KATHOLIEKE UNIVERSITEIT
LEUVEN

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science

Reasoning about Hyperproperties

Dimiter VLADIMIROV MILUSHEV

Dissertation presented in partial
fulfilment of the requirements for
the degree of Doctor
in Engineering

June 2013

Reasoning about Hyperproperties

Dimiter VLADIMIROV MILUSHEV

Jury:
Em. Prof. Dr. ir. Willy Sansen, chair
Prof. Dr. Dave Clarke, supervisor
Prof. Dr. ir. Frank Piessens, co-supervisor
Prof. Dr. ir. Wouter Joosen
Prof. Dr. ir. Marc Denecker
Prof. Dr. Gilles Barthe
(IMDEA Software Institute, Spain)
Dr. Alexandra Silva
(Radboud University Nijmegen, The Netherlands)

Dissertation presented in partial
fulfilment of the requirements for
the degree of Doctor
in Engineering

June 2013

D/2013/10.705/43
ISBN 978-90-8649-628-0

To my parents Nadia and Vladimir, for the inspiration, support and infinite belief in me.

Abstract

The importance of security and reliability of software systems makes formal methods of paramount significance for guaranteeing that a system satisfies a particular specification. Hyperproperties can be seen as an abstract formalization of security policies. Because of this, it is desirable to establish a generic verification methodology for at least the class of security-relevant hyperproperties. Unfortunately, such a generic verification methodology is lacking. This is the main motivation of this dissertation.

We observe that most interesting hyperproperties that are relevant in practice come from a class of security-relevant policies, specified using universal and possibly existential quantification on traces, as well as relations on those traces. We formalize such definitions and call them *holistic hyperproperties*. Then our goal becomes to find a methodology for the verification of holistic hyperproperties. To that end, we explore an incremental, coalgebraic perspective on systems and specifications and as a result we arrive at a different, but related kind of specifications: *incremental hyperproperties* (essentially coinductive predicates). Given some holistic hyperproperty H, the respective incremental version is called H' and its definition naturally gives the notion of an *H'-simulation relation*. Such relations enable verification of holistic hyperproperties: finding an H'-simulation relation on a candidate system implies that the incremental hyperproperty H' holds and thus the high-level, holistic hyperproperty H holds for the candidate system. We also introduce techniques that are often helpful in translating holistic hyperproperties into incremental ones.

To show that incremental hyperproperties are important in practice, we explore their connection with the most closely related verification technique — via unwinding. To achieve this, we propose a framework for coinductive unwinding of security relevant hyperproperties based on Mantel's MAKS framework [55] and our work on holistic and incremental hyperproperties. Mantel's MAKS framework cannot be used directly as it is geared towards reasoning about finite behavior and is thus not suitable to reason about holistic hyperproperties in general. However, our framework has a similar structure to MAKS: coinductive unwinding relations compose (or imply) coinductive Basic Security Predicates, which in turn compose a number of security-

relevant, holistic hyperproperties. It turns out that the coinductive unwinding relations we introduce are instances of H'-simulation relations. More importantly, incremental hyperproperties can be expressed in well-behaved logics and this opens the door to their verification.

Finally, we propose a generic verification approach for incremental hyperproperties via game-based model checking. To achieve this, we first show how to interpret incremental hyperproperty checking as games. Although one might do regular model checking of incremental hyperproperties on a transformed system, model checking games are advantageous as they do not only produce a yes-no answer, but also give more intuition about the security policy and what can potentially go wrong, by producing a concrete winning strategy. In order to show that the theory developed here is practical, we present and illustrate methods of using several off-the-shelf tools for verification of incremental hyperproperties expressed in the polyadic modal mu-calculus.

Samenvatting

Formele methodes zijn van groot belang om de veiligheid en betrouwbaarheid van softwaresystemen te kunnen garanderen. Hyperproperties kunnen gezien worden als een abstracte formalisering van een softwarebeveiligingsbeleid. Het is daarom wenselijk om een generieke verificatiemethodologie vast te leggen voor op zijn minst de klasse van hyperproperties die relevant zijn voor beveiliging. Een dergelijke generieke methodologie bestaat helaas nog niet. Dit is de motivatie van dit proefschrift.

We stellen vast dat de interessantste hyperproperties met praktische relevantie deze zijn uit de klasse van softwarebeveiligingsregels die gespecificeerd worden gebruikmakend van universele en mogelijk existentiële kwantificering over traces en de verbanden tussen deze traces. Deze definities worden geformaliseerd en we noemen ze *holistische hyperproperties*. Ons doel wordt vervolgens het vinden van een methodologie om holistische hyperproperties te kunnen verifiëren. Hiervoor onderzoeken we een incrementeel, coalgebraïsch perspectief op systemen en specificaties en vinden een ander maar gerelateerd soort specificaties: *incrementele hyperproperties*. Gegeven een holistische hyperproperty H wordt de respectievelijke incrementele versie H' genoemd en zijn definitie leidt op natuurlijke wijze tot het begrip H'-*simulatierelatie*. Zulke relaties maken de verificatie van beveiligingsgerelateerde hyperproperties mogelijk: het vinden van een H'-simulatierelatie op een kandidaatsysteem impliceert dat de incrementele hyperproperty H' geldig is, waaruit volgt dat de hoog-niveau, holistische hyperproperty H geldig is voor het kandidaatsysteem. We voeren ook technieken in die nuttig kunnen zijn voor het vertalen van holistische naar incrementele hyperproperties.

Om de praktische relevantie van incrementele hyperproperties aan te tonen, onderzoeken we hun verband met de meest verwante verificatietechniek — via unwinding. Daartoe stellen we een raamwerk voor om beveiligingsgerelateerde hyperproperties coindicutief te unwinden. Dit raamwerk is gebaseerd op het MAKS-raamwerk [55] van Mantel en op ons eigen werk over holistische en incrementele hyperproperties. Aangezien het MAKS-raamwerk gericht is op het redeneren over eindig gedrag en

dus niet geschikt is voor het redeneren over holistische hyperproperties, kunnen we het niet rechtstreeks gebruiken. Ons raamwerk heeft evenwel een structuur die vergelijkbaar is met MAKS: coindicutieve unwindingrelaties componeren (of impliceren) coindicutieve Basic Security Predicates die op hun beurt een aantal beveiligingsgerelateerde, holistische hyperproperties samenstellen. Het blijkt dat de coindicutieve unwindingrelaties die we invoeren instanties zijn van H'-simulatierelaties. Belangrijker nog, incrementele hyperproperties kunnen in relatief goed berekenbare logica's worden uitgedrukt, wat toelaat om ze te verificeren.

Tenslotte introduceren we een generieke verificatiemethode voor incrementele hyperproperties, via spelgebaseerde model checking. Hiervoor tonen we eerst aan hoe het controleren van incrementele hyperproperties geïnterpreteerd kan worden als een spel. Alhoewel men standaard model checking van incrementele hyperproperties op een getransformeerd systeem zou kunnen toepassen, heeft spelgebaseerd model checking bepaalde voordelen aangezien dit niet enkel een ja-nee-antwoord geeft maar ook inzicht geeft in het beveiligingsbeleid en wat er mogelijk fout zou kunnen gaan, door een concrete winnende strategie te produceren. Om aan te tonen dat de ontwikkelde theorie bruikbaar is in de praktijk, presenteren en illustreren we methodes die gebruikmaken van bestaande hulpmiddelen voor de verificatie van incrementele hyperproperties uitgedrukt in de polyadische modale mu-calculus.

Acknowledgements

The road to a PhD is a series of (mostly uninformed) choices which makes connecting the dots forward very difficult (paraphrasing Steve Jobs) and potentially results in many ups and downs. I could not have succeeded overcoming the obstacles without the support of all the nice people around me and I hence try to express my gratitude next.

First of all, I am grateful to my supervisor Dave Clarke. Dave, thanks for giving me the opportunity to be a researcher in your team. I deeply appreciate your support, guidance, feedback and encouragement. I know the road has sometimes been quite rough, but the result is worth it! Thanks for giving me the academic freedom to get lost and found, to explore different fields and subfields, to be passionate about research and to work on a topic that I find significant and worth exploring with dedication.

Second, I would like to show my appreciation to all the members of my PhD committee. I am grateful to Wouter Joosen for providing such a wonderful, inspiring and creative atmosphere at DistriNet, as well as a feeling of belonging together, a commitment to a common cause, a vision and a mission for the group. I am also thankful to my co-supervisor Frank Piessens for his extremely positive attitude and readiness to give helpful advice (which I should have searched for more often). I hope that we will be able to explore together at least one of the directions for future work presented in this dissertation. I would also like to thank Marc Denecker, who realized the importance of my research early on and posed a number of interesting questions that are still worth exploring and will hopefully result in substantial scientific contributions. Marc, thanks for the encouragement and interest in the thesis. I also thank Gilles Barthe for his interesting insights on the work and a number of promising ideas for future work. I am grateful to Willy Sansen for being a nice, cheerful and understanding chair of the PhD committee. Last but not least, I am extremely thankful to Alexandra Silva for her critical and thorough review of this dissertation, which resulted in a significant improvement of the presentation and to some extent the content, but most importantly for making me fully aware of the strengths of this dissertation which made me really proud. Alexandra, thank you for the inspiring

feedback and for raising interesting questions which may give a direction to a future academic career.

I cannot overstate how much I owe all my teachers: I want to thank all of you! In particular, I should mention Dieter Gollmann who is responsible for my keen interest in IT security and inspired me to pursue a PhD in the field. I would also like to thank Martin Dwomoh-Tweneboah for the inspiration and encouragement during my college years.

Next, I would like to thank some of my friends and colleagues. José Proença, Dries Vanoverberghe and Dominique Devriese, thanks for being not only good friends, but also always ready for a technical discussion or to give an expert opinion on a draft. I would like to thank Job Noorman, Dimitar Shterionov, Dimitri Van Landuyt, Ansar-Ul-Haque Yasar, Ilya Sergey, Radu Muschevici, Rula Sayaf and Pieter Agten for being good and fun friends and always ready to help. Aram, thanks for being a good friend and bringing a positive atmosphere with numerous talks about football, surf, snowboard and all kinds of other entertaining aspects of life on an almost daily basis :). I am also grateful to Arun Kishore Ramakrishnan, Koosha Paridel and Ping Chen for being good friends and fun (and sometimes funny) office-mates, but also for helping me hone my table-tennis skills. It has always been fun to talk to Marco Patrignani and discuss the "Fundamental(s) exercise" or/and snowboarding and sometimes even research. Thanks also to Adriaan Larmuseau, Nick Nikiforakis, Jan Tobias Mühlberg, Thomas Heyman, Gijs Vanspauwen, Zubair Wadood Bhatti, Francesco Gadaleta, Nelis Boucké, Mario Henrique Cruz Torres, Adriaan Moors, Pieter Philippaerts, Davy Preuveneers, Steven Op de Beeck, Riccardo Scandariato and Raoul Strack for always being friendly and a good and interesting company.

I wish to thank our project office managers Katrien Janssens and Ghita Saevels for being always competent, helpful and friendly. In addition, I am grateful to our immensely helpful administrators Esther Renson, Marleen Sommers, Denise Brams.

This dissertation would have been impossible without the financial support of the European Union co-funded FP7-project NESSoS (contract n. 256980). Thank you!

I would like to also thank my good friend and fishing buddy Janot Castermans. Janot, thanks for introducing me to the Belgian rivers and the good time there, as well as for sharing a passion for good bamboo rods. Thanks for building one of those beautiful pieces of art for me.

Next, I take the time to express my gratitude to my closest people. First, I want to thank my parents Nadia and Vladimir and my sister Maia, who have always been there for me and this victory is as much theirs as mine. Dear Mom and Dad, you are the people I owe the most, I would have not succeeded in this endeavor without your boundless support, encouragement and belief in me! Thank you for the inspiration you gave me, for the example you set and for helping me in each and every way to

become the person I always wanted to be. I know you have sacrificed too much to make sure that your children are fine and have the chance to be happy and successful in life. I know that finishing this work is a very minor return on your investment in me, but I still hope that this dissertation makes you proud and I dedicate it to you! Dear Maia, thanks for always being there when I needed you, for checking on me on a daily basis that all is well! Your moral support, encouragement and enormous belief in me gave me strength in the hardest moments. Thank you! Spending time with you and Marc Adrover helped immensely "recharge the batteries" and be ready for new conquests :). Marc, thanks for your support! It has always been fun (and sometimes funny;)) to be around you! Благодаря ви! Обичам ви!

I thank my grandparents for the cares and wonderful memories from my childhood. To them a few words in Bulgarian: Бабо, дядо, благодаря ви за грижите и за подкрепата, за прекрасните спомени от детството! Обичам ви! Дядо, благодаря ти за това, че ме научи да обичам природата, за хубавите излети в Малешевието и за увлекателните истории и легенди, които разказваш!

Finally, I would like to thank my partner in crime (life). Dear Tanya, thank you very much for your daily encouragement, support and understanding, which were essential to keep me sane during the years (especially during the write-up phase). Thanks for being next to me and for loving me during the good times and the bad ones! Most of all, thank you for being about to make me the happiest person, by giving birth to our little princess! Благодаря ти! Обичам те!

Dimiter Vladimirov Milushev
Leuven, June 2013

Contents

List of Figures

Chapter 1

Introduction

The fast development of information and communication technologies and the proliferation of network-enabled devices has empowered us to conveniently use and share all kinds of data and computing resources. This contemporary "digital era" has radically changed the way we live and interact with each other. In many ways, our day-to-day life depends upon computing and networking technology. All this has brought more convenience, our daily transactions and interactions are considerably simplified; however, there is a price to pay: we have become extremely dependent on the software systems running on this computing infrastructure for most spheres of our life. We can hardly imagine our lives without this "digital cloud", moreover we rely heavily on its proper functioning. The ubiquity and importance of network-enabled devices and computing in all spheres of modern life (ranging from governmental transactions to personal, online banking) has made us dependent on the security and reliability of software systems. In addition, issues of online privacy are only starting to emerge, as more and more people realize they are neither in control of their data nor of the digital traces of their actions on different networks. Thus, the problem of verification of a system with respect to (security) policy specifications is becoming increasingly more important.

Informally, a security policy can be seen as a specification of what is the normal behavior of a system and what is prohibited. Until recently, there was no mathematical formalization of security policies in general. This made the problem of verification very difficult, as every individual system had to be verified with respect to a concrete application-specific policy in a potentially different manner. That is why it became important to invent a formal classification of security policies, in the hope of obtaining verification methodologies corresponding to different types of security policies. Such a formal classification of security policies as properties and hyperproperties was

proposed by Clarkson and Schneider [18, 19]. Unfortunately, a generic verification methodology for at least the security-relevant hyperproperties does not exist. This dissertation starts a quest towards such a verification methodology.

