increasingly large and heterogeneous. Typical examples include mechatronic [17, 35], embedded [23, 21], and cyber-physical systems [30, 25]. For instance, embedded systems are defined by a combination of versatile descriptions such as hardware, software, requirements, as well as design and analysis models. Moreover, in cyber-physical systems, the architectural and functional representations are associated with the descriptions of the physical processes.

The modeling of any large and complex system inevitably depends on a combination of multiple engineering disciplines. Hence, the design of such systems takes place in collaborative distributed environments, and involves many stakeholders. The different stakeholders carry their own concerns and perspectives on the system defined by the knowledge of their discipline. Then, a variety of tools and languages are utilized in order to model the versatile perspectives, and each expert derives their own models for the design of a system. On the other hand, the different perspectives of those involved in the process inevitably intersect and overlap. Therefore, the multiple models that represent the perspectives of the corresponding designers, are different but yet related (Figure 1.1). This practice is known as *multi-view modeling*, and is gaining in popularity since it supports the design of modern systems. In multi-view modeling (MVM for short), the different stakeholders derive separate models, called *views*, of the same system, i.e., a system is described from multiple points of view (cf. [11, 5, 25]). Given a system, its views can be either structural or behavioral or both. Moreover, the views can describe requirements or constraints (for instance for the

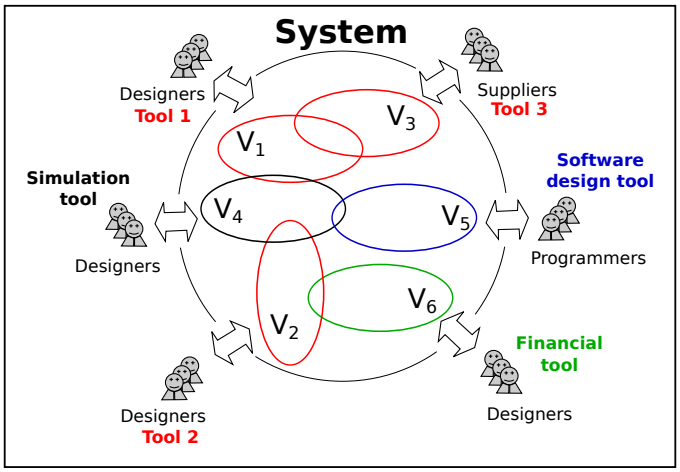


Figure 1.1. Example of multiple design teams involved in the modeling of a system. The different teams carry their own perspectives of the system, and derive distinct models using a variety of tools. Modified figure from [35].

architecture or the dynamic aspects) that should be satisfied by the original system.

Consider for instance the specification of a system within the context of power and thermal aware modeling (Figure 1.2) [21]. The stakeholders involved in the design of the system derive five views, namely the Architecture View (Backbone View), the Performance View, the Power View, the Thermal view, and the Application View. In the sequel, two further (sub)views are used in order to specify the structure and the behavior for each of the five views. For a given view (i.e. Thermal View), the structural (sub)view refers to the state of view from the other views (Floorplan area), and the behavioral (sub)view (Thermal Manager Unit) represents the control of the elements of the view. Then, the stakeholders utilize several means in order to describe their views. However, each of the five views is inevitably related with some of the rest views. The interconnections among the separate views of the designers, can then be described with different types of relations including allocation (for actions that associate the application view with the architecture view), abstraction (for structural elements that occur in more that one views), and characterization (for the behavioral part of a view that is defined by some external physical laws). These relations illustrate that the different views are not independent from each other, and hence they may overlap. This example illustrates one type of a multi-view modeling approach. Other examples of concrete systems, that motivate the multi-view modeling approaches, can be found in [13, 35, 28, 25].

Multi-view modeling techniques are usually either projective or synthetic [38]. In the former case, there exists a single model of the system from which views are derived using a projection mechanism. The views can then be used to serve for other

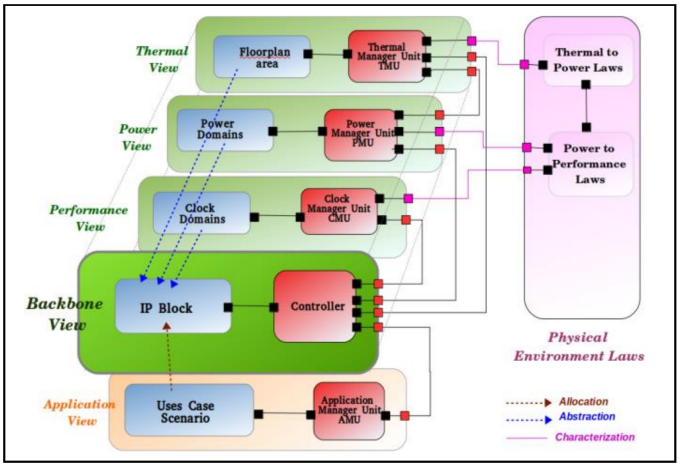


Figure 1.2. Example of the multi-view modeling approach for a system within the context of power and thermal aware design. Figure taken from [21].

purposes (like verification). In the synthetic case, a model for the original system is absent, and is derived by integrating the views into a comprehensive model. In this setting, a basic problem is to ensure *consistency* among the views, i.e., that the views do not contradict each other [13, 4, 25, 32]. Multi-view consistency can also imply that the views for a given system conform to some relationship that is prescribed to hold among them.

Checking for multi-view consistency is not a trivial problem. The difficulty lies on the complexity of the system to be built. In many cases, single views consist of further multiple views in order to provide exhaustive descriptions for the views structure and behavior. On the top of that, views are often described with versatile means. Even in the case of describing the views with the same formalism, the task of identifying the relations among the views, is cumbersome.

Ensuring consistency among the views is one of the main challenges in multi-view modeling. Apart from the *multi-view consistency problem*, there exist other problems related to multi-view modeling. In the synthetic case, a related problem is the *synthesis* of a complete system given a set of consistent views. Other problems include *handling inconsistencies* within the views [12], checking *conformance* of views to a system, and *view reuse* (cf. [25, 32]).

The focus of this thesis is to contribute in the formal study of the multi-view modeling approach, and the multi-view consistency problem for systems and views described as sets of behaviors. Actually, multi-view modeling and the consistency problem occur in the early literature of software engineering. Most of the existing work uses some multi-modeling language (for instance UML or SysML) in order to describe

the different views of a system. Then, checking for multi-view consistency reduces to checking for consistency among the different architectures. On the other hand, multi-view consistency is usually described by non-emptiness of some kind of composition (e.g. intersection) of the corresponding views. However, these work do not provide any formal framework for multi-view modeling. Moreover, a formal definition of the view consistency problem, why is key to multi-view modeling approaches, has been lacking.

Only the recent work [32, 31] provides a generic formal framework for multi-view modeling, and investigates its basic problems. More precisely, in [32] systems are described as set of behaviors within any global universe, and views are also sets of behaviors obtained by some kind of abstraction of system behaviors. In other words, abstraction functions are used to generate views from a system. Then, the notion of multi-view consistency is defined in a strict mathematical way, and the authors provide sufficient and necessary conditions to check for the consistency problem. In contrast to previous work, the authors show that in their framework, non-emptiness of some kind of conjunction among views, does not always imply consistency. In [32] the generic framework is instantiated for discrete systems and the abstraction functions used are variable hidings. Later on, in [31] the framework is also extended to systems described either only by automata or by ω-automata, and the abstraction functions used are projections of an alphabet of events (describing the system domain) onto a subalphabet (describing the view domain).

The present thesis contributes in the multi-view consistency problem for behavioral views. In particular, the problem is studied for two distinct settings, namely, for infinitary languages [27], and discrete systems [26]. Both cases are instantiations of the generic formal framework proposed in [32]. More precisely, [27] (publication I) extends the work of [31] for finite automata or ω-automata to the case of mixed automata (finite automata that accept both finite and infinite words) and the corresponding infinitary regular languages (sets of finite or infinite words, accepted by mixed automata). Essentially, a mixed automaton consists of two independent counterparts, a finite automaton and an ω-automaton. Projections are used as abstraction functions to obtain the views from the system, and inverse projections for the other direction. Then, a systematic study is provided for mixed automata, and their class of their behaviors is proved to be closed under union, intersection, complementation, as well as under projection and inverse projection. The closure properties of the class of infinitary languages are necessary in order to check for multi-view consistency.

Checking for multi-view consistency is perfomed by extending one of the conditions proposed in [31] to the infinitary language setting, and the problem is proved to be PSPACE-complete.

The motivation for studying the multi-view consistency problem for mixed automata lies on the fact that projections may turn an infinite view behavior into an finite one, while inverse projections may turn a finite behavior into a set of infinite behaviors. Moreover, it may be the case that the different views of the system are described by regular or ω-regular languages or infinitary regular languages, which may result in a language containing both finite and infinite words for describing the original system's behavior.

In the second contribution [26] (publication II), the systems and the views are defined by discrete systems. Discrete systems are described symbolically, and their behaviors are finite or infinite sequences of states. In contrast to [32], where the abstraction functions are variable hidings, timing abstractions are used in [26], and in particular periodic samplings. Intuitively, given a positive integer number T (period), the periodic sampling abstraction consists in sampling the system's behaviors once every T steps. Apart from periodic samplings, inverse periodic samplings are also considered, and the closure of discrete systems is studied under these operations. Then, an algorithm is provided in order to detect view inconsistencies. The algorithm is sound and it applies state based reachability in order to find conflicts among the views. However, the algorithm is incomplete since it neglects the transition structure of the views, which may also give rise to inconsistencies.

The initial motivation for studying periodic samplings as abstraction functions originated from the fact that many systems, like embedded systems, consist of components operating over multiple periods. Then, observing the operation of such a system partially, one could periodically sample its behaviors. In case of absence of the system, one would have to synthesize its views, operating at different periods, in order to derive the overall multi-periodic system.

The organisation of this (article-based) thesis is as follows. Apart from this Introduction, the thesis contains 3 sections. Section 2 provides a detailed overview of some representative related work in multi-view modeling. Section 3 presents the summary of the author's contributions on the multi-view modeling problem. Finally, Section 4 concludes the thesis, and discusses possible directions for future research.
The author's work [27, 26] (publications I and II respectively) can be found as they were published at the end of the thesis.

2. Related Work

Multi-view modeling and the multi-view consistency problem occur in the early literature of software engineering within the context of a variety of applications. In what follows we provide an extensive description of some representative related work, and we discuss the state of the art approaches to multi-view modeling.

2.1 Early Work on Multi-View Modeling

One of the early work adressing the multi-view modeling problem is [13]. In particular, [13] provides an informal framework and a prototype to deal with the so called *multiple perspective problem* of composite systems. Composite systems characterize systems which are defined by multiple perspectives (descriptions) using different means. Then, the multiple perspective problem consists in dealing with all these perspectives which are overlapping, and hence there is need for coordination in order to ensure consistency among the several requirements. The framework in [13] is quite informal, and most of the notions are illustrated by examples rather than by formal definitions. However, [13] serves as a nice introduction to the problem and captures the important aspects of multi-view modeling including *viewpoints* and the *consistency problem*. Moreover, the proposed framework is quite generic and it consists one of the first papers that considers the problem of heterogeneity in the models used to describe the different aspects of a system under development.

Motivated by [13], the authors of [11] investigate the view-based approach of system specification. The authors explain that the development of large software systems requires multiple teams of engineers to work concurrently. Each team restricts on different aspects of the same conceptual system and derives separate but overlapping views, for which inconsistencies should be resolved. The authors consider the

approach of using multiple views for the modeling of software systems (viewpoints approach where a viewpoint is defined by a view) combined with the reference model approach, i.e., each team of engineers develops separate views by applying view operations on a given incomplete common reference model. The reference model and the views are defined by typed graphs (graph transformation systems), i.e., graphs which are characterized by a type. Then, view relations are defined to relate either multiple views or the original system with a view. Roughly, a view relation describes a projection of a graph to another. Three types of inconsistencies are considered, two static and one dynamic: (i) The same concept is specified in two different views with different names (investigated a lot in databases concept). To resolve this problem the reference model has to be extended in order to consider the unspecified concepts. (ii) The same names are used in two different views and they refer to different semantic concepts. In this case, the views are kept separate for the overall system or they are both rejected. (iii) A view operation violates the constraints of another view, hence the views overlap, and they have to be integrated (synchronized) into one consistent view. The integration procedure for two views consists in modifying the views so that they contain a same name only if they have a same origin in the reference model. Then, one takes the union of the given views to integrate them.

In [34] the concept of consistency in studied within the context of programming languages. In particular, the authors point out that when building systems, the programmers want to ensure consistency with respect to type safety (i.e., static type checking), and the different source code versions used. To achieve that, the authors develop a programming enviroment, CONMAN, for constructing and debugging inconsistent software systems, where inconsistencies are obtained with respect to source code versions used, and type safety. The central idea in [34] is that some degree of inconsistency in unavoidable within a system, and most importantly it is often more cost effective to obtain a software system with some inconsistencies rather than a system with no inconsistencies at all. For this, there is need to distinguish the severity of inconsistencies and fix the critical ones. Consider for instance, a software program generated by multiple modules (where some of them are dependent), and a type safety bug for an object in some module. A programmer should be able to fix the bug by imposing changes only on those modules affected by the bug. Then, the resulting system may be inconsistent in the perspective that it potentially contains modules where for instance the particular type of the object is different. However, this inconsistency is not problematic as long as it does not affect the correctness of

the program. On the other hand, it would be more cost effective to correct the bug in every module (for instance in some cases fixing a bug introduces others bugs). CONMAN is a tool built upon this idea, such that: (i) Detects and keeps track of different types of inconsistencies without requiring their immediate removal, (ii) Provides a smarter compilation technique that reduces the (type safety) costs by detecting which modules should be modified to restore safety. Smart compilation techniques identify declarations that have changed in files of the program, and enable for recompilation when changes have been applied only to the related files, and (iii) Supports debugging and testing.

A versatile approach for the notion of inconsistency is considered in [7] for systems defined by propositional or first order calculi. In [7] a system, defined by a set of calculi, is considered to be inconsistent if there is a formula of the system such that both the formula and its negation, are theorems of this system. Such a methodology is out of the scope of this thesis, thus it is not discussed further.

Afterwards, a body of research focused on the study of multi-view modeling and its related problems, with emphasis on the consistency problem. Multi-view modeling was further motivated by the proliferation of complex heterogeneous systems. As result, the multi-view modeling idea has been tracked in a variety of design techniques either explicitly or inherently. In what follows, we present some existing work on these techniques.

2.2 Multi-Modeling Languages

Multi-view modeling is supported by multi-modeling languages such as UML and SysML. UML and SysML are standardized visual modeling languages used for the artifacts of software systems. These languages use a plethora of diagrams in order to specify and construct various abstractions of software and hardware systems (i.e., structural, behavioral, specifications etc). SysML actually improves UML by providing additional extensions to address requirements. Several work has focused on solving specific consistency problems among different structures of these languages.

In [24] the authors provide a semantic consistency analysis for Class and Object diagrams. Class diagrams (CD) are UML structure diagrams and represent a static view of a system, while object diagrams (OD) are engineering instances of classes interconnected by links. Semantic variabilities in these two structures create the need to check for inconsistencies in their semantics. For example, one would have to check

whether empty object models are valid for class diagrams, or whether incomplete ODs are consistent with the semantics in the CD, after being completed. In order to check for such inconsistencies, a feature model, in the form of a tree, is initially considered for formalizing the possible semantics of the diagrams. In particular, a tree is formalized for the CDs and the ODs separately, and the semantics of the diagrams are given in the form of sets of objects and relations among them.

The notion of consistency among a set of arbitrary diagrams is defined in [24] such that the intersection of the semantics of the diagrams is nonempty. The (semantic) consistency between a CD and OD is similarly defined. In order to check a CD and OD for consistency, another simple tree is considered with two successors, one for the CD semantics and one for the OD semantics. The authors provide a fully automated way for checking for semantic consistency between the two types of diagrams. In order to achieve that they use a parametrized transformation to the Alloy tool [18]. The input to the transformation is a CD, an OD, and a valid configuration of the CD/OD consistency feature model. Then, some examples are used to illustrate the Alloy module for the transformation. The authors provide a prototype implementation of their work and their evaluation results indicate that the implementation works well for small CDs and ODs, but not for more complicated diagrams.

Another approach is considered in [9], where the authors reduce the multi-view consistency problem to model checking. In particular, the authors develop a tool, called Vooduu, that extends UML in order to enable verification of UML diagrams. Initially, a system is given as a software design and its dynamics is modeled in Poseidon. Then, Vooduu is plugged in Poseidon and generates XML files which trigger the initiation of the verification procedure performed by UPPAAL (cf. [1]). The Vooduu tool allows to automatically check for view consistency of object oriented designs. The views are dynamic and they are described semantically either by state charts or by sequence diagrams, both augmented with communication and timing constraints. Then, the multi-view consistency checking between two such views is reduced to model checking a system described by the state charts against properties described by the sequence diagrams.

Vooduu receives as inputs the views representing the system and the required properties. Then, Vooduu converts the set of state charts describing the system, to a network of timed automata given as XML files. Moreover, the required properties described by sequence diagrams are used to generate observer automata that move to error states whenever a timing violation happens or an erroneous message arrives.

After performing model checking, the given views are considered to be consistent, if the property is satisfied by the system, and inconsistent, otherwise. In the latter case, a counterexample is returned, that is an error trace described by a sequence diagram, and the verification procedure is repeated upon correction.

The extension of UML to SysML resulted in studying multi-view modeling using the latter as a common language for the description of the views for a system. For instance, the focus of [35] is to model the different views of an embedded system with SysML, and to identify possible dependencies among the views. As it is argued in [35], using only SysML as a modeling language is not sufficient to completely represent all the different aspects of an embedded system. For instance, SysML cannot be used for specifying schematics for assembly (EPLAN's function) or simulating control systems and dynamic system behavior (Modelica's function). As a result, the authors propose a different methodology than modeling directly the different views in SysML. Given an embedded system, its multiple descriptions involve the use of different tools and domains. As a first step, the latter are formally abstracted using metamodels in the MOF standard. Then, a SysML profile is generated for each of the metamodels. Actually, the authors customize SysML with profiles (a case of domain specific languages) in order that SysML supports the existence of versatile domains. On the other hand, a generic SysML model for the complete system is provided. Then, it should be the case that the separate views are generated by this generic model, and correspond to the SysML profiles. For the development and integration of views the authors use graph transformation. The framework is illustrated through a specific mechatronic design problem. However, the authors do not investigate the consistency problem in their approach.

In [14] the authors consider views described by hierarchical structures in the SysML language. The authors call these views thematic views and the hierarchical structures, thematic structures. The thematic views and structures are to be described by some model in SysML. To achieve that, a common SysML model will be used in order to generate the several structures according to views they refer to. Most importantly, the modeling procedure should be such that the generated view thematic structures are consistent. A thematic view structure is described by a set of component structures (thematic structures) connected by some kind of relation. The components themselves are also described by a set of components, SysML block instances, which are connected to each other. The authors use a composition relation (the child instance cannot exist independently of the parent block) to relate block instances

that form components, and aggregation relations (the child instance can exist independently of the parent block) for building the thematic structures. Inconsistency issues are discussed in [14], but no formal definition is considered. More precisely, the authors check for consistency automatically by adding ports to the multiple parts of the components that occur in the several views.

Although the aforementioned investigations propose some practical methods for dealing with consistency in UML and SysML design models, they do not consider a formal treatment for the problem. In [3] the authors formalize the consistency problem for dynamic behavioral views described either by a state machine model or as a message sequence chart. The motivation is to reduce the problem of checking a system against dynamic behavioral properties to the problem of checking view consistency. In particular, the authors are interested in safety properties of systems expressed by finite traces. In order to check consistency between a state machine model and a message sequence chart, a metamodel is defined for each one of them, in order to serve as a unified modeling formalism. Then, checking consistency between two views consists in checking consistency between their metamodels.

In [3], a metamodel is defined as a labelled transition system (LTS) with some additional structure on the states defined by sets of objects. Each object is characterized by an identity, its state and a set of operations. The views of a given system are obtained by applying a type of projection operation. The intuition of a system projection is that the resulting system (view) misses both objects from the states and the relevant transitions where the missing objects change their local states. Moreover, each view is again a LTS with some additional constraints (these constraints are expressed as must and never mode of the LTS describing the view). Then criteria are formulated in order to check (i) whether a given view is a view of a particular system, and (ii) whether two views are consistent. The consistency notion between two views is defined as follows. If the two views are of the same mode then they are consistent. If the two views have with different modes, then they are consistent whenever the must and never traces do not intersect for the common set of objects. Then, the algorithm for checking consistency between two views, reduces to checking the nonemptiness of the intersection of the two systems describing the views. Finally, checking whether a system satisfies a property reduces to checking that the property is a view of the system, while checking that a state machine (message sequence chart respectively) satisfies a property reduces to checking the consistency between the view defining the property and the state machine (message sequence chart respectively).

2.3 Multi-View Modeling for Embedded and Cyber-Physical Systems

This section discusses some work on multi-view modeling approaches within the context of embedded and cyber-physical systems.

Cyber-physical systems (CPS) are heterogeneous and as a result their design involves multiple stakeholders whose concerns are related with different aspects of the system. Hence, designers have to identify among a plethora of formalisms, tools and languages for CPS, those artifacts which are appropriated to model the different viewpoints. In [5] the authors develop a framework for the relations among the multiple viewpoints, formalisms, tools and languages, in order to facilitate the modeling procedure of cyber-physical systems. In the proposed framework, given a CPS, the first step is to identify the viewpoints for the system, and then the appropriated formalisms for these viewpoints. Afterwards, one selects the languages and tools that handle such formalisms.

In [5], given a CPS, a viewpoint is obtained as an intersection of the concerns of stakeholders along with the parts of the system they are interested in. The authors illustrate the notion of viewpoints by an abstracted advanced driver assistance system, and they develop their framework for three viewpoints with respect to any CPS, that is robustness, performance, and software design. Then, a survey is provided on the main existing formalisms and their relation with the three viewpoints. The proposed formalisms include (i) finite state machines to describe mainly the discrete but also the continuous dynamics of systems, and hierarchical state machines in order to capture more complex systems, (ii) differential equations for the modeling of the physical part of CPS, (iii) timed automata that extend finite automata with continuous variables called clocks in order to express quantitative time related properties, and hybrid automata where the continuous variables are more complex, modeled by differential equations, (iv) dataflow formalisms, and (v) discrete event formalisms. In the next and final step, the authors survey the main languages and their associated tools in order to handle such formalisms. Model checkers, block diagram languages, and equation-based object-oriented languages are some of the tools mentioned in the paper.

In [25] the authors are also motivated by embedded and cyber-physical systems, and provide a classification of the main approaches to MVM and a classification and comparison of the main types of view relations, in order to associate with each MVM approach the supported view relations. Most importantly, the authors describe the notions of behavioral views, view consistency, and synthesis of views. More precisely,

in [25] a system with heterogeneous nature and multiple requirements is defined as a multi-view system. Multi-view systems are described by multiple and interrelated descriptions called views. The views are actually obtained as instances of a given metamodel that encompasses many views referring to a set of specific semantics. Such metamodels are called viewpoints. The views are considered as abstractions of a given system, and they are defined by a model associated with a set of semantics, so that specific properties are satisfied. The authors introduce the notions of faithfulness, consistency, and completeness, which formalize the properties that should be satisfied for a model describing a view for a given system. Roughly, the faithfulness of a view consists in ensuring that the system belongs to the set of the semantics describing the given view. Consistency is the existence of a system such that all the given views conform to, and completeness implies that a system can be completely defined by its views.