1.1 A Historical Perspective

The problem of verifying that a system adheres to a given specification has been an active research area for several decades. Substantial progress has been made in the verification of policies that can be expressed as *trace properties*—first-order predicates over system execution traces. Trace properties are well-understood: every trace property can be seen as the intersection of a *safety property*, guaranteeing that "nothing bad" happens and a *liveness property*, guaranteeing that "something good" eventually happens, in any trace for which the property holds [5]. Well-known examples of successful verification methodologies include the model checking techniques for an abundance of logics for system specification [15, 16, 51, 43, 80, 81]. Moreover, security policies that are safety properties, such as access control, can be enforced at runtime by a technique called *execution monitoring* [79]. This technique is based on encoding the property as a so-called security automaton and each step of the application is allowed if and only if the automaton can also produce the same step.

Unfortunately, trace properties are not expressive enough to capture a large class of security-relevant policies, such as the abundant variants of information flow and noninterference policies, as well as some quality of service specifications. Information flow policies try to regulate the direct and indirect flow of information between subjects (typically software running on behalf of users). Other definitions focus on implicit flow of information. For instance, the abundant variants of noninterference policies on systems essentially try to prevent the possibility of inferring authorized information from non-authorized information [32]. According to the original definition by Goguen and Meseguer [31], a system is secure with respect to noninterference if and only if any sequence of low inputs results in the same low outputs, no matter how one varies the high level inputs. Finally, quality of service (QoS) specifications define the expected performance of systems with respect to certain parameters, such as response time, availability, etc. [18, 19]; the QoS specifications based on inspecting and relating many, potentially all execution traces in a system cannot be expressed as properties.

In an attempt to provide an expressive, uniform theory of system specifications, Clarkson and Schneider formalized security policies as *hyperproperties* [18, 19]. A *hyperproperty* is a set of trace properties or, alternatively, a set of sets of infinite execution traces. Hyperproperties generalize properties and are expressive enough to capture the known notions of noninterference and secure information flow [18, 19].

Intuitively, a hyperproperty is the set of systems permitted by some policy. Although arising in the context of security, hyperproperties are not necessarily limited to security policies; they can be seen as very general and expressive system specifications, as witnessed by a number of quality of service specifications mentioned above.

Clarkson and Schneider generalize safety and liveness properties to *safety hyperproperties* and *liveness hyperproperties*, also known as *hypersafety* and *hyperliveness* [18, 19]. The main difference with safety and liveness is that traces are lifted to sets of traces: for instance, the "bad thing" is now a number of finite traces, whose existence makes the system necessarily and irremediably insecure. Similarly, the "good thing" means that any set of finite traces can be extended to a set of infinite traces that satisfy certain condition(s). Clarkson and Schneider also show that every hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty. In addition, the notion of a *k-safety hyperproperty* is introduced based on a generalization of Terauchi and Aiken's 2-safety properties [84]: essentially a 2-safety property can be verified by examining all 2-tuples of traces in a system. Similarly, a k-safety hyperproperty can be verified by examining all k-tuples of traces in a system. Finally, Clarkson and Schneider propose a verification methodology for k-safety hyperproperties based on invariance arguments.

1.2 The Problem: Lack of a Generic Verification Methodology for Hyperproperties

Unfortunately, a verification methodology for k-safety hyperproperties is not sufficient simply because many interesting hyperproperties are not k-safety hyperproperties. For instance, Clarkson and Schneider argue that all possibilistic information flow policies are not safety hyperproperties [18, 19]. In addition, they show that it is possible for a security definition (for instance that of relational noninterference [18, 19]) to be a safety hyperproperty in its termination-insensitive variant and not a safety hyperproperty in its termination-sensitive reincarnation. Hence, one may conclude that a significant number of interesting security-relevant hyperproperties cannot use the proposed verification technique.

In this dissertation, we start a quest for a generic verification methodology for security-relevant hyperproperties that is agnostic to the hypersafety/hyperliveness classification. In other words, the goal is to find techniques that work for safety and liveness hyperproperties alike. We next demonstrate that some of the most intuitive ideas about approaching the problem are not adequate. Nevertheless, the solution proposed in this work will result from adapting some of these ideas.

The most obvious idea would be to try and reuse existing methodologies for

verification of properties, for instance the ones based on model checking properties expressible in the modal mu-calculus [43] and its derivatives. Unfortunately, this is not immediately feasible: it is well-known that the modal mu-calculus and its related tree logics, such as CTL and CTL*, are not expressive enough for hyperproperties [7]. Intuitively, the reason is that they cannot express specifications relating several paths in an execution tree, yet the original notion of hyperproperties is based on the existence of such relations (see [19, 63]). The first ones to formalize this observation were Alur et al. [7] who presented a proof that secrecy, a policy defined as the existence of uncertainty as to whether a particular property is true or not, is not expressible in the modal mu-calculus. The proof can be straightforwardly adapted to show that the modal mu-calculus and the related tree logics are not expressive enough for hyperproperties.

Another intuitive idea would be to explore unwinding, a verification technique proposed back in the 80s by Goguen and Meseguer in their seminal work [32]. Unfortunately, instead of a generic verification methodology, the work on unwinding results in a number of specific verification methodologies for noninterference-based definitions of security [32, 37, 72, 61, 74]. For each definition, there is an unwinding theorem stating that the unwinding relation on the state space of the system implies the high-level policy of interest or is equivalent to it (see Figure 1.1). There is no hint as to how to approach the problem of getting an unwinding relation from some new policy. The problem with such an approach is that every new high-level specification needs to

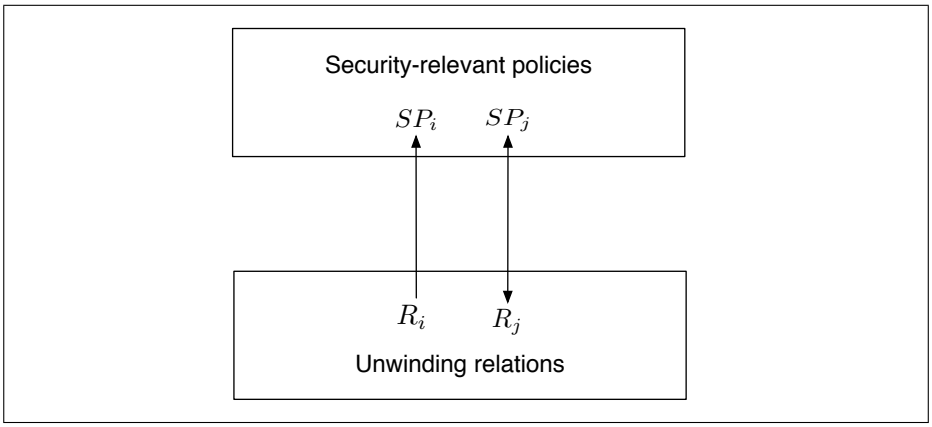


Figure 1.1: Illustration of the original unwinding relations

be examined in its own right and a specific verification method needs to be invented. Further, the proofs have many commonalities, which do not seem to depend on the particular notion of security. Mantel was the first to notice (and report) this and factor out commonalities in unwinding theorems, which lead to his Basic Security Predicates (BSPs) [55]. The BSPs can be seen as "building blocks" of the noninterference-

based possibilistic security policies. The BSPs themselves can be composed from or implied by unwinding relations. Based on these observations, Mantel proposed his Modular Assembly Kit for Security Properties (MAKS) framework, which is suitable for reasoning about noninterference-like policies on finite systems. The framework is illustrated in Figure 1.2.

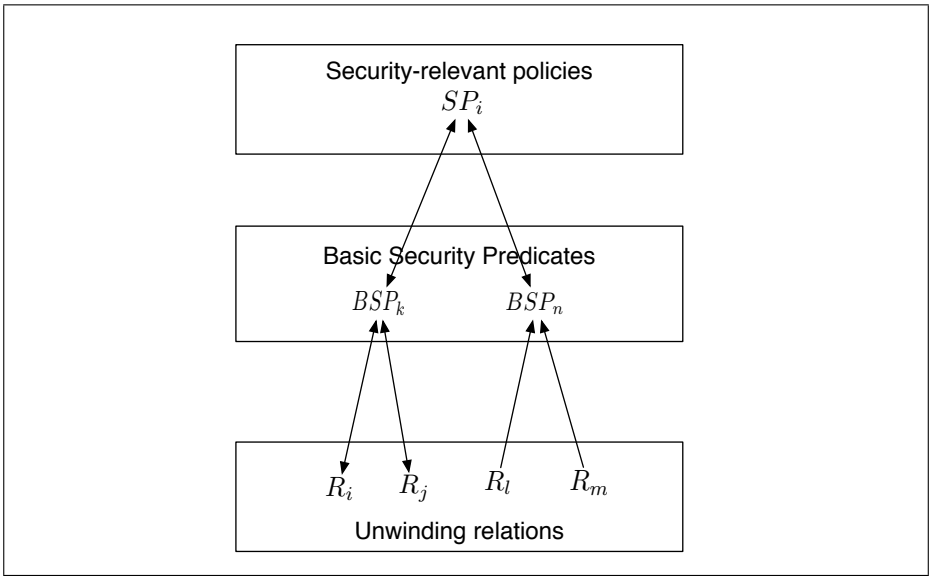


Figure 1.2: Illustration of the structure of Mantel's MAKS framework [55]

One might think that reusing Mantel's ideas would lead to a solution to our problem of finding a generic verification methodology for hyperproperties. However, his framework was developed for systems modeled as sets of finite traces, i.e. necessarily terminating, whereas hyperproperties (can) have infinite traces. Moreover, liveness hyperproperties are necessarily defined as sets of infinite traces. The reason is that the "good thing" about a set of observations should be "always possible" and "possibly infinite" [18, 18]. This implies that some work needs to be done if we were to adapt this framework to reason about hyperproperties. How much work is needed will become clear in Chapter 4.

1.3 Our Solution: Incremental Hyperproperties, a Logic and a Verification Methodology

This dissertation does not propose a generic verification methodology for hyperproperties. Instead, it proposes formal definitions of the classes of holistic and incremental hyperproperties as well as a generic verification methodology for incremental hyperproperties. The class of holistic hyperproperties (HHPs) formalizes the typical security-relevant specifications of hyperproperties defined by relations on whole traces, as well as universal and possibly existential quantification over those traces, whereas the class of incremental hyperproperties (IHPs) are essentially local, coinductive predicates on the state spaces of candidate systems. A process of transforming holistic hyperproperties into incremental ones is also suggested. This process is called *incrementalization*. The problem with incrementalization is that it is not always trivial or may not even be possible to convert a hyperproperty specification into an equivalent incremental one. However, it is typically possible to at least have an incremental specification implying the higher-level one.

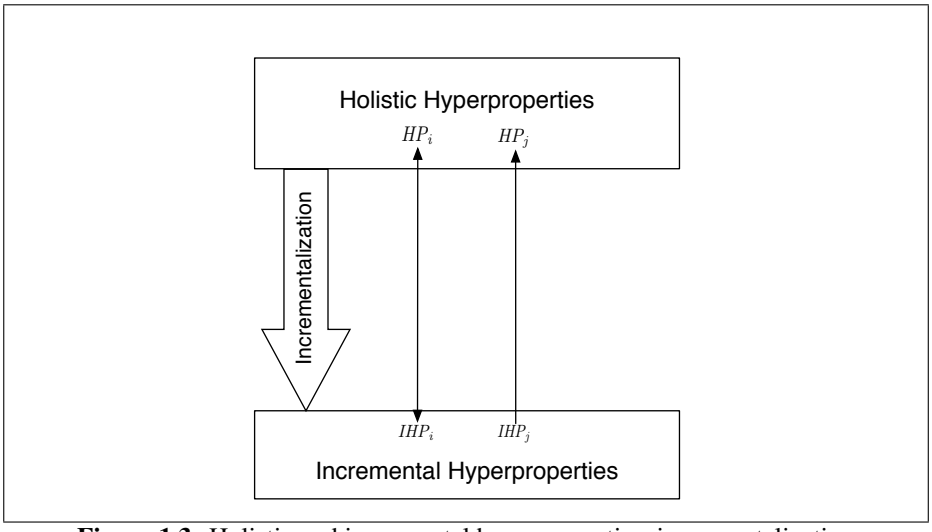


Figure 1.3: Holistic and incremental hyperproperties, incrementalization

The proposed solution and methodology are next described in more depth. We start off in Chapter 3 by exploring an incremental, coalgebraic perspective on systems and specifications as an alternative to typical holistic specifications. This results in formalizing *incremental* and *holistic* hyperproperties. Whereas holistic hyperproperties are based on relations on whole traces of execution as well as universal and possibly existential quantification over those traces [63], incremental hyperproperties are coinductive predicates on the state-space of the system of interest,

or alternatively on the resultant trees of executions. For each IHP, H', we introduce the notion of an *H'-simulation* — a relation on the state space of the system whose existence implies that the IHP H' (and thus also H) holds for the candidate system. H'-simulation relations can be seen as a generalization of bisimulation relations. The former relations are more general as they can be arbitrary, i.e. they need not be equivalence, symmetric, or reflexive relations. More importantly, the existence of an H'-simulation implies (or is equivalent to) that a holistic hyperproperty H holds, thus the relations enable us to indirectly reason about holistic specifications. Also in Chapter 3, we present some incrementalization techniques, which could be helpful in translating HHPs into IHPs.

H'-simulations turn out to be closely related to Mantel's unwinding conditions, a topic that is explored in Chapter 4. As a result of this relation, we will be able to show that a large class of possibilistic information flow hyperproperties can be verified via H'-simulations; in addition, we will argue that incremental hyperproperties indeed do capture a large class of security-relevant policies. Moreover, H'-simulations are a novel, more general kind of coinductive predicates that go beyond the state-of-the-art at the time (circa 2009), namely the (essentially pointwise) coinductive predicates proposed by Niqui and Rutten [66]. The importance of the use of coinduction in a "non-standard setting" (beyond proofs of bisimilarity and equality [45]) and of general coinductive predicates similar to H'-simulations has been reaffirmed by more recent results [45, 38].

Another advantage of incremental hyperproperties is that they can be expressed in relatively well-behaved logics, which are fragments of least fixed point logic (LFP) [13]. This opens the door to verification (explored in Chapter 5), as it turns out that one particular fragment of LFP — the polyadic modal mu-calculus [8] — is expressive enough for all IHPs we are aware of. Finally, we propose a generic verification approach for incremental hyperproperties via model checking. To that end, we first show how to interpret incremental hyperproperty checking as games in Chapter 5. Although one might do regular model checking of IHPs on a transformed system, model checking games are advantageous as they do not only produce a yes-no answer, but also give more intuition about the security policy and what can potentially go wrong, by producing a concrete winning strategy. These problems are explored in Chapter 6. In order to show that the theory developed here is practical, we have illustrated and compared the use of several tools for verification of concrete incremental hyperproperties in Chapter 6.

1.4 Outline and Main Contributions

This section provides an outline of the chapters of the dissertation and states the contributions of each chapter. Further, it lists the related publications.

Chapter 2 provides the necessary background material used throughout the dissertation. We start by introducing some basic lattice theory, needed to understand coinduction and coinductive definitions, which are used throughout this work. We then present a model of systems as sets of traces and an abstract formalization of security policies as properties and hyperproperties due to Clarkson and Schneider [18, 18]. Then, important details of Mantel's MAKS framework are summarized. To change perspective and get an alternative, incremental view on policy specification and verification, we need and present the notions of partial automata, coalgebra and languages [73]. The incremental specifications are formalized by giving a logical language, based on first order logic, its extension least fixed point logic and the polyadic modal mu-calculus [8]. Therefore, we present the syntax and semantics of these logics. Finally, we introduce a notion of games used in this work.

Chapter 3 presents the notions of holistic and incremental hyperproperties. Holistic hyperproperties are a formalization of the specifications of security-relevant hyperproperties, whereas incremental hyperproperties are coinductive predicates on system behavior. The motivation for such a distinction is that holistic specifications tend to be more intuitive but difficult to reason about, whereas incremental specifications have a more feasible verification approach. Typically, a holistic hyperproperty H has a corresponding incremental hyperproperty H', such that H' implies or is equivalent to H. We introduce and illustrate the respective notion of H'-simulation relation, which facilitates verification; finding an instance of such a relation on a particular system implies that the system satisfies the respective hyperproperty H. H'-simulations are a class of coinductive predicates that goes beyond pointwise coinductive predicates (also generalizing bisimulation) proposed by Niqui and Rutten [66]. Moreover, H'-simulations are interesting as they have a security-relevant application. It is noteworthy that at the time of their proposal, our general coinductive predicates (H'-simulations) were novel. In recent papers [38, 45], the importance of similar, more general than bisimulation coinductive predicates and non-standard coinductive reasoning has been reaffirmed.

As most interesting security-relevant hyperproperties are in the syntactic class of holistic hyperproperties, we also introduce the process of incrementalization and use it to convert holistic specifications into incremental ones. *Incrementalization* is a set of techniques that can be used to convert holistic specifications into incremental ones. Most importantly, we propose a generic framework and techniques to explore the process of incrementalization and the usefulness of the resulting incremental hyperproperties. We then use the framework to explore three incrementalizable classes

of holistic hyperproperties and their respective incremental definitions. Finally, a verification method for incremental hyperproperties is presented.

Next, Chapter 4 realizes a connection between incremental hyperproperties and the most closely related verification technique — via unwinding relations. More concretely, the chapter proposes a framework for coinductive unwinding of security relevant hyperproperties based on Mantel's MAKS framework [55] and our work on holistic and incremental hyperproperties. Unwinding relations have been widely used to prove that finite (i.e. terminating) systems are secure with respect to a variety of noninterference policies. As hyperproperties are defined on potentially infinite systems (and the good thing in a liveness hyperproperty is "always possible" and "potentially infinite" [18, 18]), a new mathematical development is needed in order to (re)use unwinding relations for generic verification of security-relevant hyperproperties. This new development is based on coinduction: the technique of choice for reasoning about infinite behavior [77]. This is not straightforward since Mantel's definitions of security policies typically refer inductively to the last confidential event in a trace. This works well for finite traces, but unfortunately not for infinite traces, because it is possible and common to have such traces having confidential events occurring infinitely often. Traces of this kind are ignored by the aforementioned security policy definitions. Hence, we need to redefine (taking a coinductive reinterpretation of) the policies themselves and thus the respective BSP definitions and unwinding relations. We do this by making all definitions coinductive, which fixes the problem in a natural way and instead of referring to the last confidential element we now refer to any such element in a trace. This is challenging, as we would like to keep the framework and the type of results it offers, but at the same time the definitions need to be changed. It is also not initially clear that such an approach would work. However, we are able to present results demonstrating that our alternative framework not only meets our specific requirements, but also has advantages similar to the ones of MAKS.

In essence, we illustrate that Mantel's BSPs, the noninterference policies they compose, as well as their respective unwinding relations, have a meaningful coinductive reinterpretation. We show that in a number of cases the coinductive variants of the unwinding relations imply the respective coinductive variants of the BSPs. Moreover, the latter can be used to compose high-level security-relevant hyperproperties for both finite and infinite systems. The unwinding theorems we have considered hold as expected. We argue that the proposed framework and results are useful both theoretically in the study of hyperproperties and in practice for verification of hyperproperties on infinite systems that cannot be approximated by finite ones. Further, we demonstrate that our methodology can be used to reason about a large class of holistic hyperproperties (via incremental ones), such as the relatively large class of possibilistic information flow hyperproperties, which are instances of liveness hyperproperties. This is a significant improvement over the state of the

art: Clarkson and Schneider present a verification technique based on invariance arguments for k-safety hyperproperties and argue that it does not work for liveness hyperproperties [19]. We show that verification via H'-simulations can deal with some liveness hyperproperties. The problem whether our techniques can deal with all liveness hyperproperties is open and left for future work.

Chapter 5 demonstrates that incremental hyperproperties have a clear verification methodology by elaborating on a decidable model checking approach. We propose a logic suitable for expressing the known incremental hyperproperties. This logic is based on the polyadic modal mu-calculus [8] and has a decidable model checking problem for finite systems. A novel characterization of the satisfaction relation between a system and an incremental hyperproperty in the logic is proposed in terms of playing a game. The games are called *incremental hyperproperty checking games* (IHP checking games) and the correctness of such games is proven. Moreover, an approach for converting these games into parity games is presented, relating the problems of model checking IHP checking games and parity games. For both IHP checking and parity games, finding a winning strategy for the verifier implies that the incremental hyperproperty holds, whereas, finding a winning strategy for the refuter implies that the incremental hyperproperty does not hold. Most importantly, this chapter bridges the gap between the problem of model checking incremental hyperproperties and the extensive work on model checking games for fixed point logics.

Next, Chapter 6 presents several different approaches that can be used to model check incremental hyperproperties. The first approach is based on a reduction of the problem of model checking a formula in the polyadic mu calculus on a k-tuple of systems to model checking a modified formula in the ordinary modal mu-calculus [43] on a modified version of the original system. This means that one can use off-the-shelf model checking tools (e.g. the mCRL2 toolset [34]) for verification of incremental hyperproperties, as long as there is a convenient way to create the modified version of the original system. Unfortunately, when the resultant system is not a model of the formula in question, it is difficult to analyze where the problem lies. The other approaches make such an analysis more intuitive and generally easier, because they are phrased in terms of winning strategies. The second approach is based on converting an IHP checking game into a parity game and directly solving the parity game using an off-the-shelf tool such as PGSolver [29]. The third approach offers more automation and is again based on model checking of a modified system with respect to a modified specification formula, but this time via games. The second and third approaches have the advantage that they are game-based and the respective model checking algorithms generate winning strategies [82]. This is beneficial, as one may use such a winning strategy to get intuition about the interaction between a system and a policy and, even more importantly, to get a precise reason as to what goes wrong and why a formula does not hold on a system. In case the formula holds, insight as to why

this is the case on the concrete system can be given too. The use of such techniques and visualizations of games may result in tools with intuitive debugging functionality. To further demonstrate the advantages of model checking via games we propose two different graph views, namely an *extended game graph view* and a *tree view* that can be used for better visualization of strategies. An extended game graph view enhances the game graph with the winning strategy of the winner from the initial node and also presents information about winning positions for both players. A tree view is a list of the different positions in the history of the play so far. It turns out that tree views are a very intuitive representation of the game. In addition, we present a known idea for a strategy-based interactive tool [83] (here based on the extended game graph and tree views) that could be useful in our setting for understanding the interaction of systems and security policies. We note that the main difference between the two game-based approaches explored in this chapter is the degree of automation: the third one has the potential for being fully automatic.

Finally, Chapter 7 summarizes the contributions and presents some promising directions for future work.

The dissertation is based on the following research papers and technical reports:

- Dimiter Milushev and Dave Clarke. Towards incrementalization of holistic hyperproperties. In Pierpaolo Degano and Joshua D. Guttman, editors, proceedings of the *First International Conference on Principles of Security and Trust (POST 2012)*, volume 7215 of *Lecture Notes in Computer Science*, pages 329-348. 24 March – 1 April 2012. Tallinn, Estonia. Springer.
- Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties. In Audun Jøsang and Bengt Carlsson, editors, proceedings of the *17th Nordic Conference on Secure IT Systems (Nordsec 2012)*, volume 7617 of *Lecture Notes in Computer Science*, pages 121-136. 31 October – 2 November 2012. Karlskrona, Sweden. Springer.
- Dimiter Milushev and Dave Clarke. Decidable incremental hyperproperty logics and model checking via games. Submitted for publication.
- Dimiter Milushev and Dave Clarke. Incremental hyperproperty model checking via games. Submitted for publication.
- Dimiter Milushev and Dave Clarke. Towards incrementalization of holistic hyperproperties: extended version. *CW Reports, volume CW616*, 32 pages, Department of Computer Science, KU Leuven. December 2011. Leuven, Belgium.
- Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties: extended version. *CW Reports, volume CW623* 28 pages, Department of Computer Science, KU Leuven. August 2012. Leuven, Belgium.

1.5 Other Contribution

Besides the contributions described in this dissertation, the author contributed to the following research result during his PhD studies:

- Dimiter Milushev, Wim Beck and Dave Clarke. Noninterference via symbolic execution. In Holger Giese and Grigore Rosu, editors, proceeding of the *IFIP International Conference on Formal Techniques for Distributed Systems joint international conference 14th Formal Methods for Open Object-Based Distributed Systems 32nd Formal Techniques for Networked and Distributed Systems (FMOODS & FORTE 2012)*, volume 7273 of *Lecture Notes in Computer Science*, pages 152-168. 13–16 June 2012. Stockholm, Sweden. Springer.

Chapter 2

Background

This chapter provides the necessary background material and formal notation for the remainder of the thesis. We start by presenting some basic lattice theory. Based on it, we introduce coinductive definitions, which are widely used throughout this work. We then present a typical model of systems as sets of traces, followed by an abstract formalization of security polices in terms of properties and hyperproperties due to Clarkson and Schneider [18, 19]. Some of the ideas in the work of Clarkson and Schneider are closely related to Mantel's noteworthy work on verification via unwinding in his MAKS framework [55]. Hence, unwinding relations and the framework are briefly introduced. Viewing systems as sets of traces or languages is equivalent to viewing them as trees, moreover both trees and sets of traces can be seen as the system behavior generated by partial automata. These are three equivalent views on systems (sets of traces, trees and partial automata) and we often explicitly or implicitly switch between them. Thus some theory about partial automata, coalgebra and languages [73], needed to explain the views, is presented here. We then recall the syntax and semantics of first-order logic (FOL), its extension with fixed points called least fixed point logic (LFP) and the polyadic modal mu-calculus [8]; these logics provide the basics for some of the new logical languages, used to formalize hyperproperties. Finally, the chapter ends by presenting the notion of games, as used in this work.

2.1 Some Lattice Theory

This section presents some results from lattice theory [21].

Definition 2.1.1. A *partially ordered set* (poset) is a set P together with a binary relation $\leq \subseteq P \times P$ such that for all elements $x, y, z \in P$ we have

- $x \leq x$ — reflexivity
- $x \leq y$ and $y \leq z$ implies $x \leq z$ — transitivity
- $x \leq y$ and $y \leq x$ implies $x = y$ — antisymmetry

A well-known example of a poset is the set of natural numbers with the usual ordering relation ($\leq$). Another one is the set of real numbers with the same relation (now on real numbers).

Definition 2.1.2. Let P be a poset and $S \subseteq P$. An element $x \in P$ is an *upper bound* of S if $s \leq x$ for all $s \in S$. The dual notion is that of a *lower bound* of S: an element $x \in P$ such that $s \geq x$ for all $s \in S$.

Definition 2.1.3. Let P be a poset. A least element of a subset $S \subseteq P$ is an element $x \in S$ that is a lower bound of S. The *least upper bound* y of S is an upper bound such that for all upper bounds z of S, $z \geq y$. The least upper bound of S is also called the *join* of S. The dual of the definition gives the *greatest lower bound* of S, i.e. the *meet* of S.

An *endofunction* on a set S is a function from S to itself.

Definition 2.1.4. Let P be a poset. Let F be an endofunction on P.

- F is said to be *monotone* if $x \leq y$ implies $F(x) \leq F(y)$ for all $x, y \in P$.
- An element $x \in P$ is a *pre-fixed point* of F if $F(x) \leq x$.
- An element $x \in P$ is a *post-fixed point* of F if $x \leq F(x)$.
- A fixed point of F is an element $x \in P$ that is both a pre-fixed point and a post-fixed point of F, i.e. $F(x) = x$.
- The least and greatest elements in the set of fixed points (if they exist) are called *least fixed point* and *greatest fixed point* of F, respectively.

Definition 2.1.5. Let P be a poset. If every subset $S \subseteq P$ has both a join and a meet, then P is called a *complete lattice*.

The complete lattice that is most interesting for us throughout this work is described next. Let S be a set. The powerset of S, denoted 2^S is a complete lattice with respect to the set inclusion relation $\subseteq$. The bottom element of the complete lattice is the

empty set $\emptyset$, the top element is S. The join is given by set union and the meet by set intersection.

The following theorem guarantees the existence of a least and a greatest fixed point of a monotone endofunction on a complete lattice.

Theorem 2.1.6. *[Knaster-Tarski Fixed Point Theorem] Let L be a complete lattice and F a monotone endofunction. The least fixed point of F exists and is the meet of all pre-fixed points of F. Dually, the greatest fixed point of F exists and is the join of all post-fixed points of F.*

On complete lattices generated by the powerset construction, we have the following interpretation of Theorem 2.1.6. If $F : 2^X \to 2^X$ is monotone, then:

$$\mathbf{lfp}(F) = \bigcap \{S \mid F(S) \subseteq S\} \text{ and}$$

$$\mathbf{gfp}(F) = \bigcup \{S \mid S \subseteq F(S)\}.$$

2.2 Inductive and Coinductive Definitions

The material presented in this section is adapted from Sangiorgi's book [76].