In [25] the main relations between the views (and the viewpoints) are classified in the groups of content, process, and operations. More precisely, content deals with semantic and syntactic relations. The authors explain that a syntactic overlap implies a semantic overlap, while the opposite does not always hold. On the other hand, in case of no semantic overlap the views are considered orthogonal. Moreover, two views can be related by the notion of abstraction (or refinement), that is a view V_1 can be an abstraction of another view V_2 in which case, V_2 is a refinement of V_1. Finally, the association relation occurs when a view is used to connect two or more (base) views. The second category refers to relations over time and causality, and includes precedence (a view exists before another view), dependence (a view contains data of another view), co-dependence, versions (successive iterations of views) and variants (alternatives of the same view). The third type of relations among views, are operations applied to views. Such operations include view composition, projection (where part of the syntactical content of a view is removed), extension (opposite of projection), and synthesis (implementation of the system given a number of views). The authors also discuss the main challenges in MVM. Among them, view consistency, view traceability, and view reuse are considered. Finally, the characterization of the main approaches in MVM is obtained with respect to the main MVM challenges and the aforementioned relations.

Given a system to be constructed, using multi-view modeling approaches, the several views are usually either structural, or behavioral, or a combination of both. In [28] behavioral consistency is discussed within the context of multimodel heteroge-

neous verification. However, the focus of [28] is on the heterogeneous verification rather than solving the consistency problem or synthesizing a model for the system given a set of consistent views (models). In multimodel heterogeneous verification, a system is comprised by many models of heterogeneous types, and the goal is to ensure the satisfaction of specifications at the system level given the satisfaction of specifications at the models level. In [28] the authors derive abstractions of the models in order to obtain criteria ensuring that whenever a model of the system satisfies its properties, then the system specifications are not violated. The contribution of the paper is on developing a verification framework that relies on behavior relations between the models of a system. The proposed framework is generic, in that there is no restriction on the types of behaviors or behavior relations of the models and the system.

In particular, in [28], behaviors are divided into categories according to their type. Let B denote the set of all behaviors of a particular type and consider for simplicity two such types B_1 and B_2. Then, a behavior relation between B_1 and B_2 is defined as usual by a subset of the Cartesian product of B_1 and B_2. Moreover, a model M (and a system model) is defined as the set of behaviors of a particular type allowed by M. Then, a model M_1 is an abstraction of a model M_2 whenever the behaviors of M_2 are a subset of the behaviors of M_1. Similarly, a specification S of a behavioral type B is the set of behaviors in B that satisfy S. Then specifications are defined for a model M, and the notion of abstraction is also extended for specifications. The authors consider two cases for the verification problem: (i) the interdependencies among the models are neglected, and (ii) the interdependencies are taken into consideration by adding constraints on the parameters of the abstractions defining the models and the system. For both cases, the authors develop criteria to achieve verification at the system level, and in the second case they also check that the constraints among the models and the system are consistent.

Consistency is defined in [28] at the level of the constraints of the parameters pertaining to a model. Consider for instance models M_i for $i = 0, 1, ..., n$, where M_0 refers to the (original) system, and M_i for $i = 1, ..., n$ to the different (partial) models of the system. A parameter is a real valued static variable that affects the behavior of the system. Then, the constraints on a set of parameters P, denoted by $C(P)$, can be expressed for instance in first order logic or in some other formalism. Let C_i^M denote the constraints of the parameters of the model M_i for $i = 0, ..., n$, and C_{aux} denote the constraints that capture all the dependencies among the models (including

the original system model). Then, consistency is obtained if the constraints of the system that occur in C_{aux}, restricted to the parameters of the i-th model imply the constraints of the i-th model, for all $i = 1, ..., n$. In other words, consistency is ensured by non-emptiness of the constraints of the system and each of its models.

In subsequent work, the authors of [28] study also the problem of composing components with heterogeneous formalisms using abstractions of components and abstraction functions for relating their different domains [29]. Following the methodology of [28], the framework of [29] is also generic, i.e., there is no restriction on the type of behaviors of the components as well as on the abstractions and the abstraction functions. The authors consider two cases for this problem: the local semantics of the components are (i) from the same behavior domain of the same behavior class, and (ii) from different behavior domains of the same behavior class. The goal of the paper is illustrated by a simple abstract example: Consider two components P_0 and Q_0 with their behavior domains B_0^P and B_0^Q from a behavior class B_0. Let also P_1 and Q_1 denote the abstractions of P_0 and Q_0 with respective domains B_1^P and B_1^Q from another behavior class B_1. The authors propose criteria in order to ensure that the composition of the abstractions P_1 and Q_1 is an abstraction of the composition of the originally given components P_0 and Q_0, both for components of the same behavior domain and different behavior domain.

In [29] each component is described semantically by a model M that belongs to a set of modeling formalisms (a modeling formalism for CPS is for instance hybrid automata). The semantics of M is a set of behaviors derived by a specific behavior domain B. Each behavior domain belongs to a behavior class used to define semantics for CPS models (for instance if the behavior class for CPS is discrete traces, then different behavior domains may be derived by distinct alphabets used to generate discrete traces). Then, the notion of abstraction between two models M_1, M_2 with behavior domain B is similar to [28], i.e., a model M_2 is an abstraction of M_1 if the behaviors of M_1 are included in the behaviors of M_2. A behavior abstraction function is a special case of a behavior relation defined in [28]. More precisely, an abstraction function maps behaviors of a class B_1 to behaviors of a class B_2. Then, the notion of heterogeneous abstraction between M_1 and M_2 extends the notion of classical abstraction, by taking into consideration the abstraction function between M_1 and M_2. Hence, heterogeneous abstraction between M_0 and M_1 with behavior domains B_0 and B_1 is defined such that the abstraction function A associates every behavior of model M_0 in B_0 to a corresponding abstract behavior of model M_1 in B_1. The notion

of behavior localization is used in order to move from the semantics of a behavior domain to another one, given that they both belong to the same behavior class (for instance the two components can be described by transition systems with different alphabets as different behavior domains. So behavior localization consists in defining each component in the alphabet of the other).

Composition is defined in [29] separately for (i) components with the same behavior domain and (ii) components with different behavior domain. In the former case, composition is defined as the intersection of the behavior sets of the given components. In the latter case, one should first obtain abstractions of the given components on a common behavior domain, and then compose these abstractions. Then, the authors describe for the setting of (i) and (ii), the conditions under which the abstraction of the composition of two components for a given system consists in composing the abstractions of the components. Within these conditions the notion of consistency is implicitly described on the level of abstraction functions (which are used to abstract the components for a given system). The abstraction functions are considered to be consistent if it is possible to find a common abstraction function from which the initial abstraction functions can be generated. Finally, the authors illustrate the theoretical concepts of [29] (and [28]) with the example of a cooperative intersection collision avoidance system (CICAS).

Since in the present thesis we are only interested in behavioral views, we do not discuss structural or mixed views. Some work on the latter can be found in [12, 4].

2.4 A Generic Formal Framework for Multi-View Modeling

Most of the work discussed above provides an informal treatment of multi-view modeling or solves specific consistency problems, in particular with respect to behavioral views of a system under development. In other words, a formal framework for multi-view modeling, and a formal definition of the multi-view consistency problem have been lacking. Only recently, the authors of [32] introduced a formal framework for multi-view modeling and studied its related problems, for systems and views described as sets of behaviors. Moreover, the framework is generic as there is no restriction on the kind of behaviors, and abstraction functions used.

More precisely, in [32] a system is described semantically as a set of behaviors and the views are obtained by applying some kind of transformation to the system behaviors. However, there is no restriction on the kind of behaviors used, and they

can be derived from any universe of behaviors (for instance the behaviors could be discrete, dynamic, or hybrid). Formally, *a system $\mathcal{S}$ over a domain of behaviors $\mathcal{U}$, is a subset of $\mathcal{U}$*: $\mathcal{S} \subseteq \mathcal{U}$. The transformations used to derive the views from the system, are defined by means of an *abstraction function* $a : \mathcal{U} \rightarrow \mathcal{D}$, where $\mathcal{D}$ is the *view domain*. A *view $\mathcal{V}$ over view domain $\mathcal{D}$, is a subset of $\mathcal{D}$*: $\mathcal{V} \subseteq \mathcal{D}$.

Some basic problems related to MVM are formalized in [32], including 1) *view conformance* which formalizes how faithful is a view to a system, 2) *view consistency* that is defined by the existence of a (witness) system to which all the views conform to, 3) *view reduction* which allows to optimize some views by using the information contained in other views, and 4) *view orthogonality* for expressing independence among views.

Formally, the multi-view consistency problem is defined as follows. A set of views $\mathcal{V}_1, \cdots, \mathcal{V}_n$ over view domains $\mathcal{D}_1, \cdots, \mathcal{D}_n$, are *consistent with respect to a set of conformance relations* $\models_i \subseteq 2^{\mathcal{D}_i} \times 2^{\mathcal{U}}$ (either derived from given abstraction function a_i and partial order $\sqsupseteq_i$ over $2^{\mathcal{D}_i}$, or defined as a primitive notion), if there exists a system $\mathcal{S}$ over $\mathcal{U}$ so that $\mathcal{V}_i \models_i \mathcal{S}$, for all $i = 1, \cdots, n$. Such a system $\mathcal{S}$ is called a *witness system* to the consistency of $\mathcal{V}_1, \cdots, \mathcal{V}_n$, and if there is no such system, then the views are considered to be inconsistent.

2.4.1 Instantiating the Framework for Symbolic Discrete Systems

In [32] the authors instantiate the aforementioned generic formal framework for systems and views described by symbolic discrete systems. Semantically, a symbolic discrete system (discrete system for simplicity) is described by a set of behaviors, i.e., sequences of states, which are obtained as valuations over a finite set of variables. Formally, let X denote a (finite) set of boolean variables for describing the state space. Then, a *state* s over X is a function $s : X \rightarrow \mathbb{B}$, where $\mathbb{B}$ denotes the set of Booleans. A *behavior over* X is a finite or infinite sequence of states over X, $\sigma = s_0 s_1 \cdots$. Then, a discrete system $\mathcal{S}$ over X is described semantically as a set of behaviors over X, i.e., $\mathcal{S} \subseteq \mathcal{U}(X)$, where $\mathcal{U}(X)$ denotes the set of all possible behaviors over X.

Two categories of discrete systems are considered, fully-observable discrete systems (FOS), which do not have hidden variables, and discrete systems with internal variables. Syntactically, a *fully-observable* discrete system (FOS for short) is defined by a triple $S = (X, \theta, \phi)$ where X is the finite set of boolean variables, θ is a boolean expression over X characterizing the set of all initial states, and ϕ is a boolean expression over $X \cup X'$, where $X' := \{x' \mid x \in X\}$ is the set of the next state variables.

ϕ characterizes pairs of states (s, s') representing a transition from s to s' of S. Then, $\theta(s)$ denotes that s satisfies θ, and $\phi(s, s')$ denotes that the pair (s, s') satisfies ϕ, i.e., that there is a transition from s to s'. A *behavior* of a FOS (X, θ, ϕ) is a finite or infinite sequence of states over X, $\sigma = s_0 s_1 \cdots$, such that σ can be generated by the FOS, i.e., such that $\theta(s_0)$ and $\forall i : \phi(s_i, s_{i+1})$.

In the sequel, a particular type of projection operation is introduced, namely, variable hiding. Given a state s over the set of variables X and a subset $Y \subseteq X$, the *hiding function* h_Y projects s onto the set of variables Y, hence h_Y hides from s all variables in $X \setminus Y$. Then $h_Y(s)$ is defined to be the new state s', that is, $s' : Y \to \mathbb{B}$ such that $s'(x) = s(x)$ for every $x \in Y$. The notion of variable hiding is then lifted at the level of behaviors and systems. If $\sigma = s_0 s_1 \cdots$ is a behavior over X, then $h_Y(\sigma)$ is a behavior over Y defined by $h_Y(\sigma) := h_Y(s_0) h_Y(s_1) \cdots$. For a discrete system over X, the variable hiding consists in applying the variable hiding on each of the behaviors of the system.

The authors prove that FOS are closed under intersection, but not under union and variable hiding. Then, the authors extend fully-observable discrete systems to discrete systems with internal variables. Formally, a *discrete system with internal variables*, is a tuple $S = (X, Z, \theta, \phi)$ where X, Z are disjoint finite sets of variables such that X describes the set of observable variables, and Z the set of *internal* (unobservable) variables. The initial condition θ is a boolean expression over $X \cup Z$, and the transition relation ϕ is a boolean expression over $X \cup Z \cup X' \cup Z'$. A behavior of a discrete system with internal variables $S = (X, Z, \theta, \phi)$ is a finite or infinite sequence of states over $X \cup Z$ which can be generated by S, in the same manner as with behaviors generated by a FOS. Then, discrete systems with internal variables are proved to be closed under union, intersection, and variable hiding.

The abstraction functions used in [32] for generating views from discrete systems, are variable hidings. Essentially, given a discrete system defined over some variables, then a view is obtained by hiding some of these variables. Since FOS are not closed under variable hiding, there exist FOS over a set of variables X, that do not have a canonical view with respect to a subset of variables $Y \subseteq X$. As a result, the authors investigate the existence of least or greatest views described by FOS, for a given system with or without internal variables. In particular, it is proved that for such systems there exists always a least view, but this is not the case for the greatest view. Then, the view conformance and the view consistency problems are considered for discrete systems. For the latter variable hidings are used as abstraction functions

and the problem is proved to be PSPACE-complete.

2.4.2 Instantiating the Framework for Languages and Automata

In [31] (journal version of [32]), the authors instantiate the generic formal framework for systems defined either only by (i) automata, or (ii) ω-automata (specifically Büchi automata). Then, an abstraction function is a projection of an alphabet of events onto a subalphabet, where the former refers to the domain of the system and the latter is the view domain. More precisely, consider an alphabet Σ and a finite or infinite word w over Σ. The projection of w over some subalphabet $\Sigma' \subseteq \Sigma$, denoted by $\Pi_{\Sigma \to \Sigma'}(w)$, consists in hiding from w all the letters that belong in $\Sigma \setminus \Sigma'$. Moreover, the projection of a language L over Σ (that contains finite or infinite words) onto the subalphabet $\Sigma' \subseteq \Sigma$, denoted by $\Pi_{\Sigma \to \Sigma'}(L)$, consists in projecting every word of L onto Σ'. Inverse projections are also defined for finite and infinite words, and the relevant languages. Then, projection and inverse projection constructions are developed both for finite automata and Büchi automata, and the closure of their languages is proved under these operations.

The multi-view consistency problem is investigated in [31] for languages, regular languages, and ω-regular languages. For all these cases the notion of consistency is defined with respect to $=$ as a partial order, which is a special case of the definition introduced in [32]. In particular, two languages L_1, L_2 (describing two views) on alphabets Σ_1, Σ_2, respectively, are consistent if there exists language L on some alphabet Σ such that $\Pi_{\Sigma \to \Sigma_i}(L) = L_i$, for $i = 1, 2$. The authors prove that the alphabet Σ may be unknown, but necessarily a superset of $\Sigma_1 \cup \Sigma_2$.

Moreover, the authors formulate two necessary and sufficient conditions for checking consistency between views described by languages. Intuitively the first condition expresses that given two views described by languages L_1 and L_2, over alphabets Σ_1 and Σ_2, to check for view consistency one has to obtain the inverse projections of L_1 and L_2 onto the alphabet $\Sigma_1 \cup \Sigma_2$, then construct the language of their intersection, and finally check whether this language projected onto Σ_1 and Σ_2, results in languages equivalent with the two starting languages, respectively. In the positive cases, the given views are considered to be consistent, and inconsistent, otherwise. In case of consistency, the language of the intersection of the inverse projections of L_1 and L_2 over the alphabet $\Sigma_1 \cup \Sigma_2$, describes the witness system to the consistency of the given views. This condition indicates that is sufficient to search for witness languages over $\Sigma = \Sigma_1 \cup \Sigma_2$. According to the second criterion, given two views described

by languages L_1 and L_2, over alphabets Σ_1 and Σ_2, one has to project the languages onto the intersection of their alphabets $\Sigma_1 \cap \Sigma_2$, and if the resulting languages are equivalent, then the views are consistent, and inconsistent, otherwise.

The consistency problem is solved in [31] separately for regular languages (equivalently for finite automata), and ω-regular languages (equivalently for Büchi automata). For both cases, two variants of the multi-view consistency problem are considered. More precisely, given two finite automata M_1, M_2 (describing views) on alphabets Σ_1, Σ_2, the first variant searches for the existence of a language L over the alphabet $\Sigma \supseteq \Sigma_1 \cup \Sigma_2$ such that $\Pi_{\Sigma \to \Sigma_1}(L) = L(M_1)$ and $\Pi_{\Sigma \to \Sigma_2}(L) = L(M_2)$. The second variant searches for a finite automaton M over the alphabet $\Sigma \supseteq \Sigma_1 \cup \Sigma_2$ such that $\Pi_{\Sigma \to \Sigma_1}(L(M)) = L(M_1)$ and $\Pi_{\Sigma \to \Sigma_2}(L(M)) = L(M_2)$. The problems are different since the witness system in the first case is a language, but in the second case is a language accepted by a finite automaton. However, the authors show the equivalence of the two variants. These two problems are similarly defined for Büchi automata. Finally, the authors prove that both for finite and Büchi automata the problems are PSPACE-complete.

2.5 Other Approaches to Multi-View Modeling

2.5.1 Metamodeling

Metamodeling (cf. [20, 19]) is a model transformation method that occurs often in multi-view modeling approaches. Given a system under development, the different views used to describe the system, are often expressed in versatile formalisms or modeling languages. In this setting, is essential to derive metamodels for the given views in order to relate the views, and then construct a comprehensive metamodel for the original system. Moreover, translating the views in metamodels allows capturing dependencies which otherwise would be intractable. On the other hand, metamodels can also be used to describe generic incomplete models, serving as reference models, in order to build detailed view models.

Several of the related work discussed in the previous subsections, encompasses the approach of metamodeling in order to relate the views with each other or with the original system [11, 3, 35].

2.5.2 Aspect-Oriented Modeling

Another methodology for modeling systems described by multiple concerns, is aspect-oriented design [6]. Aspects are used to describe behaviors that are tangled and scattered across a system. Aspect-oriented techniques have been proved essential for modularizing crosscutting concerns. Similarly to multi-view modeling, the stakeholders for each concern are interested in identifying conflicts among the concerns, which is checked through the composition of concerns.

The recent work of [22] indicates that aspect-oriented techniques can also be combined with multi-view modeling. Indeed, the authors propose the use of aspect-oriented techniques in order to improve the scalability of multi-view modeling, and check for consistency among the views through aspect composition and reuse. Any concern, i.e., a view, is modeled as a reusable aspect consisting of three counterparts: structural (UML class diagram), state (UML state diagram), and message view (UML sequence diagram). Then, checking for consistency is performed at three levels: for each individual aspect independently, among all the aspects, and for each aspect with respect to the final (base) model.

2.5.3 Interface Theories

Interface theories provide a method to construct a system incrementally by using partial system descriptions, and hence they can been seen as an alternative to multi-view modeling. Interfaces, are abstractions of components so that they capture only the necessary information for the component, which implies that some information is missed. Then, the goal is to compose the given interfaces using specific operations, so that the assumptions of all the components are satisfied. This implies that the notion of view consistency is defined for the interface theories by a special type of interface conjunction. Some extended work on interface theories can be found in [8, 2, 10, 37, 16].

2.6 Conclusion

Most of the existing work in multi-view modeling focuses on solving specific consistency problems without providing a generic framework for the problem. For instance, in [9, 35, 24, 14] the different views of a system are described by diagrams of some multi-modeling language (UML or SysML). Then, the notion of consistency among

diagrams is described by non-emptiness of some special kind of conjunction (in this case, intersection of the semantics of the diagrams). Most importantly, in these works there is no formal treatment of the multi-view modeling approach and the consistency problem. Even the formal approach followed in [3], falls into the same class with the previous work, since it is not generic. Indeed, the views are described by diagrams (state charts or message sequence charts), which are then translated to a type of metamodels (labelled transition systems), and checking view consistency reduces in checking the non-emptiness of the intersection of views.

Behavioral consistency is discussed in [28, 29] within the context of heterogeneous verification and composition respectively. Following the previous work on multi-view modeling, consistency is defined in [28] by non-emptiness on the constraints of the system behaviors and each of its models behaviors, and in [29] by non-emptiness of some kind of composition of the heterogeneous components for a given system. Moreover, the focus of these two works is on the heterogeneous verification rather than solving the consistency problem.

Only later on, a generic formal framework for multi-view modeling and its basic problems is developed, where both the views and the systems are described as sets of behaviors [32, 31]. Moreover, the notion of consistency presented there differs from previous work in multi-view modeling. In particular, the authors show that obtaining non-emptiness of some kind of conjunction among views, does not always imply view consistency. However, the proposed framework can capture the notion of conjunctive consistency, and hence, is a generalization of the previous conjunctive approaches [31]. Finally, the authors instantiate the generic framework for discrete systems [32], and for languages, regular languages, and ω-regular languages [31].

In the present thesis the focus is on studing formally the multi-view modeling approach and the multi-view consistency problem for behavioral views. For this, the author follows the formal framework of [32], because it is also generic with respect to the descriptions of the systems and the views, and the abstraction functions. The author studies the multi-view consistency problem for two distinct settings, namely, for infinitary languages [27], and discrete systems [26]. More precisely, in [27] the problem is studied in the context of mixed automata, which accept both finite and infinite words, and the corresponding infinitary regular languages. Actually, [27] extends the work presented in [31], where a given system and its views are described only by regular languages or ω-regular languages. In the sequel, [26] studies the multi-view consistency problem for discrete systems, defined as in [32], but the ab-

straction functions used are timing abstractions (periodic samplings) in contrast to variable hidings that were investigated in [32]. A detailed summary of these two contributions is provided at the next section.

3. Overview of Contributions to Multi-View Modeling

The interest of the author is on studying formally multi-view modeling and the multi-view consistency problem for systems and views described as sets of behaviors. In particular, the author contributes with two publications (I and II), that investigate the multi-view consistency problem for two distinct settings, infinitary languages, and discrete systems. This section provides an overview of the two contributions, which are also included at the end of the thesis.