Definition 2.2.1 (Inductively and Coinductively Defined Sets)**.** Given a complete lattice L with points being sets (for instance, the complete lattice of the powerset) and a monotone endofunction $F : L \to L$, the sets inductively and coinductively defined by F are respectively given as follows:

$$F_{ind} = \bigcap \{x \mid F(x) \leq x\}, \text{ the meet of pre-fixed points, and}$$

$$F_{coind} = \bigcup \{x \mid x \leq F(x)\}, \text{ the join of post-fixed points.}$$

Corollary 2.2.2 (Induction and coinduction proof principles)**.** *The following sound proof principles follow from Definition 2.2.1:*

- *The induction proof principle: if $F(x) \leq x$ then $F_{ind} \leq x$.*
- *The coinduction proof principle: if $x \leq F(x)$ then $x \leq F_{coind}$.*

When F is monotone, by Theorem 2.1.6 we know that $F_{ind} = \mathbf{lfp}(F)$ and $F_{coind} = \mathbf{gfp}(F)$. Moreover, the meet of pre-fixed points is also a pre-fixed point and dually the join of post-fixed points is also a post-fixed point. Hence, for a monotone endofunction F, Corollary 2.2.2 is transformed into the following:

Corollary 2.2.3 (Induction and coinduction proof principles for monotone functions)**.** *The following proof principles are sound:*

- *The induction proof principle: if $F(x) \leq x$ then $\mathbf{lfp}(F) \leq x$.*
- *The coinduction proof principle: if $x \leq F(x)$ then $x \leq \mathbf{gfp}(F)$.*

2.2.1 Rule-based Definitions

A *ground rule* on some set X is a pair (S,x), where $S \subseteq X$ and $x \in X$. The inductive interpretation is that from the premises in S we can conclude x. The coinductive one is that x can be observed and reduced to the set S.

Lemma 2.2.4 ([76])**.** *Any set of ground rules $\mathcal{R}$ on a set X produces a monotone endofunction (also called functional) $\Psi_{\mathcal{R}}$ on the complete lattice of 2^X, where $\Psi_{\mathcal{R}}$ is given as:*

$$\Psi_{\mathcal{R}}(T) = \{x \mid (T',x) \in \mathcal{R} \text{ for some } T' \subseteq T\}.$$

Since the functional $\Psi_{\mathcal{R}}$ is monotone, we know (from Theorem 2.1.6) that both the greatest ($\mathbf{gfp}(\Psi_{\mathcal{R}})$) and the least fixed point ($\mathbf{lfp}(\Psi_{\mathcal{R}})$) exist. The coinductive proof principle is then given as follows:

$$T \subseteq \Psi_{\mathcal{R}}(T) \text{ implies } T \subseteq \mathbf{gfp}(\Psi_{\mathcal{R}}).$$

This means that the coinductive hypothesis is that T is a post-fixed point of $\Psi_{\mathcal{R}}$: for all x in T, there must be some rule (S,x) in $\mathcal{R}$ such that $S \subseteq T$. If this is the case, we may conclude that $T \subseteq \mathbf{gfp}(\Psi_{\mathcal{R}})$, as we know that $\mathbf{gfp}(\Psi_{\mathcal{R}})$ is the join all post-fixed points (see Theorem 2.1.6).

Example 2.2.5 (The Set A^* of Finite Lists over Alphabet A)**.** *Before presenting the example, we need a definition of derivative. For elements $a \in A$ and sets $L \subseteq A^\infty$, define the a-*derivative *of L as $L_a = \{v \in A^\infty \mid a \cdot v \in L\}$. We often use the notation $L \xrightarrow{a} L'$, meaning that $L' = L_a$. Consider the set of finite lists over A. We can define the set A^* inductively by the following rules:*

$$\frac{}{\varepsilon \in A^*} \qquad \frac{L \xrightarrow{a} L' \qquad L' \in A^*}{L \in A^*}$$

The inductively defined set A^* specifies the objects that can be obtained with a finite proof using the above-presented rules (note that the metavariables L and L' are implicitly universally quantified). Alternatively, A^* is the set closed forward, i.e. reading the rules in the usual direction. The empty list ε is in A^* and if there is $L' \in A^*$ such that $L \xrightarrow{a} L'$ for some $a \in A$, then $L \in A^*$. Note that $\mathcal{R}_{A^*}$ for the rules above can be given as follows:

$$\mathcal{R}_{A^*} \mathrel{\widehat{=}} \{(\emptyset, \varepsilon)\} \cup \{(\{L'\}, L) \mid L \xrightarrow{a} L' \text{ for some } a \in A\}.$$

The respective functional is

$$\Psi_{\mathcal{R}_{A^*}}(T) \mathrel{\widehat{=}} \{L \mid L \text{ is the empty list or there are } L' \in T \text{ and } a \in A \text{ s.t. } L \xrightarrow{a} L'\}.$$

The set closed forward is the pre-fixed point of $\mathcal{R}_{A^*}$, i.e. the least fixed point of $\Psi_{\mathcal{R}_{A^*}}$.

Example 2.2.6 (The Set A^∞ of Finite and Infinite Lists over Alphabet A)**.** *Next consider the set of finite and infinite lists over A. We can define the set A^∞ coinductively by the following rules:*

$$\frac{}{\varepsilon \in A^\infty}\ \text{coind} \qquad \frac{L \xrightarrow{a} L' \qquad L' \in A^\infty}{L \in A^\infty}\ \text{coind}$$

This is the set of all objects that can be obtained with a finite or infinite proof using the rules above. Note that although the rules are the same, the defined sets are different. The difference is that in this example we take the greatest fixed point interpretation of the rules. That is why each inference rule is denoted *coind* on its right-hand side. In practice, this means we need and use the rules backwards: an element of the set A^∞ is either the empty list ε or if $L \in A^\infty$ then there is some $L' \in A^\infty$ such that $L \xrightarrow{a} L'$. Alternatively, this can be described as the largest set closed backwards under the rules.

2.3 A Formal Model of Systems

At a high level of abstraction, a system can be modeled as a set of finite or infinite sequences of events/actions/observations. These notions are formalized next.

Let us fix a finite alphabet A of abstract observations, also called events or actions. Let 2 be any two element set, for instance the one given as $2 = \{true, false\}$. The symbol ω denotes the set of non-negative integers. If X and Y range over sets, notation Y^X stands for the set of functions with signature $X \to Y$.

Definition 2.3.1. A *string* is a finite sequence of elements of A. The set of all strings over A is denoted A^*.

Definition 2.3.2. A *stream* of A's is an infinite sequence of elements of A. The set of all streams over A is $A^{\omega} = \{\sigma \mid \sigma : \{0,1,2,\ldots\} \to A\}$.

A stream σ can be specified in terms of its first element $\sigma(0)$ and its stream derivative σ', given by $\sigma'(n) = \sigma(n+1)$ [73] for all $n \geq 0$; these operators are also known as *head* and *tail*. In addition, $\sigma(i)$ gives the i-th element in the stream.

Definition 2.3.3. A *trace* is a finite or infinite sequence of elements of A. The set of all traces over A is denoted $A^{\infty} = A^* \cup A^{\omega}$.

Definition 2.3.4. A *system* is a (non-empty) set of traces. The set of all systems is $\mathsf{Sys} = 2^{A^{\infty}}$, the set of infinite systems is $\mathsf{Sys}_{\omega} = 2^{A^{\omega}}$.

Let Sys^n denote the n-ary Cartesian product of Sys.

2.4 Abstract Formalization of Security Policies

Clarkson and Schneider proposed an abstract formalization of security policies based on properties and hyperproperties [18, 19], which we define and illustrate next.

Generally speaking, a property is a Boolean function on objects and the *extension* of a property is the set of objects for which the function evaluates to true. The extension of a property (of single execution traces) is the *set of traces* for which the property holds, and in that sense a property is called a set of traces [5].

Definition 2.4.1. A *property* is a set of traces. The set of all properties is $\mathsf{Prop} = 2^{A^{\infty}}$.

Thus we say that a system satisfies some security policy P, which is a property, if all traces of the system are in P. This is important in practice as each security property P has a respective characteristic predicate on single execution traces [79]. The predicate can be used to determine whether any particular execution satisfies the respective property, i.e. whether the execution is in the set of traces P. A sample property is presented next.

Example 2.4.2. *The policy GServ is a property which guarantees a user that if she pays for some digital content, she can always download it later:*

$$GServ = \{t \in A^{\omega} \mid \forall i, j \in \mathbb{N} \,.\, pays_{u,cont}(t(i)) \wedge j > i \to canDownload_{u,cont}(t(j))\},$$

where pays and canDownload are predicates on actions (parameterized by user and content) and $t(i)$ specifies the element at the i-th position of trace t.

Unfortunately, properties are not expressive enough for a number of security policies, such as (seemingly) all notions of noninterference and secure information flow specifications [55, 18]. To remedy this, Clarkson and Schneider introduced the notion of hyperproperties [18].

Definition 2.4.3. A *hyperproperty* is a set of sets of traces or equivalently a set of properties. The set of all hyperproperties is $\mathsf{HP} = 2^{2^{A^\infty}} = 2^{\mathsf{Prop}} = 2^{\mathsf{Sys}}$.

Our definition, unlike the original one, does not require all traces to be infinite; as a result termination-sensitive definitions can be expressed in a more natural fashion. Hyperproperties generalize properties and are expressive enough to capture not only all notions of noninterference and secure information flow, but also many other interesting policies on systems. Intuitively, a hyperproperty is the set of systems permitted by some policy. Although arising in the context of security, hyperproperties are not necessarily limited to security policies; they can be seen as very general and expressive system specifications. Quality of Service (QoS) and Service Level Agreement (SLA) properties can be expressed as hyperproperties.

Definition 2.4.4. The *satisfaction relation for hyperproperties* $\models \subseteq \mathsf{Sys} \times 2^{\mathsf{Prop}}$ is defined as follows:

$$C \models H \mathrel{\widehat{=}} C \in H$$

Although $\mathsf{Sys} = \mathsf{Prop}$, in general both names are used for emphasis. Throughout this dissertation, we will follow this convention.

We now present an example hyperproperty, a variant of *noninterference*. Let $\tau \notin A$ represent unobservable elements of a trace. Let $A = L \cup H$, $L \cap H = \emptyset$, $A_\tau = A \cup \{\tau\}$ and assume predicates *low* and *high* on elements of A such that $low(a) \mathrel{\widehat{=}} a \in L$, $high(a) \mathrel{\widehat{=}} a \in H$ and *low* is equivalent to $\neg high$. These predicates indicate whether an event is non-confidential (low) or confidential (high). Coinductively define function $ev_Z : A^\infty \to A_\tau^\infty$ to filter out events from a set $Z \subseteq A$:

$$\frac{}{ev_Z(\varepsilon) = \varepsilon}\ coind \qquad \frac{ev_Z(x) = y \qquad a \notin Z}{ev_Z(a \cdot x) = \tau \cdot y}\ coind \qquad \frac{ev_Z(x) = y \qquad a \in Z}{ev_Z(a \cdot x) = a \cdot y}\ coind$$

This can be used to filter out events that are not in set L, i.e. to filter out confidential events from set H. Next, coinductively define *weak trace equivalence* $\approx \subseteq A_\tau^\infty \times A_\tau^\infty$ as:

$$\frac{}{\varepsilon \approx \varepsilon}\ coind \qquad \frac{x \approx y}{\tau \cdot x \approx y}\ coind \qquad \frac{x \approx y}{a \cdot x \approx a \cdot y}\ coind$$

This definition can be described as equivalence up to prefix and a form of stuttering. To illustrate it, consider the traces $ab\tau\tau\tau$ and ab: they are related, as are the traces τ^ω and ab. Actually, τ^ω is related to any trace. Using such an equivalence relation makes definitions of security *termination-insensitive* [1]. This definition can be thus seen as potentially problematic. However, such definitions are often used in the literature with the argument that the problems are negligible for some concrete applications (for instance [87]).

Coinductively define predicate $no_Z : A_\tau^\infty \to 2$, stating that there are no events from some set Z in a trace:

$$\frac{}{no_Z(\varepsilon)}\ coind \qquad \frac{a \notin Z \qquad no_Z(x)}{no_Z(a \cdot x)}\ coind$$

In the next definition, we instantiate Z in no_Z with H to get a definition of no_H. Define *noninterference* as follows:

$$NI \quad = \quad \{T \subseteq \mathsf{Sys} \mid \forall t_0 \in T \exists t_1 \in T \,.\, no_H(t_1) \wedge ev_L(t_0) \approx t_1\}.$$

For every trace t_0 in a candidate set T the definition of *NI* requires a low-equivalent modulo weak trace equivalence trace t_1 in T such that t_1 has no high events. This definition of noninterference is similar in spirit to *strong non-deterministic noninterference* (*NNI*), originally proposed by Focardi and Gorrieri [26]. The major difference is that *NI* does not distinguish between inputs and outputs; thus it is in a sense stronger than similar definitions that guard the confidentiality of high inputs only (*NI* ensures the confidentiality of high events, which implies confidentiality of high inputs). Additionally, *NNI* is defined over elements of 2^{A^*}, whereas *NI* over elements of Sys; finally, *NNI* uses string equality whereas *NI* uses weak trace equivalence (a form of weak-bisimulation).

Example 2.4.5 (Noninterference)**.** *Given $A = \{a,b,c\}$ such that $high(a)$, $high(c)$, $low(b)$ hold. Consider system $C = \{\sigma,\gamma\}$, where $\sigma = (abc)^\omega$, $\gamma = b^\omega$. Note that $no_H(\gamma) = true$, $ev_L(\sigma) \approx b^\omega$ and $ev_L(\gamma) = b^\omega$ hold. From these we deduce:*

1. *for σ there exists $t \in C$ such that $no_H(t) \wedge ev_L(\sigma) \approx t$, namely $t = \gamma$.*
2. *for γ there exists $t \in C$ such that $no_H(t) \wedge ev_L(\gamma) \approx t$, namely $t = \gamma$.*

Hence $C \models NI$: system C satisfies NI as well as variants of NNI.

[1] A termination-insensitive definition of noninterference allows leaks only via the system's termination behavior, namely whether the system terminates or not.

The former definition of noninterference is relatively abstract. In order to additionally illustrate the practical significance of the proposed approach, we also work with a variant of *reactive noninterference* [11], which is a variant of Zdancewic and Myers's definition of *observational determinism* [87] for reactive systems. Without loss of generality, assume that A may be partitioned into A_i and A_o, corresponding to input and output events. Following the original work [11], assume that systems are input-total, i.e. in every state all inputs are accepted. However, if the system is in a producer state then the input is buffered and may never be processed. Moreover, assume that every input event produces some finite or infinite output trace. The model assumes that a system waits for input in some consumer state; whenever an input event is received, the system produces a finite or infinite output trace; if the output trace was finite, the system returns to a consumer state, waiting for further events; otherwise it diverges.

For the sake of illustration, we consider deterministic reactive systems. Formally, a *deterministic reactive system RS* can be modeled as the set of traces produced by a function $f_{RS} : A_i^\infty \to A^\infty$. Let $f_i : A_i \to A_o^\infty$ be a function, taking one input event and producing some output trace. Function f_{RS} can be defined coinductively as:

$$\frac{}{f_{RS}(\varepsilon) = \varepsilon}\ {\scriptstyle coind} \qquad \frac{f_i(a) = \sigma \qquad \sigma \in A^* \qquad f_{RS}(r) = \sigma_m}{f_{RS}(a \cdot r) = a \cdot \sigma \cdot \sigma_m}\ {\scriptstyle coind}$$

$$\frac{f_i(a) = \sigma \qquad \sigma \in A^\omega}{f_{RS}(a \cdot r) = a \cdot \sigma}\ {\scriptstyle coind}$$

Note that states in our definition are implicit. In practice, each input event is processed by function f_i in a consumer state (unless the system is diverging and never returns to a consumer state, in which case the rest of the input trace is ignored), after every finite output trace the system returns to a consumer state. Let *FRS* be the set of deterministic reactive systems that can be characterized by a functional input-output relation. Let $x \approx_{\mathrm{L}} y$ denote $ev_L(x) \approx ev_L(y)$, and $\approx_{\mathrm{L_i}}$ and $\approx_{\mathrm{L_o}}$ be analogous definitions for input and output events, respectively.

Reactive noninterference [11] can be defined as a hyperproperty as follows:

$$RN = \{T_f \in FRS \mid \forall t_0, t_1 \in T_f (t_0 \approx_{\mathrm{L_i}} t_1 \to t_0 \approx_{\mathrm{L}} t_1)\},$$

where $T_f = \{t \in A^\infty \mid \exists \sigma \in A_i^\infty (f(\sigma) = t)\}$. Note that the relation $t_0 \approx_{\mathrm{L}} t_1$ is on whole traces, not on output traces as it is typically defined. This is because $t_0 \approx_{\mathrm{L}} t_1$ implies $t_0 \approx_{\mathrm{L_o}} t_1$, as L_o is a subset of L and the way traces are generated.

Unlike batch-job program models, where all program inputs are available at the start of execution and all program outputs are available at program termination, reactive programs receive inputs and send outputs to their environment during execution.

RIMP [11] is a language geared towards writing reactive systems, allowing agents to interact with the system by sending and receiving messages. Messages are typically considered secret to certain agents and public to others. Inputs in RIMP are natural numbers sent over channels and outputs are natural numbers over channels or a tick (τ), signifying an internal action. For instance, $ch^i_H(0)$ and $ch^i_L(0)$ inputs 0 on the high and low channels respectively, whereas $ch^o_H(0)$ and $ch^o_L(0)$ outputs 0 on the high and low channels. The channels model users or security levels in some security lattice; typically, L and H model the low and high channel respectively. The detailed syntax and semantics of RIMP are available in the original paper [11].

Example 2.4.6. *The following RIMP program illustrates reactive noninterference:*

```
input chH(x) {i := x;}
input chL(x) {if i <= x then  output chL(0);
              else output chL(1);}
```

Program 2.1: Simple program in RIMP

Let $\sigma_{in} = [ch^i_H(0), ch^i_L(0)]$ *and* $\gamma_{in} = [ch^i_H(1), ch^i_L(0)]$ *be input strings. Clearly* $\sigma_{in} \approx_{L_i} \gamma_{in}$. *The traces generated by the input traces* σ_{in} *and* σ_{out} *are* $\sigma = [ch^i_H(0), \tau, \tau, ch^i_L(0), \tau, ch^o_L(0), \tau]$ *and* $\gamma = [ch^i_H(1), \tau, \tau,\ ch^i_L(0), \tau, ch^o_L(1), \tau]$ *respectively; because they are not weak trace equivalent at L, it follows that P is not secure, i.e.* $P \not\models RN$.

2.5 MAKS Framework Overview

The original notion of unwinding dates back to the work of Goguen and Meseguer [32]. As they describe it, unwinding is the process of translating a security policy, first, into local constraints on the transition system, inductively guaranteeing that the policy is satisfied, and second, into a finite set of lemmas such that any system that satisfies the lemmas is guaranteed to satisfy the policy.

There is a substantial amount of work on unwinding of information flow policies [32, 37, 72, 61, 74]. Such contributions typically result in unwinding theorems that give more practical means of proving the respective high-level security policy. Unwinding theorems are developed for specific definition(s) and hence their proofs lack modularity. This is unfortunate, as it results in the need to reprove many similar results.

In an attempt to remedy this, Mantel [52, 55] introduced the Modular Assembly Kit for Security Properties (MAKS) framework — a modular framework in which most well-known information flow policies can be composed from a set of *basic security predicates* (BSPs). A major advantage of Mantel's framework is precisely its

modularity: BSPs common to different definitions need to be verified only once per system; the same holds for unwinding relations on systems. Prominent examples of policies specifiable in Mantel's framework include *generalized noninterference* [58], *noninference* [86], *generalized noninference* [86], *separability* [59], *perfect security property* [86] etc. One may also use the framework to create new policies. Interestingly, some BSPs are equivalent to, and can be constructed as, conjunctions of unwinding relations, whereas other BSPs are over-approximated by conjunctions of unwinding relations. Mantel's unwinding relations are noteworthy for at least two major reasons. First, because they can be *arbitrary* relations rather than equivalence relations, as are typically found in the literature. And second, because they can be specified *locally*, on states of the system (inspired by Rushby's technical report [72]), as opposed to the more traditional, global, trace-based unwinding relations. In addition to the local unwinding relations for his BSPs, Mantel presented *unwinding theorems* for most known possibilistic security policies.

As mentioned above, security policies may be decomposed into their building blocks, the so-called *BSPs*, which in turn can be constructed from *unwinding relations*. A

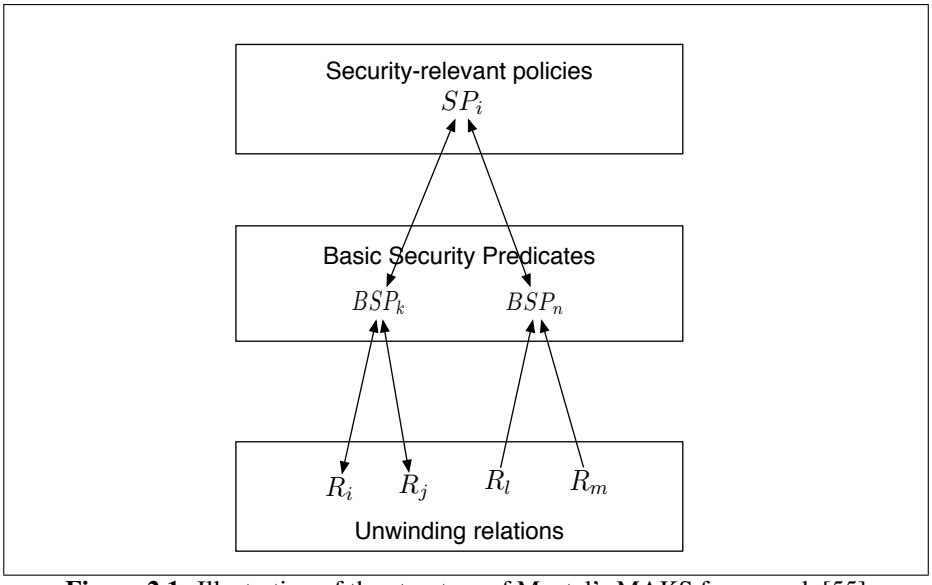


Figure 2.1: Illustration of the structure of Mantel's MAKS framework [55]

high level view of this hierarchy can be seen in Figure 2.1. Note that conjunctions are hidden in the figure. Consider a security policy called SP_i. It is a (hidden) conjunction of building block BSPs, here BSP_k and BSP_n. A BSP is a conjunction of unwinding relations, here BSP_k is the conjunction of R_i and R_j, similarly BSP_n is the conjunction of R_l and R_m. Note however, that $R_i \wedge R_j$ is equivalent to BSP_k, whereas R_l and R_m

implies BSP_n. This framework is both conceptually appealing and well-established, hence we would like to have similar results for security-relevant hyperproperties.

2.5.1 Definitions and Notation

Next, we present definitions adopted from Mantel's MAKS framework [55]. The first two definitions are needed to understand some of Mantel's definitions presented in Chapter 4.

Definition 2.5.1. A *state-event system* (*SES*) can be given as the following sextuple (S, s_0, A, A_i, A_o, T), where S is the state-space, A is the alphabet, A_i and A_o are inputs and outputs respectively and $T \subseteq S \times A \times S$ is a transition relation.

The transition relation is denoted $s_1 \xrightarrow{a} s_2$. It is extended to finite traces in the obvious way and denoted $s_1 \xrightarrow{t} s_2$, where t is a finite trace.

Definition 2.5.2. Consider some state event system *SES*. A state $s \in S$ is *reachable*, denoted $reachable(SES, s)$, if there is a finite trace t such that $s_0 \xrightarrow{t} s$.

Definition 2.5.3. For alphabet A define a *view* to be a tuple $V = (A_v, A_n, A_c)$, where $A = A_v \cup A_n \cup A_c$ is a partition of A, corresponding to *visible*, *neutral* (neither visible nor confidential) and *confidential* events.

Let $\mathcal{H}$ denote the view $(L, \emptyset, H)$ where H and L are the sets of *high* and *low* confidentiality events, and the set of neutral events is empty. Let sets I and O represent inputs and outputs such that $I \subseteq A$, $O \subseteq A$ and $I \cap O = \emptyset$. Let $\mathcal{H}I$ denote the view $(L, H \setminus HI, HI)$, where HI is the set of high inputs, i.e. $H \cap I$. Let the set of all views (partitions of A) be $\mathcal{V}$ and ρ be a function from *views* to subsets of A, i.e. $\rho : \mathcal{V} \to 2^A$. In the following definition $\approx_Z$ is a binary relation on finite traces, defined as equality of the projections of the traces to a set $Z \subseteq A$.

Definition 2.5.4. An event is defined to be ρ-admissible in a tree T after a possible finite trace β for some view $V = (A_v, A_n, A_c)$ if $Adm^{\rho}_V(T, \beta, e)$ holds, where $Adm^{\rho}_V(T, \beta, e)$ is defined as follows:

$$Adm^{\rho}_V(T, \beta, e) \mathrel{\widehat{=}} \exists \gamma \in A^*.(\gamma \cdot e \in T \wedge \gamma \approx_{\rho(V)} \beta).$$

The idea of policies based on ρ (a function from a view to a subset of A) is to create uncertainty about the nonoccurrence of events. Typical instantiations of ρ are $\rho_{A_c}(A_v, A_n, A_c) = A_c$ and $\rho_A(A_v, A_n, A_c) = A$. For instance, one policy on some system T with respect to ρ_{A_c} might be defined as follows: given some view (A_v, A_n, A_c), observing events in A_v one should not be able to deduce the nonoccurrence of ρ-admissible events in system T from the set A_c. This is the intuition behind policy

$IA_V^{\rho o}$ from Section 4.3.2. Depending on the particular policy one tries to enforce, other instantiations of ρ may be meaningful.

2.6 Partial Automata, Coalgebras and Languages à la Rutten

We next present some theory about partial automata, coalgebra and languages, based on work by Rutten [73].

Let F be an arbitrary endofunctor $F : Set \to Set$, where Set is the category whose objects are sets.

Definition 2.6.1. An *F-coalgebra* is a pair $\langle S, f\rangle$, where S is the system state space and $f : S \to F(S)$ is a function giving the transition structure of the system.

Definition 2.6.2. An F-coalgebra is *final* if there is a unique homomorphism from any other F-coalgebra to it.

Definition 2.6.3. A *partial automaton* with input alphabet A is defined *coalgebraically* as a 3-tuple $\langle S, o, t\rangle$, where set S is the possibly infinite state space of the automaton, the observation function $o : S \to 2$ says whether a state is accepting or not, and the partial function $t : S \to (1+S)^A$ gives the transition structure.

Notation $1+S$ is used for the set $\{\perp\} \cup S$: if the function $t(s)$ is defined for some $a \in A$, then $t(s)(a) = s'$ gives the next state; otherwise, if $t(s)$ is undefined for some $a \in A$, then it is mapped to $\perp$. The symbol $\delta \notin A$ is used to represent *deadlock*. Let $A^* \cdot \delta = \{w \cdot \delta \mid w \in A^*\}$ be the set of finitely deadlocked words.

Definition 2.6.4. An automaton is in a *deadlock* state s_δ if for all $a \in A$, $t(s_\delta)(a) = \perp$ (the transition function is undefined for all $a \in A$).

Definition 2.6.5. Any language acceptable by partial automata is a subset of $A_\delta^\infty = A^* \cup (A^* \cdot \delta) \cup A^\omega$.

Example 2.6.6. *Consider Figure 2.2, presenting a partial automaton, having a corresponding language $L = \{ab\delta, (ac)^\omega, bc\}$, graphically. There is a deadlock state, namely s_2, and thus $ab\delta$ is in the language L. State s_5 is an accepting state, hence $bc \in L$. The infinite word $(ac)^\omega$ is also in L. No other words are in L.*

Let the truncation of an infinite word $w = a_1 a_2 a_3 \ldots$ to the first n ($n \in \mathbb{N}$) letters be denoted $w[n] = a_1 \ldots a_n$.

Definition 2.6.7. For words $w \in A^*$ and sets $L \subseteq A_\delta^\infty$, define the *$w$-derivative* of L to be $L_w = \{v \in A_\delta^\infty \mid w \cdot v \in L\}$.

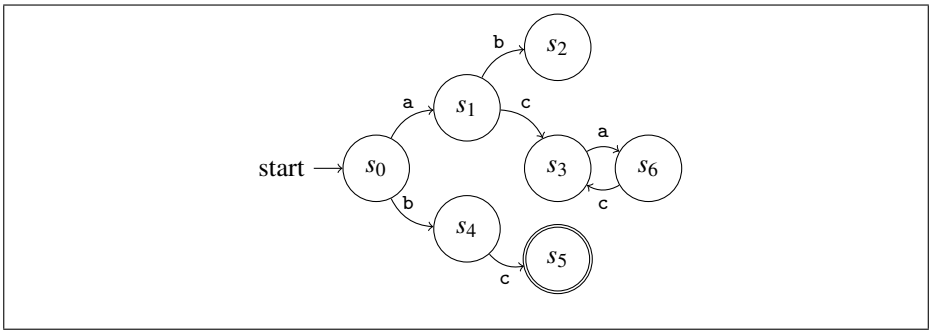


Figure 2.2: A partial automaton accepting the language $L = \{ab\delta, (ac)^{\omega}, bc\}$

Definition 2.6.8. Define a set $L \subseteq A_{\delta}^{\infty}$ to be *closed* if for any infinite word w, if $\forall n \geq 1$ we have $L_{w[n]} \neq \emptyset$, then it has to be that $w \in L$.