3.1 Overview of Publication I

In [27] (publication I), the author studies multi-view modeling and the multi-view consistency problem for infinitary languages. The setting of [27] is a concrete instantiation of the generic formal framework for multi-view modeling introduced in [32]. In the latter, the system behaviors can be defined within any global universe while the views may be obtained by some kind of transformation, like abstraction functions, to the system's behaviors. Formally, *a system* $\mathcal{S}$ *over a domain of behaviors* $\mathcal{U}$, *is a subset of* $\mathcal{U}$: $\mathcal{S} \subseteq \mathcal{U}$, and an *abstraction function* is defined by $a : \mathcal{U} \to \mathcal{D}$, where $\mathcal{D}$ is the *view domain*. Then, a *view* $\mathcal{V}$ *over view domain* $\mathcal{D}$, *is a subset of* $\mathcal{D}$: $\mathcal{V} \subseteq \mathcal{D}$.

More precisely, the work of [27] actually extends [31]. In the latter, the system and view behaviors are described either only by regular languages or ω-regular languages. Then, projections of an alphabet (system domain) onto a subalphabet (views domain) are used as abstraction functions to obtain the views from a system, and inverse projections for the other direction. In [27] the author studies the problem with respect to the same abstractions functions, i.e., projections (and inverse projections), but for the case where the system and the views are described by mixed automata, which accept both finite and infinite words, and for the corresponding infinitary reg-

ular languages.

The motivation for this setting lies on the fact that projections in general may turn an infinite behavior into a finite one, while inverse projections may turn a finite behavior into a set of infinite behaviors. In order to illustrate this with some examples, some preliminary notions are introduced. For a given alphabet Σ, Σ^* denotes the set of all finite words over Σ, and Σ^ω the set of all infinite words over Σ. The set of all words over Σ is Σ^∞, i.e., $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$. Moreover, a $*$-language (star language) L on Σ is a set of finite words, subset of Σ^*, i.e., $L \subseteq \Sigma^*$. An ω-language (omega language) L on Σ is a set of infinite words, subset of Σ^ω, i.e., $L \subseteq \Sigma^\omega$. An ∞-language (infinitary language) L is a set of finite or infinite words, subset of Σ^∞, i.e., $L \subseteq \Sigma^\infty$.

Consider now a finite or infinite word w over some alphabet Σ. Then, the projection of w over some subalphabet $\Sigma' \subseteq \Sigma$, consists in hiding from w all the letters that belong in $\Sigma \setminus \Sigma'$. As a result, the projection operation applied on a star language derives always a star language, but when applied on an omega language it may derive an infinitary language in general. For example, if $\Sigma = \{b, c\}$ and $\Sigma' = \{b\}$, then for the language $L = b^*c^\omega \cup c^*b^\omega \subseteq \Sigma^\omega$, we obtain that the projection of L from Σ onto Σ' is the language $b^* \cup b^\omega \subseteq \Sigma'^\infty$. On the other hand, given a (finite or infinite word) w over some alphabet Σ, its inverse projection onto $\Sigma' \supseteq \Sigma$ is the set of all words, that if projected onto Σ they coincide with w. The inverse projection of a finite word can be either a set of finite or infinite words, while the inverse projection of an infinite word is always a set of infinite words. For example, if $\Sigma = \{b\}$ and $\Sigma' = \{a, b\}$, then for $w = b \in \Sigma^*$ we obtain that the inverse projection of w onto Σ' generates $a^*b(a^* \cup a^\omega) \subseteq \Sigma'^\infty$. With similar arguments, one obtains that the inverse projection of a star language is an infinitary language in general. As a result, it appeared interesting to consider the multi-view consistency problem for the generic case of infinitary languages.

In [27] is studied the case that a system and its views are described by infinitary regular languages. Therefore, a model is proposed for automata that accept infinitary languages, namely mixed automata. Such automata have been defined in the literature [33], but a systematic study has been lacking. In [27] a mixed automaton is a pair of a finite automaton and an ω-automaton. The ω-automaton is fixed to be a Büchi automaton, and both counterparts are assumed to be nondeterministic, hence the mixed automaton is also nondeterministic. In particular, a nondeterministic mixed automaton (NXA for short) over a finite alphabet Σ is defined as a pair $M = (A, B)$ where $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ is a nondeterministic finite state automaton and $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$ is a nondeterministic Büchi automaton, with

$Q_A \cap Q_B = \emptyset$. The language $L(M)$ of the mixed automaton M is defined by the disjoint union $L(M) = L(A) \cup L(B) = \{w \in \Sigma^* \mid w$ is accepted by $A\} \cup \{w \in \Sigma^\omega \mid w$ is accepted by $B\}$. Every NXA M over Σ can be considered either as nondeterministic finite automaton whenever $C_B = \emptyset$, or as a nondeterministic Büchi automaton, whenever $F_A = \emptyset$.

In order to study the consistency problem for infinitary languages, there is need to investigate several closure properties for their class. More precisely, mixed automata are proved to be closed under union, intersection, complementation, as well as under projection and inverse projection. The two latter operations are based on the constructions of projection and inverse projection for finite and ω-automata studied in [31]. Then, the closure results for mixed automata are used to consider the multi-view consistency problem in the infinitary language setting. Given a set of views, the notion of consistency in [27] is defined with respect to a set of abstraction functions and the partial order $=$. More precisely, a set of views $\mathcal{V}_1, \ldots, \mathcal{V}_n$ over view domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ respectively, are *consistent with respect to a set of abstraction functions* $a_1, ..., a_n$, if there exists a system $\mathcal{S}$ over system domain $\mathcal{U}$ so that $\mathcal{V}_i = a_i(\mathcal{S})$, for all $i = 1, ..., n$. This definition for consistency is a special case of the generic definition provided in [32] (see also subsection 2.4 of this thesis).

In the sequel, two variants of the consistency problem are investigated for the setting that the views of a system are described by ∞-regular languages. Given a set of views described by nondeterministic mixed automata, the first problem searches for a witness system described by an infinitary language, while the second one, for a witness system defined by a regular infinitary language, i.e., accepted by some nondeterministic mixed automaton. The two variants of the consistency problem are then proved to be equivalent, using similar arguments to those developed in [31]. Consistency in [27] is actually checked by extending the first of the sufficient and necessary conditions of [31], discussed in subsection 2.4, to the case of mixed automata (or for infinitary languages). Informally, given a set of nondeterministic mixed automata M_1, M_2 over two alphabets Σ_1 and Σ_2, describing views, one has to compute their inverse projections onto the alphabet $\Sigma_1 \cup \Sigma_2$, and if the intersection of these inverse projections projected back to the original alphabets, derives languages equivalent to those accepted by the views, then consistency is ensured. In case of consistency, the previously described intersection defines the witness system. Finally, the multi-view consistency problem is proved to be PSPACE-complete exploiting the known PSPACE-completeness result of language equivalence for nondeterministic finite au-

tomata and nondeterministic Büchi automata (cf. [15, 36]).

3.2 Overview of Publication II

In [26] (publication II) the multi-view consistency problem is studied for discrete systems with respect to timing abstractions, and in particular periodic sampling abstraction functions. Similarly to publication I, this work is also a concrete instantiation of the multi-view generic formal framework introduced in [32]. The latter, investigates also the multi-view consistency problem for discrete systems, but the abstraction functions used there are variable hidings.

Following [32], discrete systems are described symbolically, and they are used to describe both systems and their views as sets of behaviors. In particular, given a finite set of variables X, a state is a valuation of X over the set of Booleans. Then, a behavior over X is in general a finite or infinite sequence of states over X, and semantically a discrete system $\mathcal{S}$ over X is a set of behaviors over X, i.e., $\mathcal{S} \subseteq \mathcal{U}(X)$, where $\mathcal{U}$ denotes the domain of possible behaviors. Then, discrete systems are divided in fully-observable discrete systems (FOS) without internal variables, and non-fully-observable discrete systems (nFOS) which contain also hidden variables (non-fully-observable discrete systems characterize exactly the discrete systems with internal variables from [32]). For non-fully-observable discrete systems, there is a distinction between their observable and unobservable behavior. The former is defined with respect only to the set observable variables of the nFOS, while the latter takes into consideration all the variables in the system (see subsection 2.4.1 of the thesis for the formal definition of the aforementioned notions). Obviously, FOS are a special case of nFOS, and the latter are used in [26] to describe the views of system.

The abstraction functions studied in [26] are periodic samplings in contrast to variable hidings considered in [32]. Intuitively, given a period which is a positive integer number T, the periodic sampling abstraction consists in sampling the system once every T steps. For instance, given a system behavior which is a sequence of states $s_0 s_1 s_2 \ldots$, the periodic sampling w.r.t. $T = 2$ produces the abstract behavior $s_0 s_2 s_4 \ldots$. Formally, given a finite set of variables X, a domain of behaviors $\mathcal{U}(X)$ and a view domain $\mathcal{D}(X) = \mathcal{U}(X)$, a periodic sampling abstraction function from $\mathcal{U}(X)$ to $\mathcal{D}(X)$ w.r.t. period T and initial position τ, denoted by $a_{T,\tau}$, is defined by the mapping $a_{T,\tau} : \mathcal{U}(X) \to \mathcal{D}(X)$ such that for every behavior $\sigma = s_0 s_1 \cdots \in \mathcal{U}(X)$, $a_{T,\tau}(\sigma) := s'_0 s'_1 \cdots \in \mathcal{D}(X)$ where $s'_i = s_{\tau+i\cdot T}$ for every $i \geq 0$. Lifting this notion at

the discrete systems level, the periodic sampling is applied on each behavior of the system.

Apart from the (forward) periodic samplings, inverse periodic samplings are also considered, since they could serve for defining a witness system in case of consistency among the views. An inverse periodic sampling abstraction function from $\mathcal{D}(X)$ to $\mathcal{U}(X)$ with $\mathcal{D}(X) = \mathcal{U}(X)$, w.r.t. period T and initial position τ, denoted by $a^{-1}_{T,\tau}$, is defined by the mapping $a^{-1}_{T,\tau} : \mathcal{D}(X) \to \mathcal{U}(X)$ such that for every behavior $\sigma = s_0 s_1 \cdots \in \mathcal{D}(X)$, $a^{-1}_{T,\tau}(\sigma) := \{\sigma' \mid \sigma' = s'_0 s'_1 \cdots \in \mathcal{U}(X)$ s.t. $s'_{\tau+i\cdot T} = s_i, i \geq 0\}$ or equivalently $a^{-1}_{T,\tau}(\sigma) := \{\sigma' \mid a_{T,\tau}(\sigma') = \sigma\}$. Then, given a system $\mathcal{S} \subseteq \mathcal{U}(X)$, the inverse periodic sampling is defined by $a^{-1}_{T,\tau}(\mathcal{S}) := \bigcup_{\sigma \in \mathcal{S}} a^{-1}_{T,\tau}(\sigma)$.

The closure of FOS and nFOS is studied with respect to the above operations, and both FOS and nFOS are proved to be closed under periodic samplings, while only nFOS are closed under inverse periodic samplings. Then, three variations of the multi-view consistency problem are considered for a given set of views described by nFOS, and the relations of these problems are also discussed. Note that for such an investigation, only the observable behaviors of the nFOS are taken under consideration (cf. [32]). More precisely, given a finite set of nFOS S_i over the same domain of observable variables X, describing views, and periodic samplings a_{T_i}, for $1 \leq i \leq n$, the three problems consist in checking whether there exists (i) a system $\mathcal{S}$ over $\mathcal{U}(X)$, or (ii) a nFOS system S, or (iii) a FOS system S, such by applying each of the periodic samplings a_{T_i}, for $1 \leq i \leq n$, on the system, one obtains semantic equivalence with the given views (for the cases (ii) and (iii) the set of observable variables of S is the set X). This distinction is rational since the first problem asks for a *semantic* witness system, not necessarily representable as a symbolic discrete system. Moreover, the existence of a FOS witness system, implies the existence of a nFOS, and semantic witness system, while a nFOS witness system does not always imply the existence of a FOS witness system. This fact is proved in [26] with a counterexample. On the other hand, it is open whether the finite-state nature of nFOS is enough to represent all possible semantic witnesses of consistent nFOS views.

In the sequel, a sound (but incomplete) algorithm is provided in [26] for detecting inconsistencies among a finite number of views with respect to periodic sampling abstraction functions. The notion of inconsistency in this setting expresses that the given views return different sets of states for the same positions with respect to the behaviors of the original system. These are actually the positions in behaviors of the original system that are multiples of the least common multiple of the periods that

the views have been sampled with. Hence, the algorithm tries to detect such positions for a given set of views, and therefore it applies to sets of views that satisfy one of the following conditions: either every view generates only infinite behaviors; or every view generates only finite behaviors. Additionally, the initial position of the periodic samplings is assumed to be $\tau = 0$ for every view.

The algorithm exploits known techniques from automata theory, and hence it actually checks consistency among a set of views described as finite automata. For this, the first step of the algorithm consists in generating automata simulating the behaviors of the given views (nondeterministic finite automata for views with finite behaviors only, and nondeterministic Muller automata for views with infinite behaviors only). Then, the algorithm involves a special construction, a finite automaton, that encodes the positions where the views should return the similar sets of states. A type of composition is defined among this construction, and the modified versions of views into deterministic finite automata (determinization procedure allows to trace all the possible states for the positions under consideration). Applying state based reachability, the algorithm detects whether an inconsistency is found or not. In the latter case, the algorithm cannot ensure that the views are consistent (incompleteness of the algorithm), as the algorithm neglects the transition structure of the system (and the views), which is proved to generate inconsistencies as well. This fact is illustrated in [26] by a counterexample.

3.3 Conclusion

To conclude, [27, 26] (publications I and II respectively) contribute to the study of the multi-view consistency problem for systems and views described as sets of behaviors. In [27], the author extends the multi-view modeling approach of [31] for finite automata or ω-automata to the case of mixed automata. In other words, it is proved that the results presented in [31] (section 4 of the paper) are also valid for the setting of infinitary languages. In the sequel, [26] studies a different instantiation of the generic formal framework proposed in [32]. In particular, the focus is on discrete systems with views derived by applying periodic samplings. In comparison to [27], [26] does not investigate complete conditions (sufficient and necessary) for ensuring consistency, but focuses on detecting inconsistencies among the views.

For further details on [27, 26] the reader is pointed to the corresponding publications included at the end of the thesis.

4. Conclusions and Perspectives

This thesis focuses on the formal study of multi-view modeling and the related multi-view consistency problem. Multi-view modeling uses multiple separate models, called views, for the design of a system under development. In the presence of complex heterogeneous systems, such modeling approaches are becoming of high importance. On the other hand, multi-view modeling encompasses the challenge of ensuring consistency among the different views. This stems from the fact that, although, the views refer to partial aspects of a system, they inevitably overlap with other views. Moreover, when developing a view one may inherently make (possibly incorrect) assumptions for some other views. Hence, a key challenge is to check for multi-view consistency.

The main parts of this thesis is the overview of representative related work, and the further study of the multi-view consistency problem for behavioral views. In the literature review both theoretical and practical investigations for multi-view modeling are considered, and some related approaches are discussed as well. Then, the multi-view consistency problem is investigated for the distinct settings of infinitary languages, and discrete systems. The settings developed in [27, 26] are concrete instatiations of the generic formal framework introduced in [32]. Previous work solved the consistency problem for discrete systems with respect to variable hidings as abstraction functions [32], and for regular or ω-regular languages separately, where the abstraction functions used were projections [31].

In [27] the author studies the case of describing a system and its views by ∞-regular languages or equivalently by the relevant automata accepting such languages. A model for accepting such languages is proposed, namely mixed automata. Mixed automata simply simulate a classical finite automaton and an ω-automaton using two separate counterparts. In order to study the consistency problem, a systematic study for mixed automata is necessary. Hence, the closure of the class of languages ac-

cepted by these automata is proved for the operations of union, intersection, complementation, as well as under projection and inverse projection. Then, exploiting these results, and extending the techniques introduced in [31], the multi-view consistency problem is studied. Moreover, a complexity result proves the PSPACE-completeness for the problem in this setting.

In the sequel, in [26] the author studies the multi-view consistency problem for symbolic discrete systems with respect to timing abstractions and in particular periodic sampling abstraction functions. A discrete system is defined semantically by a set of finite or infinite sequence of states, and the states are obtained as a valuation of a finite set of variables to the set of Booleans [32]. Two cases of discrete systems are considered, fully-observable discrete systems which do not contain hidden variables, and non-fully-observable discrete systems which contain both observable and unobservable variables. Obviously, the latter category of systems encompasses the former systems, and thus non-fully-observable discrete systems are used to describe systems and views. Then, a periodic sampling abstraction function is defined by sampling the system's behaviors once every T steps, where T denotes the period. Inverse periodic samplings are also considered, and the closure of FOS and nFOS is investigated. In particular, it is proved that both FOS and nFOS are closed under periodic samplings, but only the latter are closed under inverse periodic samplings. Then, three variants of the multi-view consitency problem are considered, with respect to the type of the candidate witness system (semantic system, nFOS, or FOS). In comparison to [32, 31, 27], the framework of [26] does not provide complete conditions (both necessary and sufficient) for view consistency. Instead, it provides an algorithm for detecting some view inconsistencies. The proposed algorithm is proved to be sound, but incomplete, meaning that whenever the algorithm does not report an inconsistency then the views may be consistent or not. This results from the fact that the algorithm uses state based reachability to detect inconsistencies, and hence the transition structure of the given views is neglected.

A limitation of [26] and [27] is that the views of a system are described with the same formalism, mixed automata, and discrete systems respectively. However, distinct means are usually required for defining the views, since the latter represent different aspects of the system. On the other hand, multi-view modeling techniques often search for a common metamodel in order to translate the initially distinct models for the views. Then, one can easier detect inconsistencies among the views, which otherwise would be intractable. Hence, one could use mixed automata or discrete

systems in future work, to serve as metamodels, for the design of some systems under development. Future work also includes the investigation of open questions from [26]. More precisely, one could develop a complete view consistency algorithm. Another open question is whether the problem of searching for a semantic witness system to the consistency of a set of views, described by discrete systems (nFOS) and obtained with periodic samplings, is equivalent with the problem of searching for a nFOS witness system.

In general, possible directions for future research could tackle other abstraction functions than projections or periodic samplings. Moreover, one could study heterogeneous instantiations of the multi-view modeling framework, i.e., using timed automata to capture behaviors in continuous time, or hybrid views in order to combine the discrete and continuous framework. Apart from these theoretical directions, future work includes implementations of the existing formal approaches [32, 31, 27, 26]. In particular, the framework developed in [26] could potentially serve for the modeling of embedded systems, where multi-periodicity is present. Different views could model the system's behavior on different periods, using discrete systems and periodic samplings. Then, in case of absence of the original system, one could develop techniques for checking for view consistency, and in positive cases for synthesizing a model for the original system. Finally, it would be interesting to consider real case studies that build upon the methodology of the generic formal framework for multi-view modeling [32].

References

[1] G. Behrmann, A. David, K. G. Larsen, P. Pettersson, and W. Yi. Developing UPPAAL over 15 years. *Softw., Pract. Exper.*, 41(2):133–142, 2011.

[2] A. Benveniste, B. Caillaud, A. Ferrari, L. Mangeruca, R. Passerone, and C. Sofronis. Multiple viewpoint contract-based specification and design. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *Formal Methods for Components and Objects, 6th International Symposium, FMCO 2007, Amsterdam, The Netherlands, October 24-26, 2007, Revised Lectures*, volume 5382 of *Lecture Notes in Computer Science*, pages 200–225. Springer, 2007.

[3] P. Bhaduri and R. Venkatesh. Formal consistency of models in multi-view modelling. In *2002 Workshop on Consistency Problems in UML-based Software Development, 2002*, pages 149–159, 2002.

[4] A. Bhave, B. H. Krogh, D. Garlan, and B. R. Schmerl. View consistency in architectures for cyber-physical systems. In *2011 IEEE/ACM International Conference on Cyber-Physical Systems, ICCPS 2011, Chicago, Illinois, USA, 12-14 April, 2011*, pages 151–160. IEEE Computer Society, 2011.

[5] D. Broman, E. A. Lee, S. Tripakis, and M. Törngren. Viewpoints, formalisms, languages, and tools for cyber-physical systems. In C. Hardebolle, E. Syriani, J. Sprinkle, and T. Mészáros, editors, *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling, MPM@MoDELS 2012, Innsbruck, Austria, October 1-5, 2012*, pages 49–54. ACM, 2012.

[6] S. Clarke and E. L. A. Baniassad. *Aspect-oriented analysis and design - the theme approach*. Addison Wesley object technology series. Addison-Wesley, 2005.

[7] N. C. A. da Costa. On the theory of inconsistent formal systems. *Notre Dame Journal of Formal Logic*, 15(4):497–510, 1974.

[8] L. de Alfaro and T. A. Henzinger. Interface theories for component-based design. In T. A. Henzinger and C. M. Kirsch, editors, *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings*, volume 2211 of *Lecture Notes in Computer Science*, pages 148–165. Springer, 2001.

[9] K. Diethers and M. Huhn. Vooduu: Verification of object-oriented designs using UPPAAL. In K. Jensen and A. Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part*

of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings, volume 2988 of *Lecture Notes in Computer Science*, pages 139–143. Springer, 2004.

[10] L. Doyen, T. A. Henzinger, B. Jobstmann, and T. Petrov. Interface theories with component reuse. In L. de Alfaro and J. Palsberg, editors, *Proceedings of the 8th ACM & IEEE International conference on Embedded software, EMSOFT 2008, Atlanta, GA, USA, October 19-24, 2008*, pages 79–88. ACM, 2008.

[11] G. Engels, R. Heckel, G. Taentzer, and H. Ehrig. A combined reference model- and view-based approach to system specification. *International Journal of Software Engineering and Knowledge Engineering*, 7(4):457–477, 1997.

[12] A. Finkelstein, D. M. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. In I. Sommerville and M. Paul, editors, *Software Engineering - ESEC '93, 4th European Software Engineering Conference, Garmisch-Partenkirchen, Germany, September 13-17, 1993, Proceedings*, volume 717 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 1993.

[13] A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *International Journal of Software Engineering and Knowledge Engineering*, 2(1):31–57, 1992.

[14] M. Florian, A. Albert, W. Daniel, and B. Matthias. Multi-view modeling in sysml: Thematic structuring for multiple thematic views. In A. M. Madni and B. W. Boehm, editors, *Proceedings of the Conference on Systems Engineering Research, CSER 2014, Redondo Beach, CA, USA, March 20-22, 2014*, volume 28 of *Procedia Computer Science*, pages 531–538. Elsevier, 2014.

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[16] T. A. Henzinger and D. Nickovic. Independent implementability of viewpoints. In R. Calinescu and D. Garlan, editors, *Large-Scale Complex IT Systems. Development, Operation and Management - 17th Monterey Workshop 2012, Oxford, UK, March 19-21, 2012, Revised Selected Papers*, volume 7539 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2012.

[17] R. Isermann. *Mechatronic Systems Fundamentals*. Springer, 2005.

[18] D. Jackson. *Software Abstractions - Logic, Language, and Analysis*. MIT Press, 2006.

[19] E. K. Jackson, T. Levendovszky, and D. Balasubramanian. Automatically reasoning about metamodeling. *Software and System Modeling*, 14(1):271–285, 2015.

[20] E. K. Jackson and J. Sztipanovits. Formalizing the structural semantics of domain-specific modeling languages. *Software and System Modeling*, 8(4):451–478, 2009.