For instance, the set a^{∞} is closed. In contrast, a^* is not closed.

Definition 2.6.9. Define a set $L \subseteq A_{\delta}^{\infty}$ to be *consistent* if for all words w in A^*, $\delta \in L_w$ iff $L_w = \{\delta\}$.

For instance, the set $\{a\delta, bc, ba\}$ is consistent. In contrast, set $\{b\delta, bc, ba\}$ is not.

Definition 2.6.10. The *language of a partial automaton* is a non-empty, closed and consistent subset of A_{δ}^{∞}. The set of all such languages is

$$\mathcal{L} = \{L \mid L \subseteq A_{\delta}^{\infty}, L \text{ is non-empty, closed and consistent}\}.$$

Any state of a partial automaton accepts some language having three kinds of words: firstly, all finite words that leave the automaton in an accepting state, secondly, all infinite words that cause the automaton to run indefinitely and thirdly, words that lead to a deadlock state. Figure 2.2 illustrates the three kinds of words. Intuitively, the language of a partial automaton is the language accepted by the start state.

Definition 2.6.11. The set $\mathcal{L}$ can be thought of as an automaton $\mathcal{L} = \langle \mathcal{L}, o_{\mathcal{L}}, t_{\mathcal{L}} \rangle$ [73]:

$$o_{\mathcal{L}}(L) = \begin{cases} \mathit{true} & \text{if } \varepsilon \in L \\ \mathit{false} & \text{if } \varepsilon \notin L \end{cases} \qquad t_{\mathcal{L}}(L)(a) = \begin{cases} L_a & \text{if } L_a \neq \emptyset \\ \perp & \text{if } L_a = \emptyset. \end{cases}$$

Definition 2.6.12. A bisimulation between two automata $S_1 = \langle S, o, t \rangle$ and $S_2 = \langle S', o', t' \rangle$ is a relation $R \subseteq S \times S'$ such that for all s in S, s' in S' and a in A

$$s \; R \; s' \implies o(s) = o'(s') \bigwedge t(s)(a) \; (1+R) \; t'(s')(a).$$

Condition $t(s)(a)\ (1+R)\ t'(s')(a)$ holds iff either $t(s)(a) = \bot$ and $t'(s')(a) = \bot$ or $t(s)(a)\ R\ t'(s')(a)$. The maximal bisimulation $\sim$, also called *bisimilarity*, is the union of all bisimulation relations.

Theorem 2.6.13 ([73])**.** (1) *The automaton* $\mathcal{L} = \langle \mathcal{L}, o_{\mathcal{L}}, t_{\mathcal{L}} \rangle$ *satisfies the* coinduction proof principle*. In other words, for all languages* L *and* K *in* $\mathcal{L}$ *we have:* $L \sim K \iff L = K$*.* (2) *The automaton* $\mathcal{L} = \langle \mathcal{L}, o_{\mathcal{L}}, t_{\mathcal{L}} \rangle$ *is* final*: for any automaton* (S, o, t)*, there is a unique homomorphism* $h : S \to \mathcal{L}$ *such that for all* $s_1, s_2 \in S$*,* $s_1 \sim s_2$ *iff* $h(s_1) = h(s_2)$*.*

The first part of Theorem 2.6.13 guarantees that bisimilarity implies equality. This is the coinduction proof principle: if we can find a bisimulation between two languages, we have shown that they are equal. The second part presents the *coinductive definition principle* [73]: we can define a function from a set S into $\mathcal{L}$, by defining an output function o and transition function t on S. Because $\mathcal{L}$ is final, this results in a unique homomorphism $h_l : S \to \mathcal{L}$, which assigns to each state s the language that s accepts.

Coalgebras of the polynomial functor $G : \mathit{Set} \to \mathit{Set}$, given by $GX = 2 \times (1+X)^A$, will be called *G-coalgebras* or *G-systems*. As partial automata are in a one-to-one correspondence with G-coalgebras [73], all the theory presented in this section is applicable to G-coalgebras. In this work the functor G is fixed; thus, whenever we talk about G-coalgebras, they are of this particular functor.

Definition 2.6.14. A coalgebra together with a start state is called a *pointed coalgebra*. A pointed G-coalgebra will be denoted $\langle S, o, t, s_0 \rangle$, where s_0 is the start state.

2.7 First-Order Logic

This is a standard exposition of First-Order Logic (FO) as found in the literature [49].

Definition 2.7.1. A *vocabulary* σ is a collection of constant symbols $(c_1, \ldots, c_n, \ldots)$, predicate, or relation symbols $(P_1, \ldots, P_n, \ldots)$, and function symbols $(f_1, \ldots, f_n, \ldots)$. Each predicate and function symbol has a respective arity.

A vocabulary is called *relational*, if it contains only relations and constants. This restriction is not severe, as each k-ary function corresponds to a $(k+1)$-ary relation [49].

Definition 2.7.2. A σ-*structure* (also called a model) is $\mathcal{M} = (S, \{c_i^{\mathcal{M}}\}, \{P_i^{\mathcal{M}}\}, \{f_i^{\mathcal{M}}\})$, where S is the universe and there is an interpretation mapping

- each constant symbol c_i from vocabulary σ to $c_i^{\mathcal{M}} \in S$.

- each predicate symbol P_i from vocabulary σ to a k-ary relation on S, i.e. to $P_i^{\mathcal{M}} \subseteq S^k$.
- each k-ary function symbol f_i from vocabulary σ to a function $f_i^{\mathcal{M}} : S^k \to S$.

A structure is called *finite* if its universe S is a finite set. Let $STRUCT[\sigma]$ denote the class of all finite, σ-structures.

2.7.1 Syntax

Assume a countably infinite set of variables $Var = \{x, y, z, \ldots\}$.

Definition 2.7.3. Inductively define the terms and formulae of first-order logic as follows:

- Each variable x is a term.
- Each constant symbol c is a term.
- If $t_1, \ldots, t_k$ are terms and f is a function symbol with arity k, then $f(t_1, \ldots, t_k)$ is a term.
- If t_1 and t_k are terms, then it follows that $t_1 = t_k$ is an atomic formula.
- If P is a predicate symbol of arity k and $t_1, \ldots, t_k$ are terms, then it follows that $P(t_1, \ldots, t_k)$ is an atomic formula.
- If ψ_1 and ψ_2 are formulae, then it follows that $\psi_1 \wedge \psi_2$, $\psi_1 \vee \psi_2$ and $\neg\psi_1$ are formulae.
- If ψ is a formula, then it follows that $\exists x\psi$ and $\forall x\psi$ are formulae.

The typical assumption that $\psi \to \phi$ stands for $\neg\psi \vee \phi$ and $\psi \leftrightarrow \phi$ stands for $(\psi \to \phi) \wedge (\phi \to \psi)$ are in order.

Definition 2.7.4. A *free variable* (of a term or formula) is defined as follows:

- A term x has only one free variable and that is x. A constant term has no free variables.
- The free variables of $t_1 = t_k$ are the free variables of t_1 and t_k. The free variables of $P(t_1, \ldots, t_k)$ and of $f(t_1, \ldots, t_k)$ are the free variables of $t_1, \ldots, t_k$.
- The free variables of $\neg\psi$ are the same as the ones of ψ. The free variables of $\psi_1 \wedge \psi_2$ and $\psi_1 \vee \psi_2$ are the free variables of ψ_1 and ψ_2.

- The free variables of $\exists x\psi$ and $\forall x\psi$ are the free variables of ψ without x.

Variables that are not free are called *bound*. A *sentence* is a formula which has no free variables.

2.7.2 Semantics

For some structure/model $\mathcal{M}$, define inductively for each term t with free variables $(x_1,\ldots,x_k)$ the value $t^{\mathcal{M}}(\overline{a})$, where $\overline{a} \in S^k$.

- If t is a constant symbol c, then the value of t in $\mathcal{M}$ is $c^{\mathcal{M}}$.
- If t is a variable x_i, then the value of $t^{\mathcal{M}}(\overline{a})$ is a_i.
- If $t = f(t_1,\ldots,t_k)$, then the value of $t^{\mathcal{M}}(\overline{a})$ is $f^{\mathcal{M}}(t_1^{\mathcal{M}}(\overline{a}),\ldots,t_1^{\mathcal{M}}(\overline{a}))$.

In addition, for each formula define the satisfaction relation $\mathcal{M} \models \psi(\overline{a})$ (where $\overline{a} \in S^k$), by case analysis of ψ:

- If $\psi \equiv (t_1 = t_k)$, then $\mathcal{M} \models \psi(\overline{a})$ iff $t_1^{\mathcal{M}}(\overline{a}) = t_k^{\mathcal{M}}(\overline{a})$.
- If $\psi \equiv P(t_1,\ldots,t_k)$, then $\mathcal{M} \models \psi(\overline{a})$ iff $(t_1^{\mathcal{M}}(\overline{a}),\ldots,t_1^{\mathcal{M}}(\overline{a})) \in P^{\mathcal{M}}$.
- If $\psi \equiv \neg\psi(\overline{a})$ iff $\mathcal{M} \models \psi(\overline{a})$ does not hold.
- $\mathcal{M} \models \psi_1(\overline{a}) \wedge \psi_2(\overline{a})$ iff $\mathcal{M} \models \psi_1(\overline{a})$ and $\mathcal{M} \models \psi_2(\overline{a})$.
- $\mathcal{M} \models \psi_1(\overline{a}) \vee \psi_2(\overline{a})$ iff $\mathcal{M} \models \psi_1(\overline{a})$ or $\mathcal{M} \models \psi_2(\overline{a})$.
- If $\psi \equiv \exists y\psi(y,\overline{x})$, then $\mathcal{M} \models \psi(\overline{a})$ iff $\mathcal{M} \models \psi(a',\overline{a})$ for some $a' \in A$.
- If $\psi \equiv \forall y\psi(y,\overline{x})$, then $\mathcal{M} \models \psi(\overline{a})$ iff $\mathcal{M} \models \psi(a',\overline{a})$ for all $a' \in A$.

2.8 Least Fixed Point logic (LFP)

We next present Least Fixed Point logic (LFP), the extension of first-order logic with fixed point operators. Let σ be a relational vocabulary and R a relation symbol $R \notin \sigma$. Suppose $\psi(R,x_1,\ldots,x_k)$ is a formula of vocabulary $\sigma \cup \{R\}$. Note that R is an explicit parameter, as formula ψ gives rise to an operator on σ-structures.

For each σ-structure $\mathcal{M}$ in $STRUCT[\sigma]$, formula $\psi(R,\bar{x})$ gives rise to the operator $F_\psi : 2^{S^k} \to 2^{S^k}$ given as:

$$F_\psi(X) = \{\bar{a} \mid \mathcal{M} \models \psi(X/R,\bar{a})\}.$$

Note that $\psi(X/R,\bar{a})$ means that R has to be interpreted as X in formula ψ. More formally, if $\mathcal{M}'$ is a $(\sigma \cup \{R\})$-structure expanding $\mathcal{M}$ (with R) and where R can be interpreted as X, then it has to be that $\mathcal{M}' \models \psi(\bar{a})$. For a formula ψ and relation symbol R, an occurrence of R in ψ is *positive* if it is under the scope of an even number of negations. Dually, an occurrence of R in ψ is *negative* if it is under the scope of an odd number of negations. A formula ψ is positive in R, if there are no negative occurrences of R in ψ.

Lemma 2.8.1. *If $\psi(R,\bar{x})$ is positive in R, then the operator F_ψ is monotone.*

Now, we are ready to define the logic LFP.

Definition 2.8.2. LFP is an extension of FO with the following rule:

- Let $\psi(X/R,\bar{x})$ be a formula positive in R, with free second-order variable R and free first order variables $x_1,\ldots,x_k$. Let $\bar{t}$ be a k-tuple of terms. Then

  $$[\underset{R,\bar{x}}{\mathbf{lfp}}\, \psi(R,\bar{x})](\bar{t})$$

  is an LFP formula. The free variables (of the formula) are the ones of $\bar{t}$.

The semantics is given as follows:

$$\mathcal{M} \models [\underset{R,\bar{x}}{\mathbf{lfp}}\, \psi(R,\bar{x})](\bar{a}) \text{ iff } \bar{a} \in \mathbf{lfp}(F_\psi).$$

Note that sometimes we shall write $[\mathbf{lfp}\, R\, \bar{x}\, .\, \psi(R,\bar{x})](\bar{a})$ instead of $[\mathbf{lfp}_{R,\bar{x}}\, \psi(R,\bar{x})](\bar{a})$.

2.9 The Polyadic Modal Mu-calculus over Trees

The polyadic modal mu-calculus [8] is a logic whose formulae are interpreted over k-tuples of transition systems. It can be seen as an extension of the modal mu-calculus [43] with different diamond and box modalities associated with each system (from the k-tuple). In this work, formulae will be interpreted over k-tuples of trees, denoted $\overline{\mathcal{T}}$. The elements of these tuples will be referred to as $\overline{\mathcal{T}}_i$, where $1 \leq i \leq k$. Note that each $\overline{\mathcal{T}}_i$ implicitly defines a state space, namely the set of subtrees of $\overline{\mathcal{T}}_i$.

Finally, for a k-tuple of trees $\overline{\mathcal{T}}$ we use notation $\overline{\mathcal{T}} \xrightarrow{a}_i \overline{\mathcal{T}'}$ to mean that $\overline{\mathcal{T}'}_i = t(\overline{\mathcal{T}}_i)(a)$, where for all j s.t. $1 \leq j \leq k$ and $j \neq i$, $\overline{\mathcal{T}'}_j = \overline{\mathcal{T}}_j$.

Assume a set $Var_2 = \{X, Y, Z, \ldots\}$ of second-order variables and a set $P = \{Q_i, O_i, \ldots \mid 1 \leq i \leq k\}$ of propositional constants. Formulae in the polyadic modal mu-calculus $\mathcal{L}_\mu^k$ have the following syntax:

$$\Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i\Phi \mid \langle a\rangle_i\Phi \mid \nu Z.\Phi \mid \mu Z.\Phi,$$

where tt and ff are the constant true and false formulae, for $a \in A$, $[a]_i$ ("box-i a") and $\langle a\rangle_i$ ("diamond-i a") are the typical modal operators relativized to the i-th tree, where $1 \leq i \leq k$ (see Figure 2.3). As usual, μZ and νZ are the least and greatest fixed point operators respectively. Every second-order variable gets bound by a fixed point quantifier at most once in a formula. Assume that operators ν and μ have wider scope than the other operators. Sometimes, for $K \subseteq A$ it will be more convenient to abbreviate $\bigwedge_{a\in K}[a]_i\Phi$ as $[K]_i\Phi$ and $\bigvee_{a\in K}\langle a\rangle_i\Phi$ as $\langle K\rangle_i\Phi$. Finally, propositional variables are ranged over by second-order variables from Var_2.