[21] A. Khecharem and R. De Simone. A Multi-View Co-Modeling and Co-Simulation Framework for Heterogeneous Embedded Systems. In *eSAME 2015 - Embedded software and micro-electronics conference*, Sophia Antipolis, France, Nov. 2015. eSAME.

[22] J. Kienzle, W. A. Abed, and J. Klein. Aspect-oriented multi-view modeling. In K. J. Sullivan, A. Moreira, C. Schwanninger, and J. Gray, editors, *Proceedings of the 8th International Conference on Aspect-Oriented Software Development, AOSD 2009, Charlottesville, Virginia, USA, March 2-6, 2009*, pages 87–98. ACM, 2009.

[23] F. Lopes and I. Fonseca. Networked embedded systems – example applications in the educational environment. In K. Tanaka, editor, *Embedded Systems - High Performance Systems, Applications and Projects*, pages 103–128. InTech, 2012.

[24] S. Maoz, J. O. Ringert, and B. Rumpe. Semantically configurable consistency analysis for class and object diagrams. In J. Whittle, T. Clark, and T. Kühne, editors, *Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings*, volume 6981 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2011.

[25] M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, pages 10:1–10:10. IEEE, 2013.

[26] M. Pittou and S. Tripakis. Checking multi-view consistency of discrete systems with respect to periodic sampling abstractions. In *The 13th International Conference on Formal Aspects of Component Software (FACS), Besançon, France*, 2016.

[27] M. Pittou and S. Tripakis. Multi-view consistency for infinitary regular languages. In *International Conference on Embedded Computer Systems: Architectures, MOdeling and Simulation (SAMOS XVI), Samos, Greece*, 2016.

[28] A. Rajhans and B. H. Krogh. Heterogeneous verification of cyber-physical systems using behavior relations. In T. Dang and I. M. Mitchell, editors, *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*, pages 35–44. ACM, 2012.

[29] A. Rajhans and B. H. Krogh. Compositional heterogeneous abstraction. In C. Belta and F. Ivancic, editors, *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*, pages 253–262. ACM, 2013.

[30] R. Rajkumar, I. Lee, L. Sha, and J. A. Stankovic. Cyber-physical systems: the next computing revolution. In S. S. Sapatnekar, editor, *Proceedings of the 47th Design Automation Conference, DAC 2010, Anaheim, California, USA, July 13-18, 2010*, pages 731–736. ACM, 2010.

[31] J. Reineke, C. Stergiou, and S. Tripakis. Basic problems in multi-view modeling. Submitted journal version of [32]. The submitted version has been made available by its authors to the author of this thesis.

[32] J. Reineke and S. Tripakis. Basic problems in multi-view modeling. In *TACAS*, volume 8413 of *LNCS*, pages 217–232. Springer, 2014.

[33] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.

[34] R. W. Schwanke and G. E. Kaiser. Living with inconsistency in large systems. In J. F. H. Winkler, editor, *Proceedings of the International Workshop on Software Version and Configuration Control, January 27-29, 1988, Grassau, Germany*, volume 30 of *Berichte des German Chapter of the ACM*, pages 98–118. Teubner, 1988.

[35] A. A. Shah, A. A. Kerzhner, D. Schaefer, and C. J. J. Paredis. Multi-view modeling to support embedded systems engineering in sysml. In G. Engels, C. Lewerentz, W. Schäfer, A. Schürr, and B. Westfechtel, editors, *Graph Transformations and Model-Driven Engineering - Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday*, volume 5765 of *Lecture Notes in Computer Science*, pages 580–601. Springer, 2010.

[36] A. P. Sistla, M. Y. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. In *ICALP*, volume 194 of *LNCS*, pages 465–474. Springer, 1985.

[37] S. Tripakis, B. Lickly, T. A. Henzinger, and E. A. Lee. A theory of synchronous relational interfaces. *ACM Trans. Program. Lang. Syst.*, 33(4):14:1–14:41, 2011.

[38] R. von Hanxleden, E. A. Lee, C. Motika, and H. Fuhrmann. Multi-view modeling and pragmatics in 2020 - position paper on designing complex cyber-physical systems. In R. Calinescu and D. Garlan, editors, *Large-Scale Complex IT Systems. Development, Operation and Management - 17th Monterey Workshop 2012, Oxford, UK, March 19-21, 2012, Revised Selected Papers*, volume 7539 of *Lecture Notes in Computer Science*, pages 209–223. Springer, 2012.

Publication I

Maria Pittou and Stavros Tripakis. Multi-View Consistency for Infinitary Regular Languages. In *XVI International Conference on Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS 2016)*, Samos, Greece, July 2016. IEEE, 2016.

Multi-View Consistency for Infinitary Regular Languages

Maria Pittou
Aalto University

Stavros Tripakis
Aalto University and UC Berkeley

***Abstract*—Multi-view modeling is a system design methodology where different facets of a system are modeled each with a separate model, called *view*. The problem of view consistency then arises, namely, does there exist a system which could generate a given set of views? In previous work this problem has been studied for the case of discrete systems such as finite automata over finite words, on one hand, and finite automata over infinite words, on the other hand. In this work we study the problem for the case of *mixed* automata, which accept both finite and infinite words, and the corresponding infinitary regular languages. This model is particularly useful in the multi-view modeling setting, where views are obtained as projections of the system, and where these projections may turn an infinite behavior into a finite one.**

1. Introduction

Modeling provides an abstract formal description of a system under development, and it has been proved essential in system design. Obtaining a model of a system efficiently allows analysis and detection of failures, that can be resolved prior to experimental work. The construction of complex systems, and hence their modeling, is usually delegated to more than one stakeholders, in order to capture the various aspects of the systems and reduce the involved complexity. In multi-view modeling in particular, the different stakeholders obtain several models, the so called views, of the same system [2], [14].

The views can be either behavioral or structural since they may refer to the behavior or structure of the system respectively, and they are usually expressed in different semantics in order to describe the various tasks of the system. As a result, the views model only partial perspectives of the same system, and therefore the aspects not considered by each view are ignored. However, possible overlaps among the views may give rise to inconsistencies. One of the main challenges in multi-view modeling is to ensure consistency among the different views [14]. Other problems related to multi-view modeling include the construction (or *synthesis* [14]) of a complete system given its views, and also view traceability and reuse [9]. For general discussions on multi-view modeling the reader is referred to [2], [9], and for a formal treatment to [13], [14].

In this work we are interested in solving the consistency problem for the behavioral views of a system under construction. The behavior of a system can be defined in general within any global universe while the views may be obtained by some kind of transformation, like abstraction functions, to the system's behavior. [14] defines a generic multi-view modeling framework and instantiates that framework for the case of symbolic discrete systems. In [13], the behavior of a given system and of its views is described by languages in general, regular languages or ω-regular languages. Projections are used as abstraction functions to obtain the views from the system, and inverse projections for the other direction. Even though the views of a system may be defined only by regular or ω-regular languages, the application of projections and inverse projections may result in an ∞-regular language (*infinitary* regular language, i.e., a language containing both finite and infinite words) for describing the system's behavior. Moreover, it may be the case that the different views of the system can be described only by an ∞-regular language. Hence, in this paper we consider the case of describing the behavior of a system and its views by ∞-regular languages or equivalently by the relevant automata accepting such languages, which we also study in this paper.

In summary, the main contributions of this paper are three: first, we propose a model of (non-deterministic) *mixed* automata accepting infinitary languages; second, we show that mixed automata are closed under the usual set-theoretic operations (union, intersection, complementation), as well as under projection and inverse projection; third, we use these results to solve the multi-view consistency problem in the infinitary language setting.

This work has been partially supported by the Academy of Finland and the National Science Foundation (awards #1329759 and #1139138).

2. Related work

Multi-view modeling is a well known problem, related to the construction of complex systems. Existing literature mainly refers to *structural* views of a system, e.g., see [1], [4]. However, our interest is in describing a system and its views as sets of *behaviors*. Behavioral views are investigated for instance in [11], [12] within the context of cyber physical systems. In particular, the authors address the problem of heterogeneity in such complex systems in order to aid their verification, and consider behavior relations for the different semantic models used to decribe the different aspects of the same system.

Our work follows the framework proposed in [14] where the behavioral view consistency problem is defined formally. [14] provides a generic framework where both the system and views can be defined within any global universe and can be related by any kind of abstraction functions. [14] also includes a notion of conformance to capture faithfulness of a view w.r.t. a system. The problems of view consistency, orthogonality and reduction are also defined. The aforementioned can then be instantiated for versatile formalisms of a system, its views and abstraction functions. In [14] the framework is instatiated for symbolic discrete systems, while recently it has also been extended to languages and automata [13], but not infinitary languages, which is the focus of this paper.

Automata accepting a mix of finite and infinite words have been studied earlier. [10] studies *weighted* such automata for the fuzzy semiring (weighted automata form the quantitative extension of the classical finite state automata). In [3] and [7] one can find futher works for the model of [10]. Moreover, in [15] there is the definition of an (unweighted) finite state automaton that accepts both finite and infinite words, but no further results for this class of automata are presented. In this paper, we consider a definition different, albeit semantically equivalent to the one of [15], and we provide a systematic study for this class which is needed for our multi-view framework.

3. Background

3.1. Languages over finite and infinite words

Alphabet, Finite words, Infinite words: A finite alphabet Σ is a non-empty finite set of symbols. Σ^* is the set of all finite words over Σ and Σ^ω is the set of all infinite words over Σ. The set of all words over Σ is Σ^∞, i.e., $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

Languages: A $*$-language (star language) L on Σ is a set of finite words, subset of Σ^*, i.e., $L \subseteq \Sigma^*$. An ω-language (omega language) L on Σ is a set of infinite words, subset of Σ^ω, i.e., $L \subseteq \Sigma^\omega$. An ∞-language (infinitary language) L is a set of finite or infinite words, subset of Σ^∞, i.e., $L \subseteq \Sigma^\infty$.

3.2. Automata over finite and infinite words

Nondeterministic finite automaton: Let $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ be a nondeterministic finite state automaton (NFA for short) where Q_A is a finite set of states, Σ is a finite alphabet, $I_A \subseteq Q_A$ is the set of initial states, $\Delta_A \subseteq Q_A \times \Sigma \times Q_A$ is a transition function, and $F_A \subseteq Q_A$ is the set of final states. A path P_w^A of A over a finite word $w = w_0 \dots w_{n-1} \in \Sigma^*$ is a finite sequence $P_w^A\colon (q_{0A}, w_0, q_{1A}) \dots (q_{n-1A}, w_{n-1}, q_{nA})$ such that $q_{0A} \in I_A$ is the initial state and $(q_{iA}, w_i, q_{i+1A}) \in \Delta_A$ for every $0 \leq i < n$. A path P_w^A of A over a finite word $w \in \Sigma^*$ is called accepting if additionally $q_{nA} \in F_A$. A finite word $w \in \Sigma^*$ is accepted by A if there is an accepting path P_w^A of A over w. The language accepted by A, also called behavior of A, written $L(A)$, is the set of finite words accepted by A: $L(A) = \{w \in \Sigma^* \mid \exists$ accepting path P_w^A of A over $w\}$. A language L is called regular if there exists a nondeterministic finite automaton A over Σ accepting L, i.e., such that $L = L(A)$.

Example 1. *Consider the NFA A shown in Figure 1, $A = (\{q_{0A}, q_{1A}\}, \{a, b\}, \{q_{0A}\}, \Delta_A, \{q_{1A}\})$, with $\Delta_A = \{(q_{0A}, a, q_{0A}), (q_{0A}, a, q_{1A}), (q_{1A}, b, q_{1A})\}$. The language accepted by A is $L(A) = a^+b^*$, where a^+ denotes a non-empty finite sequence of the letter a and b^* denotes a (possibly empty) finite sequence of the letter b.*

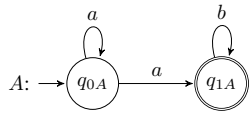


Figure 1: NFA example.

Union of NFA: Let $A^i = (Q_A^i, \Sigma, I_A^i, \Delta_A^i, F_A^i)$, for $i = 1, 2$, be two NFA. The union of A^1 and A^2 is the NFA $A^1 \cup A^2 = (Q_A, \Sigma, I_A, \Delta_A, F_A)$, with $Q_A = Q_A^1 \cup Q_A^2$, $I_A = I_A^1 \cup I_A^2$, $F_A = F_A^1 \cup F_A^2$ and the transition function $\Delta_A \subseteq Q_A \times \Sigma \times Q_A$ defined by $\Delta_A = \{(q_A, \sigma, q'_A) \mid (q_A, \sigma, q'_A) \in \Delta_A^1$ or $(q_A, \sigma, q'_A) \in \Delta_A^2\}$.

Intersection of NFA: The intersection of A^1 and A^2 is the NFA $A^1 \times A^2 = (Q_A, \Sigma, I_A, \Delta_A, F_A)$, with $Q_A = Q_A^1 \times Q_A^2$, $I_A = I_A^1 \times I_A^2$, $F_A = F_A^1 \times F_A^2$ and the transition function $\Delta_A \subseteq Q_A \times \Sigma \times$

Q_A defined by $\Delta_A = \{((q_A^1, q_A^2), \sigma, (q_A'^1, q_A'^2)) \mid (q_A^1, \sigma, q_A'^1) \in \Delta_A^1$ and $(q_A^2, \sigma, q_A'^2) \in \Delta_A^2\}$.

It is well known that $L(A^1 \cup A^2) = L(A^1) \cup L(A^2)$ and $L(A^1 \times A^2) = L(A^1) \cap L(A^2)$ [6].

Closure properties of regular languages: The class of regular languages is closed under union, intersection, and complementation [6], as well as under projections [13], which will be presented below.

Nondeterministic Büchi automaton: Let $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$ be a nondeterministic Büchi automaton (NBA for short) where Q_B is a finite set of states, Σ is a finite alphabet, $I_B \subseteq Q_B$ is the set of initial states, $\Delta_B \subseteq Q_B \times \Sigma \times Q_B$ is a transition function, and $C_B \subseteq Q_B$ is the set of final states. A path P_w^B of B over an infinite word $w = w_0 w_1 \ldots \in \Sigma^\omega$ is an infinite sequence $P_w^B\colon (q_{0B}, w_0, q_{1B})(q_{1B}, w_1, q_{2B}) \ldots$ such that $q_{0B} \in I_B$ is the initial state and $(q_{iB}, w_i, q_{i+1B}) \in \Delta_B$ for every $i \geq 0$. For every path P_w^B of B over an infinite word $w \in \Sigma^\omega$ we denote with $Inf\left(P_w^B\right)$ the set of states occurring an infinite number of times along P_w^B. Then, a path P_w^B of B over $w \in \Sigma^\omega$ is called accepting if additionally $Inf(P_w^B) \cap C_B \neq \emptyset$. An infinite word $w \in \Sigma^\omega$ is accepted by B if there is an accepting path P_w^B of B over w. The language accepted by B, also called behavior of B, written $L(B)$, is the set of infinite words accepted by B: $L(B) = \{w \in \Sigma^\omega \mid \exists$ infinite accepting path P_w^B of B over $w\}$. A language L is called ω-regular if there exists a nondeterministic Büchi automaton B over Σ accepting L, i.e., $L = L(B)$.

Example 2. *Consider the NBA of Figure 2,* $B = (\{q_{0B}, q_{1B}\}, \{a, b\}, \{q_{0B}\}, \Delta_B, \{q_{1B}\})$ *with* $\Delta_B = \{(q_{0B}, b, q_{0B}), (q_{0B}, b, q_{1B}), (q_{1B}, a, q_{0B})\}$. *Then the language accepted by* B *is* $L(B) = (b^+ab^+)^\omega$.

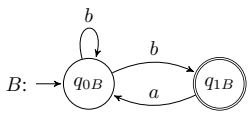


Figure 2: NBA example.

Union of NBA: Consider two NBA $B^i = (Q_B^i, \Sigma, I_B^i, \Delta_B^i, C_B^i)$ for $i = 1, 2$. The union of B^1 and B^2 is a NBA $B^1 \cup B^2 = (Q_B, \Sigma, I_B, \Delta_B, C_B)$, with $Q_B = Q_B^1 \cup Q_B^2$, $I_B = I_B^1 \cup I_B^2$, $C_B = C_B^1 \cup C_B^2$ and $\Delta_B \subseteq Q_B \times \Sigma \times Q_B$ defined by $\Delta_B = \{(q_B, \sigma, q_B') \mid (q_B, \sigma, q_B') \in \Delta_B^1$ or $(q_B, \sigma, q_B') \in \Delta_B^2\}$.

Intersection of NBA: The intersection of B^1 and B^2 is the NBA $B^1 \times B^2 = (Q_B, \Sigma, I_B, \Delta_B, C_B)$, with $Q_B = Q_B^1 \times Q_B^2 \times \{1, 2\}$, $I_B = I_B^1 \times I_B^2 \times \{1\}$, $C_B = Q_B^1 \times C_B^2 \times \{2\}$ and $\Delta_B \subseteq Q_B \times \Sigma \times Q_B$ is defined by $\Delta_B = \{((q_B^1, q_B^2, 1), \sigma, (q_B'^1, q_B'^2, j)) \mid (q_B^1, \sigma, q_B'^1) \in \Delta_B^1$ and $(q_B^2, \sigma, q_B'^2) \in \Delta_B^2$ and if $q_B^1 \in C_B^1$, $j = 2$ else $j = 1\} \cup \{((q_B^1, q_B^2, 2), \sigma, (q_B'^1, q_B'^2, j)) \mid (q_B^1, \sigma, q_B'^1) \in \Delta_B^1$ and $(q_B^2, \sigma, q_B'^2) \in \Delta_B^2$ and if $q_B^2 \in C_B^2$ $j = 1$ else $j = 2\}$.

Then, it can be proved that $L(B^1 \cup B^2) = L(B^1) \cup L(B^2)$ and $L(B^1 \times B^2) = L(B^1) \cap L(B^2)$.

Closure properties of ω-regular languages: The class of ω-regular languages is closed under union, intersection, complementation [8], infinite projection, and inverse projection [13].

3.3. Projections and inverse projections

Projection of words and languages: Consider two alphabets Σ and Σ' such that $\Sigma' \subseteq \Sigma$. The projection of a word $w \in \Sigma^\infty$ onto the subalphabet Σ', is performed by the function $\Pi_{\Sigma \to \Sigma'} : \Sigma^\infty \to \Sigma'^\infty$ defined as follows (where $\cdot$ denotes word concatenation):

$$\Pi_{\Sigma \to \Sigma'}(w) = \begin{cases} \epsilon & \text{if } w = \epsilon \\ \sigma \cdot \Pi_{\Sigma \to \Sigma'}(u) & \text{if } w = \sigma \cdot u \text{ and } \sigma \in \Sigma' \\ \Pi_{\Sigma \to \Sigma'}(u) & \text{if } w = \sigma \cdot u \text{ and } \sigma \notin \Sigma' \end{cases}$$

The projection of a finite word is always a finite word while the projection of an infinite word may be either a finite or infinite word. For example, if $\Sigma = \{a, b\}$ and $\Sigma' = \{b\}$, then for $w^1 = abab \in \Sigma^*$ we have $\Pi_{\Sigma \to \Sigma'}(w^1) = bb \in \Sigma'^*$ while for $w^2 = ba^\omega \in \Sigma^\omega$ we have $\Pi_{\Sigma \to \Sigma'}(w^2) = b \in \Sigma'^*$.
The projection of a language $L \subseteq \Sigma^\infty$ is defined as $\Pi_{\Sigma \to \Sigma'}(L) = \{\Pi_{\Sigma \to \Sigma'}(w) \mid w \in L\}$. The projection of a $*$-language is always a $*$-language, while the projection of an ω-language is generally an ∞-language. For example, if $\Sigma = \{b, c\}$ and $\Sigma' = \{b\}$, then for $L^1 = cb^*c \subseteq \Sigma^*$, we obtain that $\Pi_{\Sigma \to \Sigma'}(L^1) = b^* \subseteq \Sigma'^*$, while for $L^2 = b^*c^\omega \cup c^*b^\omega \subseteq \Sigma^\omega$, we obtain that $\Pi_{\Sigma \to \Sigma'}(L^2) = b^* \cup b^\omega \subseteq \Sigma'^\infty$.

Inverse projection of words and languages: Consider two alphabets Σ and Σ' such that $\Sigma' \supseteq \Sigma$. We define the inverse projection of $w \in \Sigma^\infty$ onto Σ' as the set $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(w) = \{u$ over $\Sigma' \mid \Pi_{\Sigma' \to \Sigma}(u) = w\}$. The inverse projection of a finite word can be either a finite or infinite word, while the inverse projection of an infinite word is always an infinite word. For example, if $\Sigma = \{b\}$ and $\Sigma' = \{a, b\}$, then for $w^1 = b \in \Sigma^*$ we obtain that $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(w^1) = a^*b(a^* \cup a^\omega) \subseteq \Sigma'^\infty$, and for $w^2 = b^\omega \in \Sigma^\omega$ we obtain that $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(w^2) = a^*(a^*ba^*)^\omega \subseteq \Sigma'^\omega$.

Moreover, the inverse projection of a language $L \subseteq \Sigma^\infty$ is defined as $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L) = \{w$ over $\Sigma' \mid \Pi_{\Sigma' \to \Sigma}(w) \in L\}$. The inverse projection of an ω-language is always an ω-language, while

the inverse projection of a $*$-language is generally an ∞-language. For instance, if $\Sigma = \{b\}$ and $\Sigma' = \{b, c\}$, then for $L^1 = b^* \subseteq \Sigma^*$, we obtain that $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L^1) = c^* b^* (c^* \cup c^\omega) \subseteq \Sigma'^\infty$, while for $L^2 = (bb)^\omega \subseteq \Sigma^\omega$, we obtain that $\Pi_{\Sigma' \leftarrow \Sigma}(L^2) = c^*(c^*bc^*bc^*)^\omega \subseteq \Sigma'^\omega$.

More examples of projections and inverse projections of languages are given below where we discuss how these operations are implemented on automata.

Lemma 1. *1. The projection of the inverse projection of a language yields the original language, i.e.,: for every language $L \subseteq \Sigma^\infty$ and $\Sigma \subseteq \Sigma'$ it holds that $\Pi_{\Sigma' \to \Sigma}(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L)) = L$.*

2. The inverse projection of the projection of a language generally yields a superset of the original language: for every language $L \subseteq \Sigma^\infty$ and $\Sigma' \subseteq \Sigma$ it holds that $\Pi^{-1}_{\Sigma \leftarrow \Sigma'}(\Pi_{\Sigma \to \Sigma'}(L)) \supseteq L$.

Next, we show how the projection and inverse projection operations can be implemented on NFA and NBA. We omit the details and correctness proofs of these constructions, which can be found on [13], and we rather illustrate the results with some examples.

Projection of NFA: Consider two alphabets Σ and Σ' such that $\Sigma' \subseteq \Sigma$, and a NFA $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ over Σ. The projection of A on Σ' denoted by $\Pi_{\Sigma \to \Sigma'}(A)$, is a NFA $(Q_A, \Sigma', I_A, \Delta'_A, F_A)$ with $\Delta'_A = \{(q_A, \sigma, q'_A) \mid \sigma \in \Sigma' \text{ and } (q_A, \sigma, q'_A) \in \Delta_A\} \cup \{(q_A, \epsilon, q'_A) \mid \sigma \in \Sigma \setminus \Sigma' \text{ and } (q_A, \sigma, q'_A) \in \Delta_A\}$. An example of NFA projection is shown in Figure 3.