Formally, a valuation V is a function from Var_2 to k-tuples of trees. V is also overloaded to assign to each propositional constant Q_i a subset $\mathsf{V}(Q_i) \subseteq SubT(\mathcal{T}_0^i)$, where $SubT(\mathcal{T}_0^i)$ is the set of subtrees of $\mathcal{T}_0^i$, including $\mathcal{T}_0^i$ itself. The propositional constant O will have a special meaning in the logic — intuitively, O_i indicates whether (the root of) tree $\mathcal{T}_i$ is accepting or not. Formally, define the valuation of the propositional constant O in tree i as follows: $\mathsf{V}(O_i) \mathrel{\widehat{=}} \{T \in SubT(\mathcal{T}_0^i) \mid o(T)\}$. Propositions O_i, where $1 \leq i \leq k$, correspond to the observation function o of t_i. Let $\mathsf{V}[\mathtt{T}/Z]$ be the valuation V' that is the same as V everywhere except at Z where $\mathsf{V}'(Z) = \mathtt{T}$.

The satisfaction relation, relative to some valuation V, is denoted $\models_\mathsf{V}$ and defined inductively in Figure 2.3. Let $||\Phi||_\mathsf{V}^{\overline{\mathcal{T}}}$ denote the set of k-tuples of trees in $SubT(\overline{\mathcal{T}})$ satisfying a formula Φ under valuation V, defined as $||\Phi||_\mathsf{V}^{\overline{\mathcal{T}}} \mathrel{\widehat{=}} \{T \in SubT(\overline{\mathcal{T}}) \mid T \models_\mathsf{V} \Phi\}$. Note that here $SubT$ is lifted to mean the set of k-tuples of subtrees of the original k-tuple of roots. It should be noted that negation is defined implicitly (we have no $\neg$ operator), by taking the complement of each formula in the grammar.

Although the full expressive power of $\mathcal{L}_\mu^k$ is never really needed for the specifications we have encountered so far, the IHP checking games we propose in Section 5.4 for the full $\mathcal{L}_\mu^k$ turn out to be interesting from both theoretical and practical points of view. In an exploration of the practical significance of this work, in Section 5.6 we propose a fragment of $\mathcal{L}_\mu^k$, expressive enough to capture all incremental hyperproperties we are aware of.

$$
\begin{array}{lll}
T & \models_{\mathsf{V}} & tt \\
T & \not\models_{\mathsf{V}} & ff \\
T & \models_{\mathsf{V}} & Z \text{ iff } T \in \mathsf{V}(Z) \\
T & \models_{\mathsf{V}} & \Phi \wedge \Psi \text{ iff } T \models_{\mathsf{V}} \Phi \text{ and } T \models_{\mathsf{V}} \Psi \\
T & \models_{\mathsf{V}} & \Phi \vee \Psi \text{ iff } T \models_{\mathsf{V}} \Phi \text{ or } T \models_{\mathsf{V}} \Psi \\
T & \models_{\mathsf{V}} & [a]_i\Phi \text{ iff } \forall S \in \{T' \mid T \xrightarrow{a}_i T' \text{ and } a \in A\} \text{ such that } S \models_{\mathsf{V}} \Phi \\
T & \models_{\mathsf{V}} & \langle a\rangle_i\Phi \text{ iff } \exists S \in \{T' \mid T \xrightarrow{a}_i T' \text{ and } a \in A\} \text{ such that } S \models_{\mathsf{V}} \Phi \\
T & \models_{\mathsf{V}} & \nu Z.\Phi \text{ iff } T \in \bigcup\{\mathtt{T} \subseteq \overline{\mathcal{T}} \mid \mathtt{T} \subseteq ||\Phi||^{\overline{\mathcal{T}}}_{\mathsf{V}[\mathtt{T}/Z]}\} \\
T & \models_{\mathsf{V}} & \mu Z.\Phi \text{ iff } T \in \bigcap\{\mathtt{T} \subseteq \overline{\mathcal{T}} \mid ||\Phi||^{\overline{\mathcal{T}}}_{\mathsf{V}[\mathtt{T}/Z]} \subseteq \mathtt{T}\}
\end{array}
$$

Figure 2.3: Inductive definition of satisfaction relation $\models_{\mathsf{V}}$

It is noteworthy that any (hyper)property expressed in $\mathcal{L}^k_\mu$ can be checked in polynomial time [67]. To be more precise, we next recall a known result [8]: on finite state systems, the model checking problem for $\mathcal{L}^k_\mu$ is efficiently decidable.

Theorem 2.9.1 (Model-checking of $\mathcal{L}^k_\mu$ is decidable)**.** *There exists an algorithm for deciding $\overline{\mathcal{T}} \models \Phi$ in time $O(|\Phi|^m(|S_1|\ldots|S_k|)^{m-1}|\mathcal{T}_1|\ldots|\mathcal{T}_k|)$, where Φ is closed, $\overline{\mathcal{T}}$ a k-tuple of finite transition systems with state spaces $S_1,\ldots,S_k$ and m is the alternating depth of Φ.*

Here $|T_i|$ denotes the size of the underlying state space plus the size of the transition function plus 1 (the size of the initial state) and $|S_i|$ denotes the size of the state space. Note that this theorem is applicable to our setting for reasoning about potentially infinite trees, generated by finite-state partial automata, which are effectively transition systems.

The following definitions are useful for working with the logic.

Definition 2.9.2. A formula Φ is *normal* if it satisfies two conditions.

1. If σZ_1 and σZ_2 are distinct subterms in formula Φ, then $Z_1 \neq Z_2$.

2. No free variable is also used in a binder, i.e. if Z is free, then σZ must not occur in formula Φ.

Definition 2.9.3. Inductively define the set of subformulae $Sub(\Phi)$ of a certain formula Φ as:

$$\begin{aligned}
Sub(tt) &= \{tt\} \\
Sub(ff) &= \{ff\} \\
Sub(Z) &= \{Z\} \\
Sub(\Phi_1 \wedge \Phi_2) &= \{\Phi_1 \wedge \Phi_2\} \cup Sub(\Phi_1) \cup Sub(\Phi_2) \\
Sub(\Phi_1 \vee \Phi_2) &= \{\Phi_1 \vee \Phi_2\} \cup Sub(\Phi_1) \cup Sub(\Phi_2) \\
Sub([a]_i\Phi) &= \{[a]_i\Phi\} \cup Sub(\Phi) \\
Sub(\langle a\rangle_i\Phi) &= \{\langle a\rangle_i\Phi\} \cup Sub(\Phi) \\
Sub(\nu Z.\Phi) &= \{\nu Z.\Phi\} \cup Sub(\Phi) \\
Sub(\mu Z.\Phi) &= \{\mu Z.\Phi\} \cup Sub(\Phi)
\end{aligned}$$

Definition 2.9.4 (Subsumes)**.** Let Φ be normal. Let $\sigma X.\Psi$, $\sigma Z.\Psi' \in Sub(\Phi)$. We say that variable X subsumes Z if $\sigma Z.\Psi' \in Sub(\sigma X.\Psi)$.

2.10 Games

The games [57] considered in this thesis have two players — *verifier*, denoted as player V, and *refuter*, denoted as player R. A *game* consists of an arena and a winning condition, as defined next.

Definition 2.10.1. An *arena* is a triple $\mathcal{A} = (\mathbb{V}_V, \mathbb{V}_R, E)$, where $\mathbb{V}_V$ is a set of vertices, associated with player V, $\mathbb{V}_R$ is a set of vertices, disjoint from $\mathbb{V}_V$, associated with player R, $\mathbb{V}_V \cup \mathbb{V}_R = V$ and $E \subseteq \mathbb{V} \cup \mathbb{V}$ is a set of edges representing possible moves.

A play can be imagined as follows: initially, there is a token on some vertex $v \in V$. If $v \in \mathbb{V}_V$ then player V makes a move, otherwise player R makes a move into one of the possible next positions.

Definition 2.10.2. Define a *play* in the arena $\mathcal{A}$ to be a finite or infinite path over $\mathbb{V}$, i.e. a play is an element of $\mathbb{V}^* \cup \mathbb{V}^\omega$.

There are different winning conditions used to determine the winner of a play.

Definition 2.10.3. A *winning condition* is a set $Win \subseteq \mathbb{V}^* \cup \mathbb{V}^\omega$.

Formally, a *game* is a tuple $(\mathcal{A}, Win)$, where $\mathcal{A}$ is the arena and *Win* is the winning condition. Some well-known examples of winning conditions from the literature [57] are Muller, Rabin, Streett and parity winning conditions. Let a strategy be a family of rules prescribing how the players in a game move.

Definition 2.10.4. A *strategy* is a partial function $f_\sigma : \mathbb{V}^*\mathbb{V}_\sigma \to \mathbb{V}$, where $\sigma = \{V, R\}$.

A *history-free strategy* is a strategy that does not depend on what happened previously in the play, but only on the current position. A game is *determined* if either V or R has a winning strategy. A game of *perfect information* is a game in which each player is aware of all previous moves in the game.

Definition 2.10.5. A *parity game* is a game of perfect information, played by two players on a directed graph $G = (N, \to, L)$. N is a finite set of vertices from the set of natural numbers $\mathbb{N}$, $\to \subseteq N \times N$ is a binary relation and $L : N \to \{V, R\}$ is a labeling function, assigning elements from the set of players $\{V, R\}$ to vertices.

All plays in a parity game have infinite length, i.e. for all $n \in N$ there is some $k \in N$ such that $n \to k$. A play always starts with a token on the least vertex. The winner of a play is determined by a parity winning condition, i.e. by the label of the least vertex i occurring infinitely often, i.e. if $L(i) = V$ then V wins the game, otherwise R wins the game. Such a game is also called *min-parity* to distinguish it from *max-parity* ones, where the winner is determined by the label of the greatest vertex occurring infinitely often. On finite graphs, the two types of games are equivalent [29].

Reasoning about parity games is considerably simplified by the fact that the games are positionally determined: in other words the history of some play is not important, only the current position matters. This is guaranteed by the following theorem:

Theorem 2.10.6 ([57])**.** *Parity games are positionally determined: from each vertex/position of the game either layer V or player R has a history-free winning strategy.*

Chapter 3

From Holistic to Incremental Hyperproperties

This chapter introduces, motivates and formalizes incremental hyperproperties, which are an essential part of our approach to the verification of hyperproperties.

In their original work Clarkson and Schneider used a relatively rich language to express security-relevant hyperproperties as first-order predicates on sets of traces [18, 19], using universal and existential quantifiers over traces in a candidate set, as well as relations (e.g. equality of projections with respect to different views, various notions of observational equivalence, weak trace equivalence) on those traces. We formalize this language ($\mathcal{HL}$) and call the hyperproperties in this syntactic class *holistic* as they talk about whole traces at once (and often about systems as a whole). Their specifications tend to be relatively straightforward to read and write, but unfortunately they are difficult to reason about. This is exemplified by the fact that no general approach to verifying such hyperproperties exists, to the best of our knowledge. To address this problem we adopt a coalgebraic perspective on systems and hyperproperty specifications. Coalgebra provides an *incremental* approach to both system specification and verification through the notion of derivative (see Section 2.6). Taking a derivative of a language/property provides a way to reason incrementally about the language/property. This is the basis of our proposed *incremental* perspective on systems and hyperproperties.

More concretely, we propose to model systems as coalgebras of the functor $GX = 2 \times (1+X)^A$ and hyperproperties as coinductively-defined relations on systems' state-spaces. This is useful because, as already mentioned, coalgebras and coinductive predicates are incremental and this makes verification more feasible. Intuitively, a

system's behavior corresponds to a unique tree, where the initial state of a system corresponds to the root of the tree and possible executions build the branches by continuously taking derivatives; incremental hyperproperties reason about such trees. The problem of converting holistic specifications into incremental ones still remains. We attempt to tackle it by introducing the process of *incrementalization*.

The main results presented in this chapter are first, formal definitions of the classes of *holistic* and *incremental* hyperproperties, second, the introduction of the notion of *incrementalization* — one way of converting holistic specifications into incremental ones — and its application to three classes of holistic hyperproperties, and finally, an illustration of a verification approach for incremental hyperproperties. The chapter is based on an extended version of a recent paper of ours [63]. The relevant proofs can be found in Appendix A.

3.1 Systems — from Sets of Traces to Trees

We start off by showing that instead of viewing a system's behavior as a set of traces, we can equivalently view it as a tree. Realizing this is an important prerequisite to incrementalization: the conversion of system specifications on sets of traces to specifications on trees. Interestingly, both these views on system behavior can be seen as generated by coalgebra/partial automata and thus we will model systems as partial automata. Since reasoning about trees and partial automata is often more convenient than reasoning about sets of traces, it will be preferred in the rest of this work.

Although we predominantly reason about trees, we would like to have results about sets of traces too. To achieve this we show that sets of traces, trees and coalgebras/partial automata can be treated as equivalent perspectives on systems. These perspectives are illustrated in Figure 3.1; note that ellipses indicate infinite repetition of the string that has occurred so far. We next explain what we mean by saying that a set of traces, a tree and a G-coalgebra/partial automaton are equivalent views on systems. To that end, we show that going between these alternative representations is relatively straightforward:

- *Language→tree*: A language can be seen as a tree by continuously taking derivatives with respect to elements of A. An a branch, where $a \in A$, is present in the tree if there are finite or infinite sequences beginning with a. The subtree under the a branch corresponds to the a-derivatives (Definition 2.6.7). A node in the tree is marked as a valid stopping state if the empty string is in the derivative (see Definition 2.6.7) of the language under consideration. For instance $L_{bc} = \{\varepsilon\}$, where ε is the empty string and hence the respective tree node is marked as a valid stopping state.

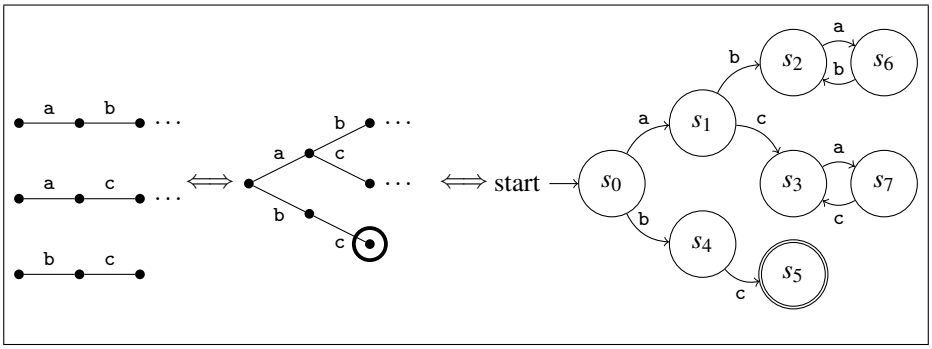


Figure 3.1: Illustration: equivalent representations of $M = \{(ab)^\omega, (ac)^\omega, bc\}$ over $A = \{a, b, c\}$: as a language/set of traces, a tree and a G-coalgebra. The branches of the tree are labelled with elements of the alphabet A, its accepting nodes are marked with a circle.