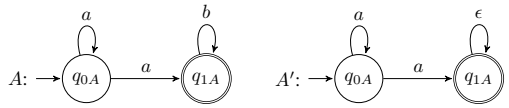


Figure 3: Example of NFA projection. NFA A over $\Sigma = \{a, b\}$ and NFA $A' = \Pi_{\Sigma \to \{a\}}(A)$.

Projection of NBA: Let $\Sigma' \subseteq \Sigma$ and consider a NBA $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$ over Σ. Since the projection of a ω-language can contain both finite and infinite words, two kinds of projections are obtained, the NBA finite and infinite projection. The NBA finite projection of B on Σ', denoted by $\Pi^*_{\Sigma \to \Sigma'}(B)$, is a NFA $(Q_B, \Sigma', I_B, \bar{\Delta}_B, \bar{F}_B)$ where $\bar{\Delta}_B = \{(q_B, \sigma, q'_B) \mid \sigma \in \Sigma' \text{ and } (q_B, \sigma, q'_B) \in \Delta_B\} \cup \{(q_B, \epsilon, q'_B) \mid \sigma \in \Sigma \setminus \Sigma' \text{ and } (q_B, \sigma, q'_B) \in \Delta_B\}$, and $\bar{F}_B = \{q_B \mid q_B \in C_B \text{ and } \exists w \in (\Sigma \setminus \Sigma')^+ : (q_B, \sigma, q_B) \in \Delta^*_B\}$, where Δ^*_B denotes the reflexive and transitive closure of Δ_B. The NBA infinite projection of B on Σ', denoted by $\Pi^\omega_{\Sigma \to \Sigma'}(B)$, is a NBA $(Q_B, \Sigma', I_B, \Delta'_B, C_B)$ with $\Delta'_B = \{(q_B, \sigma, q'_B) \mid \sigma \in \Sigma' \text{ and } \exists w \in \Sigma^* : \Pi_{\Sigma \to \Sigma'}(w) = \sigma \text{ and } (q_B, w, q'_B) \in \Delta^*_B\}$. An example of NBA projection is shown in Figure 4.

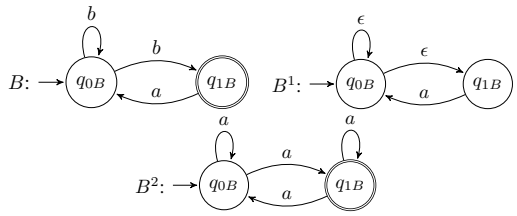


Figure 4: Example of NBA projection. NBA B over $\Sigma = \{a, b\}$, NFA $B^1 = \Pi^*_{\Sigma \to \{a\}}(B)$, and NBA $B^2 = \Pi^\omega_{\Sigma \to \{a\}}(B)$.

NFA inverse projection: Let $\Sigma' \supseteq \Sigma$ and consider a NFA $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ over Σ. Since the inverse projection of a $*$-language can contain both finite and infinite words, we consider the relevant $*$-part and ω-part of the inverse projection of A. In particular, the former is denoted by $\Pi^{-1,*}_{\Sigma' \leftarrow \Sigma}(A)$ and is a NFA $(Q_A, \Sigma', \bar{\Delta}_A, I_A, F_A)$, where $\bar{\Delta}_A = \{(q_A, \sigma, q'_A) \mid \sigma \in \Sigma \text{ and } (q_A, \sigma, q'_A) \in \Delta_A\} \cup \{(q_A, \sigma, q_A) \mid q_A \in Q_A \text{ and } \sigma \in \Sigma' \setminus \Sigma\}$. The infinite, ω-part of the inverse projection is denoted by $\Pi^{-1,\omega}_{\Sigma' \leftarrow \Sigma}(A)$ and is a NBA $(Q_A \cup \{q_{\omega A}\}, \Sigma', I_A, \Delta'_A, \{q_{\omega A}\})$ where $\Delta'_A = \{(q_A, \sigma, q'_A) \mid \sigma \in \Sigma \text{ and } (q_A, \sigma, q'_A) \in \Delta_A\} \cup \{(q_A, \sigma, q_A) \mid q_A \in Q_A \text{ and } \sigma \in \Sigma' \setminus \Sigma\} \cup \{(q_A, \sigma, q_{\omega A}) \mid q_A \in F_A \text{ and } \sigma \in \Sigma' \setminus \Sigma\} \cup \{(q_{\omega A}, \sigma, q_{\omega A}) \mid \sigma \in \Sigma' \setminus \Sigma\}$. An example of NFA inverse projection is shown in Figure 5.

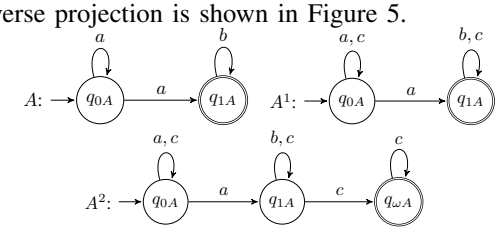


Figure 5: Example of NFA inverse projection. NFA A over $\Sigma = \{a, b\}$. NFA $A^1 = \Pi^{-1,*}_{\{a,b,c\} \leftarrow \Sigma}(A)$, and NBA $A^2 = \Pi^{-1,\omega}_{\{a,b,c\} \leftarrow \Sigma}(A)$.

NBA inverse projection: Let $\Sigma' \supseteq \Sigma$ and consider a NBA $A = (Q_B, \Sigma, I_B, \Delta_B, C_B)$ over Σ. The inverse projection of B on Σ' denoted by $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(B)$, is a NBA $(Q'_B, \Sigma', I_B, \Delta'_B, C_B)$ where $Q'_B = Q_B \cup \{\hat{q_B} \mid q_B \in C_B\}$ and $\Delta'_B = \Delta_B \cup \{(\hat{q_B}, \sigma, q'_B) \mid q_B \in C_B \text{ and } (q_B, \sigma, q'_B) \in \Delta_B\} \cup \{(q_B, \sigma, \hat{q_B}) \mid q_B \in C_B \text{ and } \sigma \in \Sigma' \setminus \Sigma\} \cup \{(\hat{q_B}, \sigma, \hat{q_B}) \mid q_B \in C_B \text{ and } \sigma \in \Sigma' \setminus \Sigma\} \cup \{(q_B, \sigma, q_B) \mid q_B \in Q_B \setminus C_B \text{ and } \sigma \in \Sigma' \setminus \Sigma\}$. An example of NBA inverse projection is shown in Figure 6.

3.4. Multi-view modeling

For our multi-view modeling framework we consider a system and its views as sets of behaviors [14].

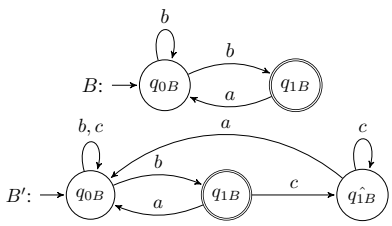


Figure 6: Example of NBA inverse projection. NBA B over $\Sigma = \{a, b\}$ and NBA $B' = \Pi^{-1}_{\{a,b,c\}\leftarrow\Sigma}(B)$.

Formally, *a system $\mathcal{S}$ over a domain of behaviors $\mathcal{U}$, is a subset of $\mathcal{U}$*: $\mathcal{S} \subseteq \mathcal{U}$. A view is intuitively an incomplete picture of a system, and may be obtained by some kind of transformation of the system behaviors into (incomplete) behaviors in another domain. Following [14], such a transformation is defined by means of an *abstraction function* $a : \mathcal{U} \to \mathcal{D}$, where $\mathcal{D}$ is the *view domain*. A *view $\mathcal{V}$ over view domain $\mathcal{D}$, is a subset of $\mathcal{D}$*: $\mathcal{V} \subseteq \mathcal{D}$.

Let us now state the multi-view consistency problem as defined in [14]. A set of views $\mathcal{V}_1, \ldots, \mathcal{V}_n$ over view domains $\mathcal{D}_1, \ldots, \mathcal{D}_n$ respectively, are *consistent with respect to a set of abstraction functions* $a_1, ..., a_n$, if there exists a system $\mathcal{S}$ over $\mathcal{U}$ so that $\mathcal{V}_i = a_i(\mathcal{S})$, for all $i = 1, ..., n$. We call such a system $\mathcal{S}$ a *witness system* to the consistency of $\mathcal{V}_1, \ldots, \mathcal{V}_n$. Obviously, if there is no such system, then we conclude that the views are inconsistent.

We consider the setting where a system and its views are described by languages over finite and infinite words. Specifically, two languages L^1, L^2 (describing two views) on alphabets Σ^1, Σ^2, respectively, are consistent if there exists language L on some alphabet Σ such that $\Pi_{\Sigma\to\Sigma^i}(L) = L^i$, for $i = 1, 2$. It is not necessary that the alphabet Σ is known, however, it should be a superset of $\Sigma^1 \cup \Sigma^2$. It is in fact proved in [13] that it suffices to consider only $\Sigma = \Sigma^1 \cup \Sigma^2$, and to check whether $L^\sharp = \Pi^{-1}_{\Sigma^1\cup\Sigma^2\leftarrow\Sigma^1}(L^1) \cap \Pi^{-1}_{\Sigma^1\cup\Sigma^2\leftarrow\Sigma^2}(L^2)$ is a valid witness:

Theorem 1. *[13] Let L^1 and L^2 be two languages on alphabets Σ^1 and Σ^2 respectively, and let $L^\sharp = \Pi^{-1}_{\Sigma^1\cup\Sigma^2\leftarrow\Sigma^1}(L^1) \cap \Pi^{-1}_{\Sigma^1\cup\Sigma^2\leftarrow\Sigma^2}(L^2)$. Then:*

1) *L^1 and L^2 are consistent if and only if there exists language L on alphabet $\Sigma^1 \cup \Sigma^2$, such that $\Pi_{\Sigma^1\cup\Sigma^2\to\Sigma^i}(L) = L^i$, for $i = 1, 2$.*
2) *Let L be a language on $\Sigma^1 \cup \Sigma^2$ that is a witness to the consistency of L^1 and L^2. Then: (a) If both L^1 and L^2 are $*$-languages, then L is also a $*$-language, i.e., $L \subseteq \Sigma^*$. (b) If both L^1 and L^2 are ω-languages, then L is also a ω-language, i.e., $L \subseteq \Sigma^\omega$.*
3) *L^1 and L^2 are consistent if and only if $\Pi_{\Sigma^1\cup\Sigma^2\to\Sigma^i}(L^\sharp) \supseteq L^i$, for $i = 1, 2$.*
4) *If L^1 and L^2 are consistent then $L^\sharp$ is a witness to their consistency, i.e., $\Pi_{\Sigma^1\cup\Sigma^2\to\Sigma^i}(L^\sharp) = L^i$, for $i = 1, 2$. Moreover, $L^\sharp$ is the greatest such witness, that is, any other witness L is such that $L \subseteq L^\sharp$.*

4. Multi-view consistency for ∞-regular languages

In this paper, we define the behavior of a system and its views by ∞-regular languages, that is, by ∞-languages accepted by a type of finite automata that we call *mixed* automata, and which will be defined below. We consider projections as abstraction functions, and we propose algorithmic solutions to the view consistency problem in this setting.

4.1. Mixed automata and ∞-regular languages

In order to solve the consistency problem for ∞-languages, we need the notion of automata that can accept an infinitary language, i.e., a language that may contain both finite and infinite words. As we have already mentioned in Section 2, such automata have been considered in the literature, but a systematic study has been lacking. [15] defines an automaton with two accepting conditions, so that it can accept both finite and infinite words. However, [15] provides no further results for this type of automata, apart from their definition. In this paper, we propose an even more intuitive definition of an automaton accepting an infinitary language, the so called *mixed* automaton, which is a *pair* of a finite automaton and an ω-automaton. We fix the ω-automaton to be a Büchi automaton, and we consider in particular nondeterministic mixed automata.

One should observe that the notion of a pair automaton is equivalent with the definition of a single automaton with two accepting sets [15]. Indeed, from the latter model one can obtain a pair automaton by making two distinct copies of the single automaton, and associating the appropriate accepting set of each element of the pair. Vice-versa, from a mixed automaton, i.e., from a pair automaton, one can construct a single automaton by taking the product of the two automata in the pair.

Nondeterministic Mixed Automaton: Let Σ denote a finite alphabet. A nondeterministic mixed automaton (NXA for short) over the alphabet Σ is defined as a pair $M = (A, B)$ where $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ is a nondeterministic finite state automaton and $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$ is a

nondeterministic Büchi automaton, with $Q_A \cap Q_B = \emptyset$. The language $L(M)$ of the mixed automaton M is defined by $L(M) = L(A) \cup L(B) = \{w \in \Sigma^* \mid w$ is accepted by $A\} \cup \{w \in \Sigma^\omega \mid w$ is accepted by $B\} = \{w \in \Sigma^\infty \mid w$ is accepted by A or $B\}$. It should be clear that this is disjoint union since A accepts only finite words over Σ and B accepts only infinite words over Σ. A language $L \subseteq \Sigma^\infty$ is called ∞-regular, if there exists a mixed automaton M such that $L(M) = L$. Moreover, every NXA M over Σ can be considered either as nondeterministic finite automaton whenever $C_B = \emptyset$, or as a nondeterministic Büchi automaton, whenever $F_A = \emptyset$.

Example 3. *Consider the NXA* $M = (A, B)$ *where* $A = (\{q_{0A}, q_{1A}\}, \{a, b\}, \{q_{0A}\}, \Delta_A, \{q_{1A}\})$ *with* $\Delta_A = \{(q_{0A}, a, q_{0A}), (q_{0A}, a, q_{1A}), (q_{1A}, b, q_{1A})\}$, *and* $B = (\{q_{0B}, q_{1B}\}, \{a, b\}, \{q_{0B}\}, \Delta_B, \{q_{1B}\})$ *with* $\Delta_B = \{(q_{0B}, b, q_{0B}), (q_{0B}, b, q_{1B}), (q_{1B}, a, q_{0B})\}$, *as shown in Figure 7. Then the language accepted by* M *is* $L(M) = L(A) \cup L(B) = a^+b^* \cup (b^+ab^+)^\omega$.

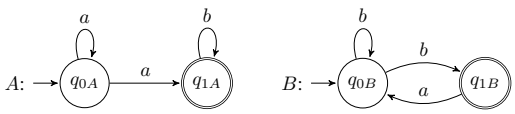


Figure 7: NXA example with NFA A and NBA B.

Throughout the paper, when we refer to mixed automata we mean nondeterministic mixed automata.

4.2. Closure properties of ∞-regular languages

We now show that the class of ∞-regular languages is closed under union, intersection, complementation, projection, and inverse projection.

Union of mixed automata: Consider two NXA $M^i = (A^i, B^i)$ where $A^i = (Q^i_A, \Sigma, I^i_A, \Delta^i_A, F^i_A)$ and $B^i = (Q^i_B, \Sigma, I^i_B, \Delta^i_B, C^i_B)$, for $i = 1, 2$ and $(Q^1_A \cup Q^1_B) \cap (Q^2_A \cup Q^2_B) = \emptyset$. The union of M^1 and M^2 is a NXA $M = (A, B)$ where: $A = A^1 \cup A^2 = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ and $B = B^1 \cup B^2 = (Q_B, \Sigma, I_B, \Delta_B, C_B)$.

Proposition 1. *The class of ∞-regular languages is closed under union.*

Proof. One can observe that $L(M) = L(A) \cup L(B) = (L(A^1) \cup L(A^2)) \cup (L(B^1) \cup L(B^2)) = (L(A^1) \cup L(B^1)) \cup (L(A^2) \cup L(B^2)) = L(M^1) \cup L(M^2)$, which completes our proof. □

Intersection of mixed automata: Consider two NXA $M^i = (A^i, B^i)$ where $A^i = (Q^i_A, \Sigma, I^i_A, \Delta^i_A, F^i_A)$ and $B^i = (Q^i_B, \Sigma, I^i_B, \Delta^i_B, C^i_B)$, for $i = 1, 2$ and $(Q^1_A \cup Q^1_B) \cap (Q^2_A \cup Q^2_B) = \emptyset$. The intersection of M^1 and M^2 is a NXA $M = (A, B)$ where: $A = A^1 \times A^2 = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ and $B = B^1 \times B^2 = (Q_B, \Sigma, I_B, \Delta_B, C_B)$.

Proposition 2. *The class of ∞-regular languages is closed under intersection.*

Proof. We observe that $L(M^1) \cap L(M^2) = (L(A^1) \cap L(A^2)) \cup (L(A^1) \cap L(B^2)) \cup (L(B^1) \cap L(A^2)) \cup (L(B^1) \cap L(B^2)) = (L(A^1) \cap L(A^2)) \cup (L(B^1) \cap L(B^2))$. Moreover, by construction of B, we have that $L(M) = (L(A^1) \cap L(A^2)) \cup (L(B^1) \cap L(B^2)) = L(M^1) \cap L(M^2)$, which completes our proof. □

Complementation of mixed automata: Consider a NXA $M = (A, B)$ where $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ and $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$, and $Q_A \cap Q_B = \emptyset$. The complement of M is a NXA $M^c = (A^c, B^c)$, where A^c is the complement automaton of the NFA A and B^c is the complement automaton of the NBA B. (One can find the relevant constructions for A^c and B^c in [6] and [17] respectively).

Proposition 3. *The class of ∞-regular languages is closed under complementation.*

Proof. We have that $L(M^c) = L(A^c) \cup L(B^c) = (\Sigma^* \setminus L(A)) \cup (\Sigma^\omega \setminus L(B)) = \Sigma^\infty \setminus (L(A) \cup L(B)) = \Sigma^\infty \setminus L(M)$, and our proof is completed. □

Proposition 4. *The emptiness problem of an ∞-regular language is decidable.*

Proof. The proof can be obtained by decidability of the emptiness problem for both regular and ω-regular languages. □

Proposition 5. *The equality problem of two ∞-regular languages is decidable.*

Proof. The proof can be obtained by decidability of the equality problem for both regular and ω-regular languages. □

4.3. Projections and inverse projections on mixed automata

In the sequel, we consider the closure of ∞-regular languages over projections and inverse projections. For this, we need to obtain the constructions of these operations on NXA, and we do so by using the relevant constructions for NFA and NBA as defined in 3.3. It should be clear that since mixed automata accept ∞-regular languages, both

the projection and inverse projection of an ∞-regular language is always an ∞-regular language.

Projection of NXA: Let Σ, Σ' denote two finite alphabets such that $\Sigma' \subseteq \Sigma$ and consider the NXA $M = (A, B)$ where $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ and $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$, with $Q_A \cap Q_B = \emptyset$. The projection of M on Σ', denoted by $\Pi_{\Sigma \to \Sigma'}(M)$ is a NXA $M' = (A', B')$ over Σ', where $A' = A^1 \cup A^2$, $A^1 = \Pi_{\Sigma \to \Sigma'}(A)$ and $A^2 = \Pi^*_{\Sigma \to \Sigma'}(B)$, and $B' = \Pi^\omega_{\Sigma \to \Sigma'}(B)$. An example of NXA projection is shown in Figure 8.

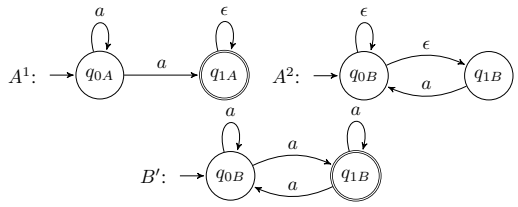


Figure 8: Example of NXA projection $\Pi_{\Sigma \to \{a\}}(M)$, where M is the NXA of the Example 3.

Proposition 6. *The class of ∞-regular languages is closed under projections.*

Proof. By [13], it holds that $L\left(\Pi_{\Sigma \to \Sigma'}(A)\right) = \Pi_{\Sigma \to \Sigma'}(L(A))$, $L\left(\Pi^*_{\Sigma \to \Sigma'}(B)\right) = \Pi_{\Sigma \to \Sigma'}(L(B)) \cap \Sigma'^*$, and $L\left(\Pi^\omega_{\Sigma \to \Sigma'}(B)\right) = \Pi_{\Sigma \to \Sigma'}(L(B)) \cap \Sigma'^\omega$. Then, $L(M') = L\left(\Pi_{\Sigma \to \Sigma'}(M)\right) = L(A') \cup L(B') = \left(L(A^1 \cup A^2)\right) \cup L(B') = L(A^1) \cup L(A^2) \cup L(B') = L\left(\Pi_{\Sigma \to \Sigma'}(A)\right) \cup L\left(\Pi^*_{\Sigma \to \Sigma'}(B)\right) \cup L\left(\Pi^\omega_{\Sigma \to \Sigma'}(B)\right) = \Pi_{\Sigma \to \Sigma'}(L(A)) \cup (\Pi_{\Sigma \to \Sigma'}(L(B)) \cap \Sigma'^*) \cup (\Pi_{\Sigma \to \Sigma'}(L(B)) \cap \Sigma'^\omega) = \Pi_{\Sigma \to \Sigma'}(L(M))$. $\square$

NXA inverse projection: Let Σ, Σ' denote two finite alphabets such that $\Sigma' \supseteq \Sigma$ and consider the NXA $M = (A, B)$ where $A = (Q_A, \Sigma, I_A, \Delta_A, F_A)$ and $B = (Q_B, \Sigma, I_B, \Delta_B, C_B)$, with $Q_A \cap Q_B = \emptyset$. The inverse projection of M on Σ', denoted by $\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(M)$ is a NXA $M' = (A', B')$ over Σ' such that $A' = \Pi^{-1,*}_{\Sigma' \leftarrow \Sigma}(A)$, and $B' = B^1 \cup B^2$ where $B^1 = \Pi^{-1,\omega}_{\Sigma' \leftarrow \Sigma}(A)$ and $B^2 = \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(B)$. An example of NXA inverse projection is shown in Figure 9.

Proposition 7. *The class of ∞-regular languages is closed under inverse projections.*

Proof. By [13], it holds that $L\left(\Pi^{-1,*}_{\Sigma' \leftarrow \Sigma}(A)\right) = \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(A)) \cap \Sigma'^*$, $L\left(\Pi^{-1,\omega}_{\Sigma' \leftarrow \Sigma}(A)\right) = \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(A)) \cap \Sigma'^\omega$, and $L\left(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(B)\right) = \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(B))$. Hence, $L(M') = L\left(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(M)\right) = L(A') \cup L(B') = L(A') \cup \left(L(B^1 \cup B^2)\right) = L(A') \cup L(B^1) \cup L(B^2) = L\left(\Pi^{-1,*}_{\Sigma' \leftarrow \Sigma}(A)\right) \cup L\left(\Pi^{-1,\omega}_{\Sigma' \leftarrow \Sigma}(A)\right) \cup L\left(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(B)\right) = \left(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(A)) \cap \Sigma'^*\right) \cup \left(\Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(A)) \cap \Sigma'^\omega\right) \cup \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(B)) = \Pi^{-1}_{\Sigma' \leftarrow \Sigma}(L(M))$. $\square$

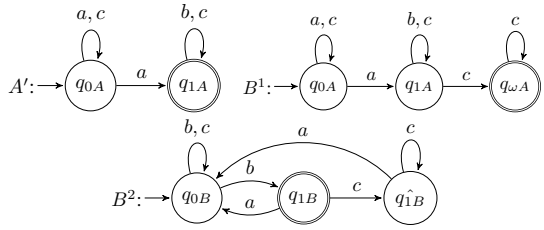


Figure 9: Example of NXA inverse projection $\Pi^{-1}_{\{a,b,c\} \leftarrow \Sigma}(M)$, where M is the NXA of the Example 3.

4.4. Solution to the multi-view consistency problem

Checking consistency for infinitary languages: We consider the setting discussed in 3.4 and we solve the view consistency problem, given that the views of a certain system are described by ∞-regular languages. We obtain two variants of the problem, and then we prove that they are equivalent:

Problem 1: Consider two NXA $M^i = \left(A^i, B^i\right)$ over the alphabet Σ^i for $i = 1, 2$ respectively. Check whether $L(M^1)$ and $L(M^2)$ are consistent, i.e., whether there exists an ∞-language L over the alphabet $\Sigma = \Sigma^1 \cup \Sigma^2$ such that $\Pi_{\Sigma^1 \cup \Sigma^2 \to \Sigma^1}(L) = L\left(M^1\right)$ and $\Pi_{\Sigma^1 \cup \Sigma^2 \to \Sigma^2}(L) = L\left(M^2\right)$.

Problem 2: Consider two NXA $M^i = \left(A^i, B^i\right)$ over the alphabet Σ^i for $i = 1, 2$ respectively. Check whether there exists a NXA M over the alphabet $\Sigma = \Sigma^1 \cup \Sigma^2$ such that $\Pi_{\Sigma^1 \cup \Sigma^2 \to \Sigma^1}(L\left(M\right)) = L\left(M^1\right)$ and $\Pi_{\Sigma^1 \cup \Sigma^2 \to \Sigma^2}(L\left(M\right)) = L\left(M^2\right)$.

Theorem 2. *There is a solution to Problem 1 if and only if there is a solution to Problem 2.*

Proof. The *if* part is trivial. For the *only if* part, we assume that $L\left(M^1\right)$ and $L\left(M^2\right)$ are consistent. Then by Theorem 1, the language $L^\sharp = \Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^1}(L(M^1)) \cap \Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^2}(L(M^2))$ is a witness to the consistency of $L\left(M^1\right)$ and $L\left(M^2\right)$. It suffices to obtain a NXA that accepts $L^\sharp$. It holds, by Proposition 7, that $\Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^i}(L(M^i)) = L(\Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^i}(M^i))$, for $i = 1, 2$. Moreover, by Proposition 2, we have that $L^\sharp = L(\Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^1}(M^1)) \cap L(\Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^2}(M^2)) = L(M^\sharp)$, where $M^\sharp = \Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^1}(M^1) \times \Pi^{-1}_{\Sigma^1 \cup \Sigma^2 \leftarrow \Sigma^2}(M^2)$, and the NXA $M^\sharp$ is a solution to Problem 2. $\square$

Theorem 3. *Problems 1 and 2 are PSPACE-complete.*

Proof. Since Problems 1 and 2 are equivalent, it suffices to consider only Problem 1. By part 3 of Theorem 1, by Theorem 2, and by Proposition 6, $L(M^1)$ and $L(M^2)$ are consistent iff $L(\Pi_{\Sigma^1\cup\Sigma^2\to\Sigma^i}(M^\sharp)) \supseteq L(M^i)$ for $i = 1, 2$. $\Pi_{\Sigma^1\cup\Sigma^2\to\Sigma^i}(M^\sharp)$ can be computed in polynomial time, and for checking language containment of NXA, one has to check language containment of the NFA and NBA parts, which are both PSPACE-complete. Therefore, checking consistency of $L(M^1)$ and $L(M^2)$ is in PSPACE. Moreover, since NFA and NBA language equivalence are both PSPACE-hard [5], [16], we obtain that NXA language equivalence is PSPACE-hard.

For PSPACE hardness of Problem 1, we prove that NFA language equivalence, which is PSPACE-complete [5], is reducible to Problem 1. Let A^1, A^2 be two NFA over the same alphabet Σ. The NFA language equivalence problem is to check whether $L(A^1) = L(A^2)$. We let $\Sigma^1 = \Sigma^2 = \Sigma$ and define two NXA $M^1 = (A^1, B_\emptyset)$ and $M^2 = (A^2, B_\emptyset)$ by setting their NBA parts to be a NBA $B_\emptyset$ accepting the empty language. Then $L(M^i) = L(A^i)$, for $i = 1, 2$. We claim that $L(A^1) = L(A^2)$ iff $L(M^1)$ and $L(M^2)$ are consistent. Assume that $L(A^1) = L(A^2)$, i.e., $L(M^1) = L(M^2) = L$. Then, $\Pi_{\Sigma\cup\Sigma\to\Sigma}(L) = L(M^1)$ and $\Pi_{\Sigma\cup\Sigma\to\Sigma}(L) = L(M^2)$. Hence, $L(M^1)$ and $L(M^2)$ are consistent. Conversely, now assume that $L(M^1)$ and $L(M^2)$ are consistent. Then, by part 1 of Theorem 1 there is a language L over Σ, such that $L = \Pi_{\Sigma\cup\Sigma\to\Sigma}(L) = L(M^1)$ and $L = \Pi_{\Sigma\cup\Sigma\to\Sigma}(L) = L(M^2)$, which implies that $L(M^1) = L(M^2)$, thus $L(A^1) = L(A^2)$.

□

5. Conclusions and future work

One of the main challenges in multi-view modeling, where different models (views) are used to represent different facets of the same system, is to ensure that the views are consistent. In this work we solved the consistency problem for behavioral views defined by ∞-regular languages. As a special case, our solution also implies the solution of the consistency problem in the case where some of the views are regular languages while some others are ω-regular languages, an open problem from [13]. Indeed, both a regular and an ω-regular language is an ∞-regular language, and in the positive cases of the consistency problem the behavior of the witness system is also defined as an ∞-regular language, i.e., can be described by the behavior of a nondeterministic mixed automaton.

Possible directions for future work would be to develop the multi-view modeling for different frameworks and solve the consistency problem. Moreover, one can consider other abstraction functions apart from projections for obtaining the views of a system or in general other types of transformations.

References

[1] A. Bhave, B. H. Krogh, D. Garlan, and B. R. Schmerl. View consistency in architectures for cyber-physical systems. In *ICCPS*, pages 151–160, 2011.

[2] D. Broman, E. Lee, S. Tripakis, and M. Törngren. Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems. In *6th International Workshop on Multi-Paradigm Modeling (MPM'12)*, 2012.

[3] M. Droste and D. Kuske. Skew and infinitary formal power series. *Theor. Comput. Sci.*, 366(3):199–227, 2006.

[4] A. Finkelstein, D. M. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multperspective specifications. *IEEE Trans. Soft. Eng.*, 20(8):569–578, 1994.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability*. W. H. Freeman, 1979.

[6] J. E. Hopcroft and J. D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. 1990.

[7] W. Kuich and G. Rahonis. Fuzzy regular languages over finite and infinite words. *Fuzzy Sets and Systems*, 157(11):1532–1549, 2006.

[8] D. Perrin and J.-E. Pin. *Infinite words : automata, semigroups, logic and games*. Elsevier, 2004.

[9] M. Persson, M. Törngren, A. Qamar, J. Westman, M. Biehl, S. Tripakis, H. Vangheluwe, and J. Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *Embedded Software, EMSOFT*, 2013.

[10] G. Rahonis. Infinite fuzzy computations. *Fuzzy Sets and Systems*, 153(2):275–288, 2005.

[11] A. Rajhans and B. H. Krogh. Heterogeneous verification of cyber-physical systems using behavior relations. In *HSCC*, pages 35–44, 2012.

[12] A. Rajhans and B. H. Krogh. Compositional heterogeneous abstraction. In *Hybrid Systems: Computation and Control, HSCC'13*, pages 253–262, 2013.

[13] J. Reineke, C. Stergiou, and S. Tripakis. Basic problems in multi-view modeling. Submitted journal version of [14].

[14] J. Reineke and S. Tripakis. Basic problems in multi-view modeling. In *Tools and Algorithms for the Construction and Analysis of Systems - TACAS*, volume 8413 of *LNCS*, pages 217–232. Springer, 2014.

[15] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier and MIT Press, 2001.

[16] A. P. Sistla, M. Y. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. In *ICALP*, volume 194 of *LNCS*, pages 465–474. Springer, 1985.

[17] M. Tsai, S. Fogarty, M. Y. Vardi, and Y. Tsay. State of Büchi Complementation. *Logical Methods in Computer Science*, 10(4), 2014.

Publication II

Maria Pittou and Stavros Tripakis. Checking multi-view consistency of discrete systems with respect to periodic sampling abstractions. In *13th International Conference on Formal Aspects of Component Software (FACS 2016)*, Besançon, France, October 2016, *Lecture notes in Computer Science*. Springer, 2016.

Checking multi-view consistency of discrete systems with respect to periodic sampling abstractions*

Maria Pittou[1] and Stavros Tripakis[1,2]

[1] Aalto University, Finland
[2] University of California, Berkeley, USA

Abstract. In multi-view modeling the system under development is described by distinct models, called views, which capture different perspectives of the system. Inevitably, possible overlaps of the views may give rise to inconsistencies. Hence, it becomes essential to check for consistency among the separate views. Existing work checks view consistency of discrete systems (transition systems or finite automata) with respect to two types of abstraction functions: (1) projections of state variables, (2) projections of an alphabet of events onto a subalphabet. In this paper, we study view consistency with respect to *timing* abstractions, specifically, periodic sampling. We define the multi-view consistency problem for periodic sampling abstractions, and provide an algorithm for the problem.

1 Introduction

Designing complex systems, such as distributed, embedded, or cyber-physical systems, is a challenging task. As many of these systems are safety-critical, design by trial-and-error is not an option, and more rigorous methods such as model-based design are preferred (see [13] for an overview). In addition, the design of such systems involves several experts and stakeholders, each having their own perspective, or *view*, of the system [2,6,14]. These views are typically different kinds of models. These model cover different and potentially overlapping aspects of the system. In such a *multi-view modeling* setting, a basic problem is to check that the views are *consistent*, i.e., that they don't contradict each other [11].

In this paper we follow the multi-view modeling framework proposed in [11], where systems are sets of behaviors (often described by transition systems), and views are also sets of behaviors obtained by some kind of *abstraction* of system behaviors. Previous work studied the view consistency problem for discrete systems (transition systems or automata) with respect to two types of abstraction functions: (1) projections of state variables [11], and (2) projections of alphabet of events onto a subalphabet [10].

In this work we study the multi-view consistency problem for discrete systems with respect to *timing* abstractions and in particular *periodic sampling* abstraction functions. Given a period which is a positive integer number T, the periodic sampling abstraction consists in sampling the system once every T steps. That is, given a system behavior

* This work was partially supported by the Academy of Finland and the U.S. National Science Foundation (awards #1329759 and #1139138).

which is a sequence of states $s_0 s_1 s_2 \cdots$, the periodic sampling w.r.t. $T = 2$ produces the abstract behavior $s_0 s_2 s_4 \cdots$.

In summary the contributions of this paper are the following: first, we define the notions of (forward and inverse) periodic sampling abstraction functions; second, we study the closure of discrete systems under these abstraction functions; third, we provide an algorithm for the multi-view consistency problem for discrete systems in the periodic sampling setting. The algorithm is *sound* in the sense that if it reports that the views are inconsistent, then an inconsistency indeed exists. However, the algorithm may fail to detect *all* inconsistencies, as it relies on a state-based reachability, and inconsistencies may also involve the transition structure of the system.

2 Related work

The view consistency problem is a well-known problem in the engineering community. The several design teams engaged in the development process of a system obtain distinct models of the system utilizing versatile tools and modeling languages [3,12].

Existing literature mainly focus on designing architectures that combine various modeling tools or elements of the same tools [5,9,15], while a formal framework has been lacking with respect to behavioral views. [7,8] study behavioral views within the context of cyber-physical systems, in order to aid their verification rather than checking view consistency. This is the focus of the recent work [10,11] towards behavioral views. [11] offers a generic formal framework for multi-view modeling and its basic problems, since the system and views are within any global universe and most importantly they can be related by any kind of abstraction functions. The framework is instantiated for discrete systems using projection of state variables as abstraction functions, and the view consistency problem is solved. In [10] the framework is also extended to languages and automata, where abstraction functions are projections of alphabet of events onto a subalphabet.

Our work follows the setting of [10,11], but the abstraction functions studied there are different, and consist in *projections*, either of state variables, or of some events in the alphabet of events. Here we consider timing abstractions, and in particular periodic sampling abstraction functions, which to our knowledge have not been investigated earlier in the multi-view modeling context.

3 Background

Sets: Let S denote an arbitrary finite set: $|S|$ denotes its cardinality and $\mathcal{P}(S)$ denotes its powerset. Also: $\mathbb{Z}_{>0} := \{n \in \mathbb{Z} \mid n > 0\}$, $\mathbb{Z}_{\geq 0} := \{n \in \mathbb{Z} \mid n \geq 0\}$ are sets of integer numbers, and $\mathbb{B} := \{0, 1\}$ is the set of booleans.

Alphabet, Finite words, Infinite words: A finite alphabet Σ is a non-empty finite set of symbols. Σ^* is the set of all finite words over Σ and Σ^ω is the set of all infinite words over Σ. The set of all words over Σ is Σ^∞, i.e., $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

3.1 Automata

Nondeterministic finite automaton: A nondeterministic finite automaton (NFA for short) is a tuple $A = (Q, \Sigma, Q_0, \Delta, F)$ where Q is the finite set of states, Σ is the finite alphabet, $Q_0 \subseteq Q$ is the set of initial states, $\Delta \subseteq Q \times \Sigma \times Q$ is the transition function, and $F \subseteq Q$ is the set of final states. A path P_w of A over a finite word $w = w_0 \cdots w_{n-1} \in \Sigma^*$ is a finite sequence $P_w\colon (q_0, w_0, q_1) \cdots (q_{n-1}, w_{n-1}, q_n)$ such that $q_0 \in Q_0$ is the initial state and $(q_i, w_i, q_{i+1}) \in \Delta$ for every $0 \leq i < n$. A path P_w of A over a finite word $w \in \Sigma^*$ is called accepting if additionally $q_n \in F$. A finite word $w \in \Sigma^*$ is accepted by A if there is an accepting path P_w of A over w. The language accepted by A, also called the behavior of A, written $L(A)$, is the set of finite words accepted by A: $L(A) = \{w \in \Sigma^* \mid \exists \text{ accepting path } P_w \text{ of } A \text{ over } w\}$.

We say that A is *deterministic* (DFA for short) iff (i) $|Q_0| = 1$ and (ii) for every $q \in Q$ and $x \in \Sigma$ there exists at most one successor state $q' \in Q$ such that $(q, x, q') \in \Delta$. We call two automata *equivalent* iff they accept the same language. It is known that every nondeterministic finite automaton can be transformed into an equivalent deterministic finite automaton [4].

Nondeterministic Muller automaton: A nondeterministic Muller automaton (NMA for short) is a tuple $A = (Q, \Sigma, Q_0, \Delta, F)$ where Q, Σ, Q_0, Δ are defined as in a NFA and $F \subseteq 2^Q$. Now an infinite path P_w of A over an infinite word $w = w_0 w_1 \cdots \in \Sigma^\omega$ is an infinite sequence $P_w\colon (q_0, w_0, q_1)(q_1, w_1, q_2) \cdots$ such that $q_0 \in Q_0$ is the initial state and $(q_i, w_i, q_{i+1}) \in \Delta$ for every $i \geq 0$. For every path P_w of A over an infinite word $w \in \Sigma^\omega$ we denote with $Inf\,(P_w)$ the set of states occurring an infinite number of times along P_w. Then, a path P_w of A over $w \in \Sigma^\omega$ is called accepting if additionally $Inf(P_w) \in F$. An infinite word $w \in \Sigma^\omega$ is accepted by A if there is an accepting path P_w of A over w. The language accepted by A, also called the behavior of A, written $L(A)$, is the set of infinite words accepted by A: $L(A) = \{w \in \Sigma^\omega \mid \exists \text{ infinite accepting path } P_w \text{ of } A \text{ over } w\}$. A deterministic Muller automaton (DMA for short) is the deterministic variant of NMA, like DFAs are deterministic NFAs. Every nondeterministic Muller automaton has an equivalent deterministic Muller automaton [1].

3.2 Multi-view modeling

In multi-view modeling, one or more design teams obtain diverse models (views) of the same system under development, as they target capturing and analyzing different aspects of the system. For our framework we consider a system and its views as sets of behaviors [11]. There is no restriction on the behavior of the system, and we only assume that it is defined within an arbitrary global universe. Formally, *a system $\mathcal{S}$ over a domain of behaviors $\mathcal{U}$, is a subset of $\mathcal{U}$*: $\mathcal{S} \subseteq \mathcal{U}$. A view is intuitively an incomplete picture of a system, and may be obtained by some kind of transformation of the system behaviors into (incomplete) behaviors in another domain. Following [11], such a transformation is defined by means of an *abstraction function* $a : \mathcal{U} \to \mathcal{D}$, where $\mathcal{D}$ is the *view domain*. A *view $\mathcal{V}$ over view domain $\mathcal{D}$, is a subset of $\mathcal{D}$*: $\mathcal{V} \subseteq \mathcal{D}$.

However, it is not always the case that the system $\mathcal{S}$ is given. Indeed, usually, only the views are available and we need to check for the existence of such a system $\mathcal{S}$, which, in positive cases, is constructed from the views. The existence of $\mathcal{S}$ implies that the views should not have inconsistencies among them. But this raises the question of what does formally consistency mean?

Following [10,11] we define the notion of consistency. A set of views $\mathcal{V}_1, \cdots, \mathcal{V}_n$ over view domains $\mathcal{D}_1, \cdots, \mathcal{D}_n$, are *consistent with respect to a set of abstraction functions* $a_1, \cdots, a_n$, if there exists a system $\mathcal{S}$ over $\mathcal{U}$ so that $\mathcal{V}_i = a_i(\mathcal{S})$, for all $i = 1, \cdots, n$. In general, we call such a system $\mathcal{S}$ a *witness system* to the consistency of $\mathcal{V}_1, \cdots, \mathcal{V}_n$. Obviously, if there is no such system, then we conclude that the views are inconsistent.

In our setting, both systems and views are finite state discrete systems, as defined in the next section, and the views are obtained by applying periodic sampling abstraction functions.

3.3 Symbolic discrete systems

We consider finite state discrete systems described symbolically as in [11]. The state space is described by a (finite) set of boolean variables X, resulting in 2^n states where $n = |X|$, and a *state* s over X is a function $s : X \to \mathbb{B}$. A *behavior over* X is in general a finite or infinite sequence of states over X, $\sigma = s_0 s_1 \cdots$, where s_i denotes the state at position i. We denote with $\mathcal{U}(X)$ the set of all possible behaviors over X. Semantically, a discrete system $\mathcal{S}$ over X is a set of behaviors over X, i.e., $\mathcal{S} \subseteq \mathcal{U}(X)$.

In the sequel we give concrete (syntactic) representations of discrete systems of two kinds: first, *fully observable* systems where all variables are observable; second, *non-fully-observable* systems which also have internal, unobservable variables.

Fully-observable discrete systems: Initially we consider *fully-observable* symbolic discrete systems, i.e., systems where all variables are observable. Syntactically, a *fully-observable* discrete system (FOS for short) is defined by a triple $S = (X, \theta, \phi)$ where X is the finite set of boolean variables, θ is a boolean expression over X characterizing the set of all initial states, and ϕ is a boolean expression over $X \cup X'$, where $X' := \{x' \mid x \in X\}$ is the set of the next state variables. ϕ characterizes pairs of states (s, s') representing a transition from s to s' of S. We write $\theta(s)$ to denote that s satisfies θ. We write $\phi(s, s')$ to denote that the pair (s, s') satisfies ϕ, i.e., that there is a transition from s to s'.

A *behavior* of a FOS (X, θ, ϕ) is a finite or infinite sequence of states over X, $\sigma = s_0 s_1 \cdots$, such that σ can be generated by the FOS, i.e., such that $\theta(s_0)$ and $\forall i : \phi(s_i, s_{i+1})$. We denote by $[\![S]\!]$ the set of all behaviors of S.

Non-fully-observable discrete systems: Fully-observable systems can also be extended with a set of *internal, unobservable* state variables. For their definition we need to introduce the notion of hiding function.

Given a state s over the set of variables X and a subset $Y \subseteq X$, the *hiding function* h_Y projects s onto the set of variables Y, hence h_Y hides from s all variables in $X \setminus Y$. Then $h_Y(s)$ is defined to be the new state s', that is, $s' : Y \to \mathbb{B}$ such that $s'(x) = s(x)$

for every $x \in Y$. We extend hiding to sets of states and to behaviors. For a set of states $s = \{s_1, \cdots, s_n\}$ where $s_i : X \to \mathbb{B}$ for every $1 \leq i \leq n$, we define $h_Y(s) = \{h_Y(s_1), \cdots, h_Y(s_n)\}$. If $\sigma = s_0 s_1 \cdots$ is a behavior over X, then $h_Y(\sigma)$ is a behavior over Y defined by $h_Y(\sigma) := h_Y(s_0)h_Y(s_1)\cdots$. If $\mathcal{S}$ is a discrete system over X, then $h_Y([\![\mathcal{S}]\!]) := \{h_Y(\sigma) \mid \sigma \in [\![\mathcal{S}]\!]\}$.

Formally, a *non-fully-observable* discrete system (nFOS for short), described symbolically, is a tuple $S = (X, Z, \theta, \phi)$ where X, Z are disjoint finite sets of variables such that X describes the set of observable variables, and Z the set of internal (unobservable) variables. The initial condition θ is a boolean expression over $X \cup Z$, and the transition relation ϕ is a boolean expression over $X \cup Z \cup X' \cup Z'$.

A behavior of a nFOS $S = (X, Z, \theta, \phi)$ is a finite or infinite sequence of states over $X \cup Z$ which can be generated by $\mathcal{S}$, in the same manner as with behaviors generated by a FOS. The *observable behavior* of a behavior σ over $X \cup Z$ is the behavior $h_X(\sigma)$ over X. In what follows we denote by $[\![S]\!]$ the set of all behaviors of S (over $X \cup Z$), and by $[\![S]\!]_o$ the set of its observable behaviors (over X). If $Z = \emptyset$, i.e., the system has no internal variables, then it is a FOS. Note that for every FOS S, it holds that $[\![S]\!] = [\![S]\!]_o$.

4 Forward and inverse periodic sampling abstraction functions

Now we would like to relate systems over $\mathcal{U}$ and views over $\mathcal{D}$, using periodic sampling abstraction functions. We define as *period* any $T \in \mathbb{Z}_{>0}$. Note that in general, we can apply the periodic sampling to the behaviors starting from the state at position $\tau = 0$ or at $\tau > 0, \tau \in \mathbb{Z}$.

Periodic sampling abstraction functions (forward): Let X be a finite set of variables. Given a domain of behaviors $\mathcal{U}(X)$ and a view domain $\mathcal{D}(X) = \mathcal{U}(X)$, a periodic sampling abstraction function from $\mathcal{U}(X)$ to $\mathcal{D}(X)$ w.r.t. period T and initial position τ, denoted by $a_{T,\tau}$, is defined by the mapping $a_{T,\tau} : \mathcal{U}(X) \to \mathcal{D}(X)$ such that for every behavior $\sigma = s_0 s_1 \cdots \in \mathcal{U}(X)$, $a_{T,\tau}(\sigma) := s'_0 s'_1 \cdots \in \mathcal{D}(X)$ where $s'_i = s_{\tau+i\cdot T}$ for every $i \geq 0$. When $\tau = 0$, instead of writing $a_{T,\tau}$ or $a_{T,0}$, we simply write a_T.

Then we lift the notion of periodic sampling abstraction function to systems. For a system $\mathcal{S} \subseteq \mathcal{U}(X)$, we define $a_{T,\tau}(\mathcal{S}) := \{a_{T,\tau}(\sigma) \mid \sigma \in \mathcal{S}\}$. Since $a_{T,\tau}(\mathcal{S}) \subseteq \mathcal{D}(X)$, $a_{T,\tau}(\mathcal{S})$ is a view over $\mathcal{D}(X)$. In what follows we refer to periodic sampling abstraction functions simply by periodic sampling.

Closure of discrete systems under periodic sampling: Given a nFOS $S = (X, Z, \theta, \phi)$ and periodic sampling $a_{T,\tau} : \mathcal{U}(X \cup Z) \to \mathcal{D}(X \cup Z)$, $\mathcal{D}(X \cup Z) = \mathcal{U}(X \cup Z)$, we would like to examine whether there exists a nFOS $S' = (X, Z, \theta', \phi')$ such that $[\![S']\!] = a_{T,\tau}([\![S]\!])$. Indeed, we prove closure for discrete systems S with $Z = \emptyset$ or $Z \neq \emptyset$.

Theorem 1. *(a) Given a FOS system $S = (X, \theta, \phi)$ and periodic sampling $a_{T,\tau}$, there exists a FOS system S' such that $[\![S']\!] = a_{T,\tau}([\![S]\!])$.*

(b) Given a nFOS system $S = (X, Z, \theta, \phi)$ and periodic sampling $a_{T,\tau}$, there exists a nFOS system S' such that $[\![S']\!] = a_{T,\tau}([\![S]\!])$.

Proof. (a) We define the FOS $S' = (X, \theta, \phi')$, where θ' contains all states over X which can be reached from some initial state of S in exactly τ steps; and ϕ' is defined as follows. Let s, s' be two states over X. Then $\phi'(s, s')$ iff S has a path from s to s' of length exactly T. Consider an arbitrary behavior $\sigma = s_0 s_1 s_2 \cdots \in [\![S]\!]$. Applying the periodic sampling $a_{T,\tau}$ to σ we obtain the behavior $a_{T,\tau}(\sigma) = s_\tau s_{\tau+T} s_{\tau+2T} \cdots$. By construction of S' we have that $\theta'(s_\tau)$ and $\phi'(s_{\tau+iT}, s_{\tau+(i+1)T})$ for every $i \geq 0$, which implies that $a_{T,\tau}(\sigma) \in [\![S']\!]$. Hence, $a_{T,\tau}([\![S]\!]) = \{a_{T,\tau}(\sigma) \mid \sigma \in [\![S]\!]\} \subseteq [\![S']\!]$. Conversely, let $\sigma' = s'_0 s'_1 s'_2 \ldots \in [\![S']\!]$. Since $\phi'(s'_0)$, by definition of S' there exists a state s_0 in S with $\theta(s_0)$ so that s'_0 can be reached from s_0 in exactly τ steps. Moreover, for σ' we have that $\phi'(s'_i, s'_{i+1})$, thus there exists a path in S from s'_i to s'_{i+1} of length exactly T for every $i \geq 0$. Then, we obtain the behavior $\sigma = s_0 s_1 s_2 \cdots \in [\![S]\!]$ where $s_{\tau+iT} = s'_i$ for every $i \geq 0$. Hence, $a_{T,\tau}(\sigma) \in a_{T,\tau}([\![S]\!])$ and $[\![S']\!] \subseteq a_{T,\tau}([\![S]\!])$ which completes our proof. The part (b) of the theorem is proved similarly. □

Inverse periodic sampling: Consider a finite set of variables X, a domain of behaviors $\mathcal{U}(X)$ and a view domain $\mathcal{D}(X) = \mathcal{U}(X)$. Then, an inverse periodic sampling abstraction function from $\mathcal{D}(X)$ to $\mathcal{U}(X)$ w.r.t. period T and initial position τ, denoted by $a^{-1}_{T,\tau}$, is defined by the mapping $a^{-1}_{T,\tau} : \mathcal{D}(X) \to \mathcal{U}(X)$ such that for every behavior $\sigma = s_0 s_1 \cdots \in \mathcal{D}(X)$, $a^{-1}_{T,\tau}(\sigma) := \{\sigma' \mid \sigma' = s'_0 s'_1 \cdots \in \mathcal{U}(X)$ s.t. $s'_{\tau+i\cdot T} = s_i, i \geq 0\}$ or equivalently $a^{-1}_{T,\tau}(\sigma) := \{\sigma' \mid a_{T,\tau}(\sigma') = \sigma\}$. Moreover, for a system $\mathcal{S} \subseteq \mathcal{U}(X)$, we define $a^{-1}_{T,\tau}(\mathcal{S}) := \bigcup_{\sigma \in \mathcal{S}} a^{-1}_{T,\tau}(\sigma)$. When $\tau = 0$, we simply write a^{-1}_T.

Non-closure of FOS under inverse periodic sampling: Given a FOS $S = (X, \theta, \phi)$ and inverse periodic sampling $a^{-1}_{T,\tau} : \mathcal{D}(X) \to \mathcal{U}(X)$, $\mathcal{D}(X) = \mathcal{U}(X)$, we show that there does not exist always a FOS $S' = (X, \theta', \phi')$ such that $[\![S']\!] = a^{-1}_{T,\tau}([\![S]\!])$.

Example 1. Consider the FOS $S = (\{x, y\}, \theta, \phi)$ where both x and y are Boolean variables, $\theta = x \wedge y$ and $\phi = (x \wedge y \to \neg x' \wedge y') \wedge (\neg x \wedge y \to x' \wedge \neg y') \wedge (x \wedge \neg y \to x' \wedge y')$ as shown in Figure 1. The system S has been obtained with periodic sampling a_T and period $T = 2$. In order to find a system S' such that $[\![S']\!] = a^{-1}_T([\![S]\!])$ one would have to replace each of the "?" shown in Figure 2 with at least one state over X, so that the first 6 states in the unique behavior σ of S' would be distinct. Indeed, S' needs at least 6 distinct states, otherwise we would have to have extra transitions between the three states $(1, 1), (0, 1), (1, 0)$ shown in the figure. But adding such transitions creates loops, and results in $[\![S']\!] \neq a^{-1}_T([\![S]\!])$, which is wrong. However, having 6 distinct states is also not possible, since we only have two Boolean variables x, y, thus, only 4 possible combinations. Hence, there exists no such FOS S'.

Closure of nFOS under inverse periodic sampling: In contrast with FOS, that are not closed under inverse periodic samplings, we prove closure for nFOS, i.e., for discrete systems with internal variables.

Theorem 2. *Given a system $S = (X, Z, \theta, \phi)$ and inverse periodic sampling $a^{-1}_{T,\tau} : \mathcal{D}(X \cup Z) \to \mathcal{U}(X \cup Z \cup W)$, there exists always a non-fully-observable system $S' = (X \cup Z, W, \theta', \phi')$ such that $[\![S']\!] = a^{-1}_{T,\tau}([\![S]\!])$.*

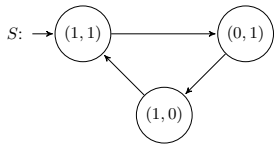


Fig. 1: FOS obtained from a_T with period $T = 2$.

Position	σ
$i = 0$	(1,1)
	|
$i = 1$	?
	|
$i = 2$	(0,1)
	|
$i = 3$	?
	|
$i = 4$	(1,0)
	|
$i = 5$	?
	|
$i = 6$	(1,1)
	|
$i = 7$	$\vdots$

Fig. 2: Incomplete behavior σ of system S'.

Proof. Given the nFOS $S = (X, Z, \theta, \phi)$ let R denote the set of reachable states of S over $X \cup Z$. Moreover, let $|R| = n$ and consider a set of Boolean variables W such that $|W| \geq \lfloor \log_2(n \cdot (T-1) + \tau) \rfloor$ (here we assume that $T \geq 2$; if $T = 1$ then we can simply take $S' = S$). By definition we have that $\sigma \in a_{T,\tau}^{-1}([\![S]\!])$ iff $a_{T,\tau}(\sigma) \in [\![S]\!]$. Moreover, $\sigma' = a_{T,\tau}(\sigma) = s_\tau s_{\tau+T} s_{\tau+2T} \cdots$, i.e., each behavior σ' in $[\![S]\!]$ has been obtained with starting position τ and period T. The system S' has to be defined such that each behavior in $[\![S']\!]$ results from σ' by (i) adding τ transitions (or states) in the beginning of σ' and T transitions (or $T-1$ states) in between the transition $\phi(s_{\tau+iT}, s_{\tau+(i+1)T})$ for every $i \geq 0$, and by (ii) replacing each $s_{\tau+iT}$ in σ' with $s'_{\tau+iT} = h_{X \cup Z}(s_{\tau+iT})$. Since S consists of n reachable states then S' should have at least $n(T-1) + \tau$ more reachable states or equivalently $\lfloor \log_2(n \cdot (T-1) + \tau) \rfloor$ more Boolean variables. One can then obtain a nFOS S' over $X \cup Z \cup W$, where $X \cup Z$ and W denote the set of observable and unobservable variables respectively, such that $[\![S']\!] = a_{T,\tau}^{-1}([\![S]\!])$. $\square$

Note that the system S' of Theorem 2 is not unique. Indeed, even for each possible value of $|W|$ one obtains a family of systems S' with $[\![S']\!] = a_{T,\tau}^{-1}([\![S]\!])$.

5 Checking View Consistency w.r.t. Periodic Sampling Abstraction Functions

For this entire section, we assume that $\tau = 0$.

5.1 Views and Consistency

Views are finite-state discrete systems with or without internal variables. In our framework views are obtained applying some periodic sampling a_T, where T is the period of the periodic sampling. Thus, if $S = (X, Z, \theta, \phi)$ is a discrete system over a set of observable variables X and domain of behaviors $\mathcal{U}(X \cup Z)$, then a view obtained with periodic sampling a_T is a discrete system $V = (X, Z, \theta', \phi')$ over the same set of observable variables X and view domain $\mathcal{D}(X \cup Z) = \mathcal{U}(X \cup Z)$.

We consider views defined by nFOS and we refer to V simply as a view of S. However, this does not exclude the case where the views are described by a FOS. Indeed, FOS is a special case of nFOS and we can always assume that the set of unobservable variables is empty. Hence, the results we derive can be naturally extended also for views described by FOS. Moreover, in the rest of the paper when we compare systems or views we compare them with respect to their observable behaviors, and instead of writing for instance $[\![V]\!]_o = a_T([\![S]\!]_o)$ we simply write $[\![V]\!] = a_T([\![S]\!])$.

Note that, although each of the views is a nFOS, this does not always imply that the *witness* system is also a nFOS. This motivates to study three different variants of the consistency problem for views obtained by periodic sampling.

Problem 1. Given a finite set of nFOS $S_i = (X, W_i, \theta_i, \phi_i)$ and periodic samplings a_{T_i}, for $1 \leq i \leq n$, check whether there exists a system $\mathcal{S}$ over $\mathcal{U}(X)$ such that $a_{T_i}(\mathcal{S}) = [\![S_i]\!]$ for every $1 \leq i \leq n$.

Problem 2. Given a finite set of nFOS $S_i = (X, W_i, \theta_i, \phi_i)$ and periodic samplings a_{T_i}, for $1 \leq i \leq n$, check whether there exists an nFOS $S = (X, W, \theta, \phi)$, $W \supseteq W_1 \cup \ldots \cup W_n$, such that $a_{T_i}([\![S]\!]) = [\![S_i]\!]$ for every $1 \leq i \leq n$.

Problem 3. Given a finite set of nFOS $S_i = (X, W_i, \theta_i, \phi_i)$ and periodic samplings a_{T_i}, for $1 \leq i \leq n$, check whether there exists a fully-observable system $S = (X, \theta, \phi)$, such that $a_{T_i}([\![S]\!]) = [\![S_i]\!]$ for every $1 \leq i \leq n$.

Observe that the three problems are different, since Problem 1 asks for a *semantic* witness system, not necessarily representable as a symbolic discrete system, while Problems 2 and 3 ask for a symbolic discrete witness system with or without internal variables respectively. Obviously, a solution to Problem 3 is also a solution to Problem 2, and a solution to Problem 2 is also a solution to Problem 1. We do not yet know whether Problems 1 and 2 are equivalent, i.e., whether existence of a semantic witness implies existence of a syntactic (nFOS) witness. This is an interesting question which has to do with whether the finite-state nature of nFOS is enough to represent all possible semantic witnesses of consistent nFOS views. Example 2 that follows shows that Problems 2 and 3 are not equivalent, that is, existence of a nFOS witness does not always imply existence of a FOS witness.

Example 2. Consider the views $V_i = (\{x_i\}, \{y_i, z_i\}, \theta_i = \neg x_i \wedge \neg y_i \wedge \neg z_i, \phi_i)$ for $i = 1, 2$, where all variables are Boolean and ϕ_1, ϕ_2 are such that $[\![V_1]\!] = \{\sigma_1 = (0,0,0)(0,1,1)(0,1,0)(0,0,1), \sigma_1' = (0,0,0)(1,1,1)(1,1,0)(1,0,1)\}$ and $[\![V_2]\!] = \{\sigma_2 = (0,0,0)(0,1,1)(0,0,1), \sigma_2' = (0,0,0)(1,1,0)(1,1,1)\}$. The views V_1 and V_2 have been sampled with periods $T_1 = 2$ and $T_2 = 3$ respectively. The observable behavior of the views is shown in the form of trees in Figure 3, along with the corresponding positions as described in the sequel. For the view V_2 which has been sampled with period $T_2 = 3$, we have that in the first position $i = 0$, V_2 is at state $x = 0$, in the next position $i = 3$, V_2 is at one of the states $x = 0$ or $x = 1$, and similarly in the last position $i = 6$. Similarly are interpreted the tree of behaviors for V_1. There exists a nFOS system S witness to the consistency of V_1 and V_2, with one observable state variable, whose observable behavior is shown in the form of a tree in the rightmost part of Figure 3 where the $*-$state of the system denotes an arbitrary state 0 or 1. However, there does not exist any fully-observable system with a single state variable x that is a witness system to the consistency of V_1 and V_2. For instance, it is not possible to define by distinct states, the five states labelled by 0 in the positions $i = 0, 2, 3, 4, 6$, using only one Boolean variable (as it can only encode 2 states).

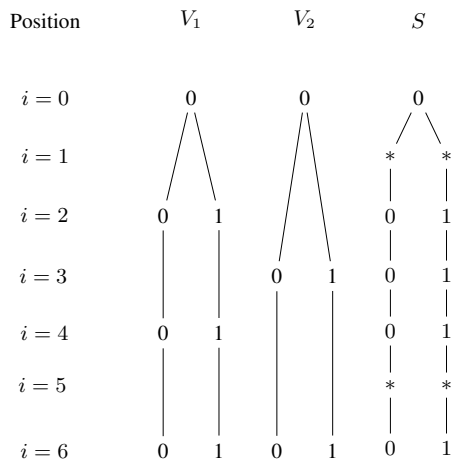


Fig. 3: Behavior trees for views V_1 and V_2 and possible nFOS witness system S.

In the sequel we prove a Lemma that will help prove one of the main results of this paper. We firstly introduce some notation. Consider a set of views $S_1, ..., S_n$. For every $1 \leq i \leq n$ and any positive integer m let Y_i^m be the set of all states that can be found at position m in some behavior of S_i, i.e., $Y_i^m = \{s_i \mid s_i : X \to \mathbb{B}$ occurs at position m in some behavior $\sigma \in [\![S_i]\!]\}$.

Lemma 1. *Consider a set of views $S_1, ..., S_n$ and periodic samplings a_{T_i}, for $1 \leq i \leq n$. If there exist $i, j \in \{1, ..., n\}$ and positive integer m multiple of $LCM(T_i, T_j)$ such that $Y_j^{m/T_j} \neq Y_i^{m/T_i}$, then $S_1, ..., S_n$ are inconsistent.*

Proof. Let $S_i = (X, W_i, \theta_i, \phi_i)$. Assume that there exist $i, j \in \{1, ..., n\}$ and positive integer m multiple of $LCM(T_i, T_j)$ such that $Y_j^{m/T_j} \neq Y_i^{m/T_i}$. W.l.o.g., suppose that there exists a state $s \in Y_i^{m/T_i} \setminus Y_j^{m/T_j}$. We would like to prove that the views $S_1, ..., S_n$ are inconsistent. Assume to the contrary that they are consistent. This implies that there exists a system $\mathcal{S}$ over $\mathcal{U}(X)$ such that $a_{T_k}(\mathcal{S}) = [\![S_k]\!]$ for every $1 \leq k \leq n$. Then, $a_{T_i}(\mathcal{S}) = [\![S_i]\!]$ and $a_{T_j}(\mathcal{S}) = [\![S_j]\!]$. Since there exists state $s \in Y_i^{m/T_i} \setminus Y_j^{m/T_j}$, then there exists some behavior $\sigma_i \in [\![S_i]\!]$ such that σ_i is at position m/T_i at state s. Because $a_{T_i}(\mathcal{S}) = [\![S_i]\!]$ we have that $\sigma_i \in a_{T_i}(\mathcal{S})$. By definition, $a_{T_i}(\mathcal{S}) = \{a_{T_i}(\sigma) \mid \sigma \in \mathcal{S}\}$ and because $\sigma_i \in a_{T_i}(\mathcal{S})$ then $\exists \sigma \in \mathcal{S}$ such that $a_{T_i}(\sigma) = \sigma_i$. By construction, σ is at state s at position m. Since $\sigma \in \mathcal{S}$ we have that $a_{T_j}(\sigma) \in a_{T_j}(\mathcal{S}) = [\![S_j]\!]$. Let $\sigma_j = a_{T_j}(\sigma)$. Because σ is at state s at position m, σ_j must be at the same state s at position m/T_j. This in turn implies that $s \in Y_j^m$, which is a contradiction. □

5.2 Algorithm for detecting view inconsistency

In this chapter we describe the steps of the algorithm for detecting inconsistencies among a finite number of views, w.r.t. periodic sampling abstraction functions. Our algorithm applies to sets of views that satisfy one of the following conditions: (i) either every view generates only infinite behaviors; (ii) or every view generates only finite behaviors. This means that we cannot have views that have both finite and infinite behaviors, and also that we cannot have some views with finite behaviors and some other views with infinite behaviors. Extending the algorithm to those cases is part of future work. Note that a view which only has finite behaviors corresponds to a transition system where all paths eventually lead to a deadlock. On the other hand, a view which only has infinite behaviors corresponds to a transition system with no reachable deadlocks.

From now on we use the term *finite automata* (FA for short) to refer to NFA or NMA, and the term *deterministic finite automata* to refer to DFA or DMA.

Our algorithm involves constructing the so called *hyper-period automaton* (HPA). The algorithm also involves a special composition operator for finite automata w.r.t. HPA called the *label-driven composition*. We define these notions next.

Hyper-period finite automaton: Consider a finite set of periods $T_1, \ldots, T_n \in \mathbb{Z}_{>0}$. Let LCM be the least common multiple operator, and let $T = LCM(T_1, \ldots, T_n)$ be the *hyper-period* of the above set of periods. Also let $M = \{0, m_1, \ldots, m_k\}$ denote the finite ordered set of multiples of $T_1, \ldots, T_n$ up to the hyper-period, i.e., with $m_k < T$. For example, let $T_1 = 2, T_2 = 3$ and $T_3 = 6$. Then, $T = LCM(T_1, T_2, T_3) = 6$ and $M = \{0, 2, 3, 4\}$.

The intuition of the hyper-period automaton is that it contains as states the elements of M and its transitions are labelled with sets of labels of the form p_{T_i}, denoting the fact that the period T_i is "active" at the corresponding time instant. The accepting states of the automaton correspond to those instants where two or more periods are "active".

We first illustrate this intuition by example, and then provide the formal definition of the hyper-period automaton.

Example 3. Consider the two periods $T_1 = 3$ and $T_2 = 2$. Then, the hyper-period automaton w.r.t. T_1, T_2, is the automaton H shown in Figure 4. We have $H = (M, P, \{0\}, \Delta, \{0\})$ where $M = \{0, 2, 3, 4\}$, $P = \mathcal{P}(\{p_{T_1}, p_{T_2}\}) = \mathcal{P}(\{p_3, p_2\})$ ($T_1 = 3$ and $T_2 = 2$, hence $p_{T_1} = p_3$ and $p_{T_2} = p_2$), and the transition function Δ is as depicted in Figure 4.

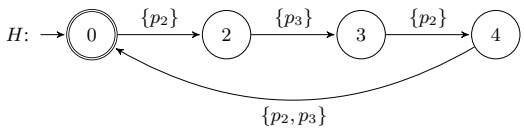


Fig. 4: HPA w.r.t. the periods 2 and 3.

The above example of HPA is simple as we only have two periods. In the sequel we provide a more interesting example which involves three periods. This example also illustrates the fact that the HPA generally has more than one accepting states.

Example 4. Consider the three periods $T_1 = 4$, $T_2 = 2$ and $T_3 = 3$. Then, the hyper-period automaton is $H = (M, P, \{0\}, \Delta, F)$ where $M = \{0, 2, 3, 4, 6, 8, 9, 10\}$, $P = \mathcal{P}(\{p_{T_1}, p_{T_2}, p_{T_3}\}) = \mathcal{P}(\{p_4, p_2, p_3\})$, $F = \{0, 4, 6, 8\}$, and the transition function Δ is as depicted in Figure 5.

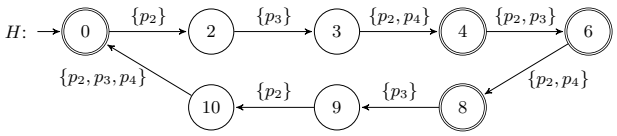


Fig. 5: HPA w.r.t. the periods 2, 3 and 4.

We now formally define the *hyper-period automaton* H w.r.t. $T_1, \ldots, T_n$ as the deterministic finite automaton $H = (M, P, \{0\}, \Delta, F)$, where:

- M is the (finite) set of states of H.
- $P = \mathcal{P}(\{p_{T_1}, \ldots, p_{T_n}\})$ is the (finite) alphabet of H, where $\{p_{T_1}, \ldots, p_{T_n}\}$ is obtained by assigning to every $i = 1, \ldots, n$ the label p_{T_i}, corresponding to the period T_i.
- Δ is defined as follows. First, let $M = \{m_0, m_1, ..., m_k\}$ where we assume that m_i are ordered in increasing sequence, i.e., $m_0 < m_1 < \cdots < m_k$. Note that under this assumption, m_0 must be 0. Then Δ contains exactly $k + 1$ transitions (i.e., as many as the elements of M): $\Delta = \{(m_0, l_0, m_1), (m_1, l_1, m_2), ...,$

$(m_k, l_k, m_0)\}$, where for $i = 0, ..., k$, $l_i = \{p_{T_j} \mid T_j$ is a divisor of $m_{i+1}\}$. Note that, as defined, Δ creates a loop starting at the initial state $m_0 = 0$, and ending at the same state, with each state in M having a unique successor as well as a unique predecessor. Given $m \in M$, let $\pi(m)$ denote the set of period labels annotating the unique incoming transition to m. For example, in the HPA of Figure 5, $\pi(3) = \{p_3\}$ and $\pi(4) = \{p_2, p_4\}$.

- $F = \{m \in M \mid |\pi(m)| \geq 2\}$, that is, F contains all states whose (unique) incoming transition is labeled by a set containing at least two period labels.

Since H is deterministic, in the sequel we use for simplicity the notation $\Delta(s, x)$ for the transition function, i.e., to denote the unique successor state of s with symbol x.

Label-driven composition of view automata with an HPA: Suppose we want to check consistency between a set of views described as finite automata $A_1, ..., A_n$, w.r.t. periods $T_1, ..., T_n$. Our view consistency algorithm, described later in this section, relies on computing a special kind of automata composition among modified versions of $A_1, ..., A_n$ and the HPA H w.r.t. $T_1, ..., T_n$. The modified version of A_i consists essentially in labeling all transitions of A_i by its period label p_{T_i}, and then determinizing. Then, the composition with H consists in synchronizing every transition of H with the corresponding automata A_i whose period is "active" on that transition, i.e., whose label p_{T_i} belongs to the corresponding label set of the transition of H. This composition is called *label-driven composition*, and is formalized next.

Consider a set of deterministic finite automata $A_r = (Q_r, \Sigma_r, Q_{r_0}, \Delta_r, F_r)$ where $\Sigma_r = \{p_{T_r}\}$ for $1 \leq r \leq n$. Let $H = (M, P, \{0\}, \Delta, F)$ be the HPA w.r.t. $T_1, ..., T_n$ defined as above. The *label-driven composition* of $A_1, ..., A_n$ and H, denoted by $A_1 \parallel \ldots \parallel A_n \parallel H$, is the finite automaton $C = (Q_c, \Sigma_c, Q_{c_0}, \Delta_c, F_c)$ where $Q_c = Q_1 \times \cdots \times Q_n \times M$, $\Sigma_c = P$, $Q_{c_0} = Q_{1_0} \times \cdots \times Q_{n_0} \times \{0\}$, $F_c = Q_1 \times \cdots \times Q_n \times F$, and the transition function $\Delta_c \subseteq Q_c \times \Sigma_c \times Q_c$ is defined as follows:

$$\Delta_c = \{((q_1, \ldots, q_n, m), l, (q'_1, \ldots, q'_n, m')) \mid (m, l, m') \in \Delta \ \wedge \\ \forall i = 1, ..., n: \text{ if } p_{T_i} \in l \text{ then } (q_i, p_{T_j}, q'_i) \in \Delta_i, \text{ otherwise } q'_i = q_i\}.$$

Example 5. Consider the finite automata A_1, A_2 as shown in Figure 6 and the HPA H of Figure 4. Then, Figure 7 depicts the label-driven composition $C = A_1 \parallel A_2 \parallel H$.

Fig. 6: Finite automata A_1 and A_2.

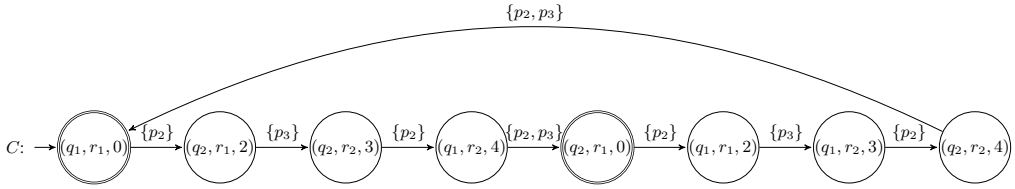


Fig. 7: The label-driven composition of automata A_1, A_2 of Figure 6 and HPA H of Figure 4.

Algorithm for detecting view inconsistency: Consider a finite set of views defined by the nFOS $S_i = (X, W_i, \theta_i, \phi_i)$, and obtained by applying some periodic sampling a_{T_i} with sampling period T_i, for $i = 1, \ldots, n$, respectively.

Let $T = LCM(T_1, \ldots, T_n)$, $P = \mathcal{P}(\{p_{T_1}, \ldots, p_{T_n}\})$ and $M = \{0, m_1, \ldots, m_k\}$ denote respectively the hyper-period of periods, the labels of periods, and the ordered set of multiples of periods up to their hyper-period, as defined previously. The algorithm for detecting inconsistency among the views $S_1, \ldots, S_n$, consists of the following steps:

- **Step 1:** Construct for each S_i, $i = 1, \ldots, n$, the FA $L_i = (Q_i, \Sigma_i, Q_{i_0}, \Delta_i, F_i)$ where $Q_i = \mathbb{B}^{X \cup W_i}$, $\Sigma_i = \{p_{T_i}\}$, $Q_{i_0} = \{s \mid \theta_i(s)\}$, $F_i = \emptyset$, and $\Delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is defined such that $(s, p_{T_i}, s') \in \Delta_i$ iff $\phi_i(s, s')$.
- **Step 2:** Determinize each of the FA L_i and obtain the equivalent deterministic FA dL_i for every $i = 1, \ldots, n$.
- **Step 3:** Construct the hyper-period automaton H w.r.t. the periods $T_1, \ldots, T_n$.
- **Step 4:** Obtain the label-driven composition $C = (dL_1, \ldots, dL_n, H)$ w.r.t HPA H.
- **Step 5:** Let $s = (s_1, \ldots, s_n, m)$ be a state of C, and let $I_s = \{i \in \{1, ..., n\} \mid p_{T_i} \in \pi(m)\}$. The algorithm reports **inconsistency** if C contains at least one reachable state $s = (s_1, \ldots, s_n, m)$ where s_i are states of dL_i for $i = 1, \ldots, n$ respectively, and $m \in F$ is a final state of H, such that $\exists i, j \in I_s : h_X(s_i) \neq h_X(s_j)$. Otherwise, it reports **inconclusive.**

The determinization procedure at Step 2 is needed because the algorithm detects view inconsistency by comparing sets of states of where each of the given views can be at same points in time. We note that this determinization procedure does not attempt to *complete* an automaton, i.e., to add transitions with missing symbols.

In the sequel we prove that the algorithm is sound, i.e., if it reports inconsistency then the views are indeed inconsistent. We first introduce two auxiliary lemmas used for proving this fact.

Lemma 2. *Let S be a nFOS, T a period, L the FA obtained from S as in Step 1 of the algorithm above, and dL the deterministic FA obtained from L. Then every reachable state of dL is non-empty.*

Proof. The set of initial states of S, and therefore also of L, is non-empty. Therefore, the initial state of dL is a non-empty set of states of L. Recall that the alphabet of both L and

dL is the singleton $\{p_T\}$. Let s be a state of dL. We show by induction that if s is a non-empty set, and (s, p_T, s') is a transition of dL, then s' is also non-empty. By definition of dL, if all elements of s are deadlocks, i.e., none of them has a transition with p_T, then no transition is added to s either (note that we do not complete the automaton dL as we mentioned above). Therefore, in order for a transition (s, p_T, s') to exist, at least one state $q \in s$ must have a transition (q, p_T, q') in L. But in that case, s' contains at least the state q', and is therefore non-empty. □

We now introduce some concepts used in the lemma that follows. First, observe that the HPA H is finite and deterministic, and so is every automaton dL_i. Moreover, by definition, the label driven composition C obtained by Step **4** of the algorithm forms a *lasso*, that is, C is finite-state and every state has a unique successor state. Consider a state m of the HPA H as obtained by Step **3** of the algorithm. We define the *indices* of m, denoted by $ins(m)$, to be the ordered set of numbers $ins(m) = \bigcup_{w \geq 0} \{w \cdot LCM(T_1, \ldots, T_n) + m\}$. For instance, consider the HPA H of Example 3. Since $T_1 = 3$ and $T_2 = 2$ we have that $LCM(T_1, T_2) = 6$. Then, the indices of the state $m = 3$ is the set $ins(3) = \bigcup_{w \geq 0} \{w \cdot 6 + 3\} = \{3, 9, 15, \ldots\}$.

Now consider a reachable state $s = (q_1, \ldots, q_n, m)$ of the label driven composition C. Because C is a lasso, there is a unique acyclic path in C that reaches m. Let ξ be the number of times that the state m of the HPA occurs in the states of this path reaching s. We define the *latent index* of s, denoted by $lin(s)$, to be the element of $ins(m)$ that occurs in position ξ in $ins(m)$. For instance, consider the label driven composition C of Example 5. For the state $s = (q_1, r_2, 3)$ of C we have that $m = 3$ and $ins(3) = \{3, 9, 15, \ldots\}$ computed as above. The state $m = 3$ of the HPA H occured twice up to the state $s = (q_1, r_2, 3)$, hence $\xi = 2$. Then, the second element of $ins(m)$ is the integer 9, and thus $lin(s) = 9$. The intuition is that $lin(s)$ represents the first point in time where s appears in a behavior of the system.

Lemma 3. *Consider the label driven composition C as obtained by Step* **4** *of the algorithm. Let $s = (s_1, \ldots, s_n, m)$ be a state of C, and let $I_s = \{i \in \{1, ..., n\} \mid p_{T_i} \in \pi(m)\}$. Then, for every $i \in I_s$, $lin(s)/T_i$ is an integer number and it holds that $s_i = Y_i^{lin(s)/T_i}$.*

Proof. Let $i \in I_s$. Then $p_{T_i} \in \pi(m)$. By definition of the HPA H, and since p_{T_i} belongs as a label in the incoming transition to m, $lin(s)$ is a multiple of the period T_i, thus $k = lin(s)/T_i$ is an integer number. We also need to show that $s_i = Y_i^k$. By definition of C, automaton dL_i has "moved" (i.e., taken a transition) k times up to state s. Thus, s_i must contain the set of all states of L_i that can be reached from an initial state after k steps. But this is exactly the set of states contained in Y_i^k. □

Theorem 3. *If the algorithm reports inconsistency then there exists no solution to Problems 1, 2, and 3.*

Proof. Assume that the algorithm reports inconsistency. Then there exists a reachable final state $(s_1, \ldots, s_n, m)$ of C such that $\exists i, j \in I_s : h_X(s_i) \neq h_X(s_j)$, where s_i is a state of dL_i, s_j is a state of dL_j, and m is a final state of the HPA H. By Lemma 2 we

have that $s_i \neq \emptyset$ and $s_j \neq \emptyset$. Moreover by Lemma 3 we have that $s_i = Y_i^{lin(s)/T_i}$ and $s_j = Y_j^{lin(s)/T_j}$, where $lin(s)$ is the latent index of state s. Since $h_X(s_i) \neq h_X(s_j)$ we obtain that $s_i \neq s_j$ and hence $Y_i^{lin(s)/T_i} \neq Y_j^{lin(s)/T_j}$. Then, by Lemma 1 the views $S_1, \ldots, S_n$ are inconsistent. Hence, there is no solution to Problem 1, and therefore neither to Problems 2 and 3. □

The algorithm is sound, but not complete, i.e., if the algorithm reports "inconclusive" then the views can either be consistent or not. Example 6 that follows indicates this fact.

Example 6. Consider the views $V_i = (\{x_i\}, \{y_i, z_i, w_i\}, \theta_i = \neg x_i \wedge \neg y_i \wedge \neg z_i \wedge \neg w_i, \phi_i)$ for $i = 1, 2$, where all variables are Boolean and ϕ_1, ϕ_2 are such that $[\![V_1]\!]_o = \{\sigma_1 = (0)(0)(0)(0)(0), \sigma_1' = (0)(1)(1)(1)(1)\}$ and $[\![V_2]\!]_o = \{\sigma_2 = (0)(0)(1), \sigma_2' = (0)(1)(0)\}$. The views V_1 and V_2 have been sampled with periods $T_1 = 2$ and $T_2 = 4$ respectively. The observable behavior of the views is shown in the form of trees in Figure 8. We claim that the views V_1 and V_2 are inconsistent. Assume the contrary. Then a possible witness system $\mathcal{S}$ should contain at least one behavior which at position $i = 8$ is at state $x = 0$, while at position $i = 4$ it is at state $x = 0$ (like σ_1 above), and also at least one behavior which at position $i = 8$ is at state $x = 1$ while at position $i = 4$ it is at state $x = 0$ (like σ_2). This implies that $a_{T_1}(\mathcal{S})$ or equivalently $[\![V_1]\!]_o$ should contain at least one behavior that is at position $i = 8$ at state $x = 1$ while at position $i = 4$ it is at state $x = 0$. This is not possible since $[\![V_1]\!]_o$ contains only two behaviors: σ_1 which is at state $x = 0$ at both positions $i = 4$ and $i = 8$; and σ_1' which is at state $x = 1$ at both positions $i = 4$ and $i = 8$. Therefore, V_1 and V_2 are inconsistent.

The algorithm cannot detect the inconsistency of V_1 and V_2, however, and reports "inconclusive". This is because at the common positions $0, 4, 8$ the behaviors of the views are both in the same sets of states ($\{0\}$ at position 0, and $\{0, 1\}$ at both positions 4 and 8). Hence, each of the reachable final states of the relevant label driven composition C, contains in the first 2 coordinates the same states.

Example 7. We provide an example run of the algorithm. Consider two views described by the FOS $S_i = (X = \{x, y\}, \theta_i = \neg x \wedge \neg y, \phi_i)$ where the definitions of ϕ_i are indicated in Figure 9, for $i = 1, 2$. Although the two views have the same initial state and the same set of (three) reachable states, they are not identical as their transitions are different. For the remaining exposition, it helps to label the states of S_1 as $1, 2, 3$, and the states of S_2 as a, b, c. Suppose that the views have been obtained with periodic samplings a_{T_i} for $i = 1, 2$ and periods $T_1 = 2, T_2 = 3$ respectively.

Applying **Step 1** of the algorithm we obtain the finite automata L_i for each of S_i for $i = 1, 2$ respectively, as shown in Figure 10. After the determinization **Step 2**, we obtain the deterministic automata shown in Figure 11. The HPA H of **Step 3** coincides with the HPA shown in Figure 4. Figure 12 shows the label driven composition $dL_1 \parallel dL_2 \parallel H$ w.r.t. H of **Step 4**. We observe that there exists a final state $(l_{123}, l_{ab}, 0)$ that $l_{123} \neq l_{ab}$. Hence, according to the **Step 5** of the view consistency algorithm, we obtain that the views S_1, S_2 are inconsistent.

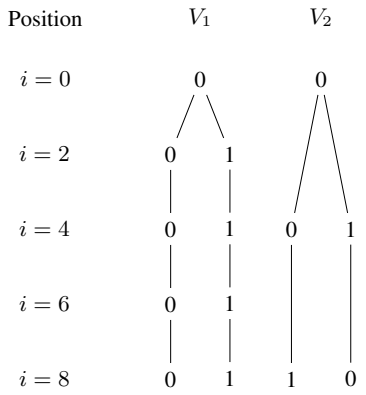
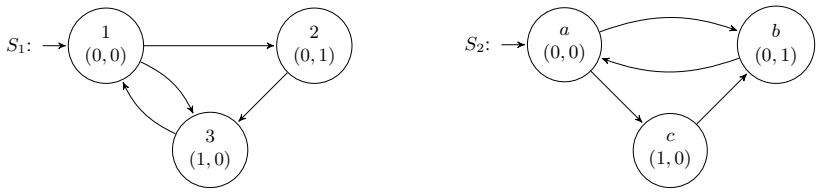
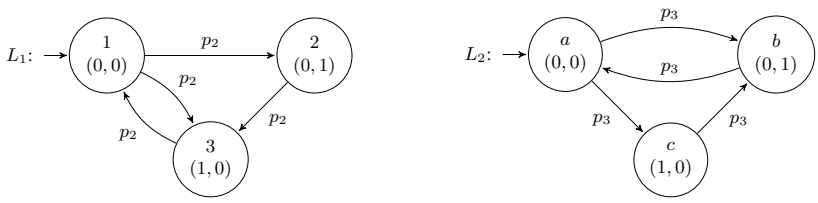


Fig. 8: Behavior trees for views V_1 and V_2.



(a) View S_1. (b) View S_2.

Fig. 9: FOS views S_1 and S_2 .

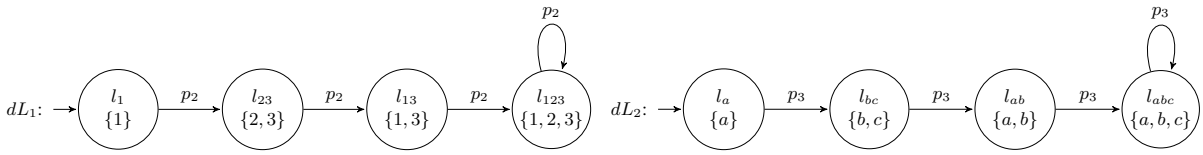


(a) FA L_1. (b) FA L_2.

Fig. 10: **Step 1**: Finite automata L_1 and L_2 obtained from the views S_1 and S_2 of Figure 9 by adding the labels p_2 and p_3 corresponding to their respective periods.

6 Conclusions and Future work

Multi-view modeling is key to systems engineering, as a technique where the use of multiple models/views guides the development of a system. However, one of the crucial issues in multi-view modeling is ensuring consistency among the views. In this work we studied the view consistency problem within the formal framework of [11,10], but for a different type of abstraction functions than those studied previously. In particular, we studied view consistency w.r.t. periodic sampling abstractions.



(a) FA dL_1. (b) FA dL_2.

Fig. 11: **Step 2**: Deterministic FA dL_1 and dL_2 obtained by determinizing the automata L_1 and L_2 of Figure 10.

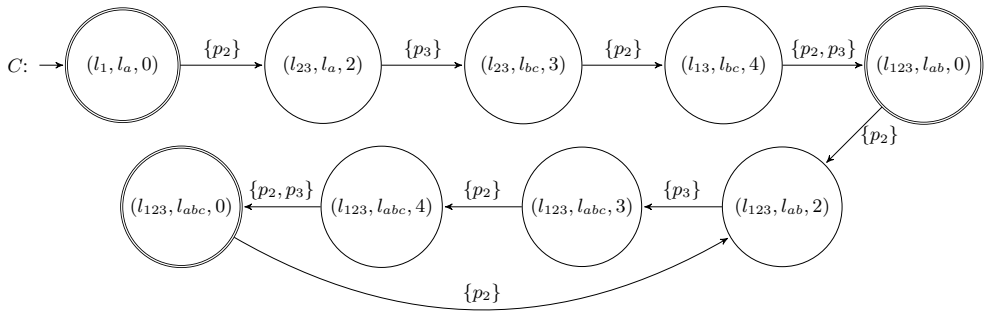


Fig. 12: **Step 4**: Label-driven composition $dL_1 \parallel dL_2 \parallel H$.

The main future work direction is to develop a complete view consistency algorithm. It would be also interesting to answer the question whether Problems 1 and 2 are equivalent, which is currently open. Moreover, we would like to extend our results to handle Problem 3, i.e., to examine under which conditions some given views have a witness fully-observable system. Other future research includes considering other abstraction functions than projections or periodic samplings. Also part of future work is to study heterogeneous instantiations of the multi-view modeling framework, e.g., where some views are discrete, some continuous, some hybrid, and so on. In addition to these theoretical directions, ongoing work includes an implementation of the current framework and experimentation with case studies.

References

1. Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
2. D. Broman, E.A. Lee, S. Tripakis, and M. Törngren. Viewpoints, Formalisms, Languages, and Tools for Cyber-Physical Systems. In *6th International Workshop on Multi-Paradigm Modeling (MPM'12)*, 2012.
3. Sinem Getir, Lars Grunske, Christian Karl Bernasko, Verena Käfer, Tim Sanwald, and Matthias Tichy. Cowolf - A generic framework for multi-view co-evolution and evaluation of models. In *ICMT*, volume 9152 of *LNCS*, pages 34–40. Springer, 2015.
4. John E. Hopcroft and Jeffrey D. Ullman. *Introduction To Automata Theory, Languages, And Computation*. Addison-Wesley, 1990.

5. Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Semantically configurable consistency analysis for class and object diagrams. *CoRR*, abs/1409.2313, 2014.
6. Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman, Matthias Biehl, Stavros Tripakis, Hans Vangheluwe, and Joachim Denil. A characterization of integrated multi-view modeling in the context of embedded and cyber-physical systems. In *EMSOFT*, pages 10:1–10:10. IEEE, 2013.
7. Akshay Rajhans and Bruce H. Krogh. Heterogeneous verification of cyber-physical systems using behavior relations. In *HSCC'12*, pages 35–44. ACM, 2012.
8. Akshay Rajhans and Bruce H. Krogh. Compositional heterogeneous abstraction. In *HSCC'13*, pages 253–262. ACM, 2013.
9. Holger Rasch and Heike Wehrheim. Checking consistency in UML diagrams: Classes and state machines. In *Formal Methods for Open Object-Based Distributed Systems, 6th IFIP WG 6.1 International Conference, FMOODS*, pages 229–243, 2003.
10. Jan Reineke, Christos Stergiou, and Stavros Tripakis. Basic problems in multi-view modeling. Submitted journal version of [11].
11. Jan Reineke and Stavros Tripakis. Basic problems in multi-view modeling. In *TACAS*, volume 8413 of *LNCS*, pages 217–232. Springer, 2014.
12. Aditya A. Shah, Aleksandr A. Kerzhner, Dirk Schaefer, and Christiaan J. J. Paredis. Multi-view modeling to support embedded systems engineering in sysml. In *Graph Transformations and Model-Driven Engineering*, volume 5765 of *LNCS*, pages 580–601. Springer, 2010.
13. S. Tripakis. Compositionality in the Science of System Design. *Proceedings of the IEEE*, 104(5), May 2016.
14. Reinhard von Hanxleden, Edward A. Lee, Christian Motika, and Hauke Fuhrmann. Multi-view modeling and pragmatics in 2020. In *17th Intl. Monterey Workshop*, LNCS, 2012.
15. Xiangpeng Zhao, Quan Long, and Zongyan Qiu. Model checking dynamic UML consistency. In *Formal Methods and Software Engineering*, volume 4260 of *LNCS*, pages 440–459. Springer, 2006.