- *Tree→language*: The language of a tree is given by the paths from the root that either end at a marked node or continue forever. For instance, bc is in the language M (see Figure 3.1) as is $(ab)^\omega$.

- *Language→G-coalgebra*: Any language is an element of the final G-coalgebra $\mathcal{L}$; the function $h : \mathsf{Sys} \to \mathcal{L}$ (see Figure 3.2) uniquely determines a pointed final G-coalgebra (see Definition 2.6.14).

- *G-coalgebra→language*: The language of a G-coalgebra, seen as a partial automaton, is given by the traces accepted by the automaton, where an infinite word is accepted if it causes the automaton to run indefinitely.

- *Tree→G-coalgebra*: The root of the tree is mapped to the start state of the corresponding G-coalgebra. Each subtree of the original tree is mapped to a state of the G-coalgebra and the transitions arise from following the labeled branches of the tree. Accepting subtrees are marked as final states of the automaton.

- *G-coalgebra→tree*: A tree can be seen as the unfolding of a G-coalgebra [3]. The start state of the G-coalgebra is mapped to the root of the resulting unique tree. An a-transition, for any $a \in A$, results in an a-labelled branch in the corresponding tree; the resulting state in the G-coalgebra after an a-transition corresponds to the respective subtree, arrived at by following the a branch of the tree. Accepting states of the G-coalgebra are mapped to accepting subtrees. Intuitively, the tree is the behavior of the respective G-coalgebra.

The incremental view of systems, combined with Rutten's observation that the set of all languages is a final G-coalgebra (see Theorem 2.6.13), allows the use of coalgebra and coinduction for reasoning about hyperproperties.

We next make the needed transition from sets of traces to G-coalgebras more concrete. The functor G has the final coalgebra $\mathcal{L}$ (see Theorem 2.6.13). Recall that the coinductive definition principle, presented in Chapter 2.6, gives a way to define maps from arbitrary G-coalgebras into the final G-coalgebra. We use the principle to convert an arbitrary set of traces into a G-coalgebra. To this end, take the state space to be $\mathsf{Sys} = 2^{A^\infty}$, the set of all possible sets of traces. Any pair $\langle o,t\rangle$ with signatures $o : \mathsf{Sys} \to 2$ and $t : \mathsf{Sys} \to (1+\mathsf{Sys})^A$ induces a unique homomorphism $h : \mathsf{Sys} \to \mathcal{L}$ that makes the diagram in Figure 3.2 commute. Thus the homomorphism h would map any set of traces $s \in \mathsf{Sys}$ to a unique element in the final coalgebra $\mathcal{L}$. Since the elements of the final coalgebra can be seen as trees, we say that h maps a set of traces to the root of a unique tree in $\mathcal{L}$ corresponding precisely to that set of traces.

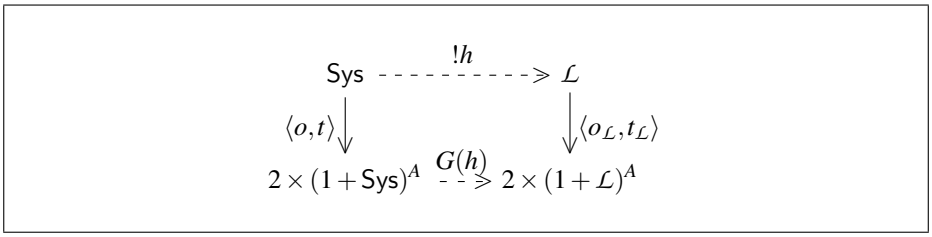


Figure 3.2: A commutative diagram to illustrate the unique homomorphism h mapping any set of traces $s \in \mathsf{Sys}$ to a unique element $\mathcal{L}$.

We now define a particular pair $\langle o,t\rangle$. Let $C \subseteq \mathsf{Sys}$, $a \in A$ and $\sigma \in A^\infty$. First define an auxiliary function $test : \mathsf{Sys} \to (A \to 2)$ as follows:

$$test_a(C) \mathrel{\widehat{=}} \exists\, \sigma.\sigma \in C \wedge \sigma(0) = a.$$

The functions o and t can be readily defined as follows:

$$o(C) \quad \widehat{=} \quad \varepsilon \in C \qquad\qquad t(C)(a) \quad \widehat{=} \quad \begin{cases} \{\sigma' \mid \sigma(0) = a\} & \text{if } test_a(C) \\ \bot & \text{if } \neg test_a(C). \end{cases}$$

The function pair $\langle o,t\rangle$ induces a unique element of the final coalgebra $\mathcal{L}$ corresponding to the (not injective in general) map of Sys into $\mathcal{L}$. Clearly Sys is closed, following directly from Definition 2.6.8. In addition, Sys is consistent in the sense of Definition 2.6.9: the reason is that there are no stuck words in Sys, i.e. there is no δ element allowed by the definition of Sys. Because Sys is nonempty, closed, consistent, and $\mathsf{Sys} \subseteq A^\infty_\delta$, it follows that $\mathsf{Sys} \subseteq \mathcal{L}$.

In summary, any system defined as a set of traces can be uniquely seen as a tree generated by a G-coalgebra. Hence we can reason about G-coalgebra and the corresponding trees to get results about the behavior of systems seen as sets of traces.

3.2 Holistic and Incremental Hyperproperties

Switching the holistic perspective with an incremental one requires us to change not only the model of systems, but also the manner in which they are specified. To achieve this we need formalisms for reasoning about holistic and incremental specifications. In this section we give such formalisms, namely, definitions of *holistic* and *incremental* hyperproperties. The logical languages used are based on *Least Fixed Point Logic* (LFP) [13] — an extension of first order logic with the addition of fixed point operators. Details on LFP are available in Section 2.8. The new logical languages are holistic hyperproperty logic $\mathcal{HL}$, in which most interesting security-relevant hyperproperties are expressible, and incremental hyperproperty logic $\mathcal{IL}$. The key difference between the languages is that the former has coinductive and/or inductive predicates over traces, whereas the latter has only coinductive predicates over trees.

3.2.1 Holistic Hyperproperty Logic $\mathcal{HL}$

Syntax of $\mathcal{HL}$

Clarkson and Schneider specify hyperproperties as first-order predicates on sets of traces [18, 19], using universal and possibly existential quantification over traces ($\forall t \in T$, $\exists t \in T$) in a candidate set T, as well as relations on tuples of traces. The hyperproperties *NI* and *RN* from Chapter 2.4 are such examples. We propose the logical language $\mathcal{HL}$ to formalize hyperproperties specified in this style.

We first present a grammar for specifying coinductive predicates over traces. Let $var_0 = \{a, b, c, \ldots\}$ be a set of variables ranging over alphabet elements and $var_1 = \{x, y, z, \ldots\}$ be a set of variables ranging over traces. The syntax of terms is given as follows:

$$t ::= a \mid x \mid \varepsilon,$$

where ε is a constant, denoting the empty trace.

Let p range over predicates in P and X range over coinductively or inductively-defined predicate variables. Formulae used to specify (the body of) coinductive or inductive predicates on traces have the following syntax:

$$\phi_0 ::= p(\bar{t}) \mid X(\bar{t}) \mid \bot \mid \neg\phi_0 \mid \phi_0 \wedge \phi_0.$$

As usual $\top = \neg\bot$, $\phi \vee \psi = \neg(\neg\phi \wedge \neg\psi)$ and the implication $\phi \rightarrow \psi$ is $\neg\phi \vee \psi$.

Next we can define *holistic hyperproperty logic* $\mathcal{HL}$ with the following syntax:

$$\phi_1 ::= \bot \mid \neg\phi_1 \mid \phi_1 \wedge \phi_1 \mid \exists x.\phi_1 \mid [\mathbf{gfp}\, X(\bar{x}).\phi_0](\bar{t}),$$

where X can occur only positively in ϕ_1. Note that $[\mathbf{gfp}\, X(\bar{x}).\phi_0](\bar{t})$ denotes the greatest fixed point and is used to define coinductive predicates on traces — this means that the terms in $\bar{t}$ range over traces only. The least fixed point operator is dual to the greatest fixed point one, i.e. $[\mathbf{lfp}\, X(\bar{x}).\phi](\bar{t}) = \neg[\mathbf{gfp}\, X(\bar{x}).\neg\phi_0](\bar{t})$, and can be used for inductive definitions. Alternation of the fixed point operators is not allowed in $\mathcal{HL}$. As usual $\forall x.\phi = \neg(\exists x.\neg\phi)$. Note that $x \in X$ iff $X(x) = \mathit{true}$, where the implicit predicate X is an inductively or coinductively defined set. More formally, $X(x) \mathrel{\widehat{=}} [\mathbf{gfp}\, X(x).\phi](t)$ or $X(x) \mathrel{\widehat{=}} [\mathbf{lfp}\, X(x).\phi](t)$.

Semantics of $\mathcal{HL}$

The semantics of $\mathcal{HL}$ is straightforwardly derivable from the semantics of LFP, as presented in Section 2.8. We next introduce the models used for this logic. Consider structures of vocabulary $\sigma = (P,F)$, where $P = \{=, \{p_i \mid i \in \{1,\ldots,n\}\}, \{R_a \mid a \in A\}\}$ is the set of predicates, $F = \{f_i \mid i \in \{1,\ldots,n\}\}$ is the set of functions and ε is the sole constant. Note that the predicates in P typically have the following signatures: $= : A^2 \rightarrow 2$, $p_i : A \rightarrow 2$, $R_a \subseteq A^\infty \times A^\infty$ and the functions can be easily converted into predicates. A σ-structure (model) is $\mathcal{M} = (S, P^{\mathcal{M}}, F^{\mathcal{M}})$, where $S \subseteq \mathsf{Sys}$ and $P^{\mathcal{M}}, F^{\mathcal{M}}$ are P and F relativized to the concrete interpretation of the model. It should be noted that the transition function is converted into a transition relation parameterized by elements of A. Essentially, for any traces s,t we have $(s,t) \in R_a$ (or $s \xrightarrow{a} t$) iff $s(0) = a$ and $s' = t$. If $s = \varepsilon$ (i.e. we have reached the end of a string), we have that $(s,s) \in R_\varepsilon$.

Definition 3.2.1. A *holistic hyperproperty* is the set of systems (alternatively the set of sets of traces) that can be specified by some formula $\phi \in \mathcal{HL}$. The set of all holistic hyperproperties is $\{S \subseteq \mathsf{Sys} \mid S \models \phi\}$, where $\phi \in \mathcal{HL}$.

Sample Hyperproperties in $\mathcal{HL}$

To illustrate the logic, some sample hyperproperties are presented next.

Example 3.2.2. *Consider the hyperproperty FLIP that assures that for every stream in a candidate set, its element-wise opposite is also in the set. Note that this hyperproperty is chosen for its simplicity and is used for illustrative purposes only. Start by defining the predicate flip* $\subseteq A^\omega \times A^\omega$*, relating each stream over* $A = \{0,1\}$ *to*

its element-wise opposite:

$$\mathit{flip} \mathrel{\widehat{=}} \boldsymbol{gfp}\, X(x,y)\,.\,(x \xrightarrow{a} x' \wedge y \xrightarrow{b} y' \wedge (\neg(a=b)) \wedge X(x',y')).$$

The simple hyperproperty FLIP can be given in $\mathcal{HL}$ as:

$$\mathit{FLIP}(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \exists x_1 \in X\,.\,\mathit{flip}(x_0,x_1).$$

In order to make definitions like *flip* more readable, we will typically use a different format for specifying coinductive trace predicates. The format is to simply give the respective inference rules, which are to be interpreted coinductively (see Section 2.2.1 for details). The fact that the rules are interpreted coinductively is indicated using *coind* on the right side of the respective inference rules. To illustrate this format, let us give an equivalent definition of *flip* using inference rules:

$$\frac{\neg(a=b) \qquad x \xrightarrow{a} x' \qquad y \xrightarrow{b} y' \qquad \mathit{flipc}(x',y')}{\mathit{flipc}(x,y)}\ \mathit{coind}$$

The logic proposed here is fairly general as it captures most security-relevant hyperproperties from the original hyperproperties paper [18]; the noteworthy exceptions are service level agreement (SLA) polices such as mean response time (*MRT*), time service factor and percentage uptime. Formal verification of systems with respect to such policies is an inherently difficult problem. For instance, consider *MRT* as a hyperproperty [18]:

$$\mathit{MRT} \mathrel{\widehat{=}} \{T \in \mathsf{Prop} \mid \mathit{mean}(\bigcup_{t \in T} \mathit{respTimes}(t)) \leq 1\}.$$

The function $\mathit{mean}(S)$ specifies the mean of a set S of real numbers, whereas $\mathit{respTimes}(t)$ specifies the set of response times in a trace t. It is generally not clear how to find the mean of a sequence of infinite number of response times in a trace because the series might be diverging; moreover, when *MRT* is defined as a hyperproperty, an additional problem arises, namely that the cardinality of the set of traces might be infinite [18]. It is also not clear how to make such a specification incremental in the general case. We do not address SLA policies in this work. However, we envision that some of the techniques developed in this dissertation can be useful for at least approximating some SLA policies on certain classes of systems.

Next, we demonstrate the proposed logic by presenting three example definitions. First, recall several coinductively-defined trace predicates from Section 2.4. Define ev_Z, filtering the Z-elements of a trace for some $Z \subseteq A$, as follows:

$$\frac{}{ev_Z(\varepsilon) = \varepsilon}\ coind \qquad \frac{ev_Z(x) = y \qquad a \in A \setminus Z}{ev_Z(a \cdot x) = \tau \cdot y}\ coind \qquad \frac{ev_Z(x) = y \qquad a \in Z}{ev_Z(a \cdot x) = a \cdot y}\ coind$$

Note that to make the notation even more readable, we abbreviate it using pattern-matching. For instance, $ev_Z(x) = \tau \cdot y$ stands for $(ev_Z(x))(0) = \tau$ and $(ev_Z(x))' = y$.

Next, coinductively define predicate $no_Z : A^\infty \to 2$ (parameterized by set Z), which states that there are no events from some nonempty set Z in a trace:

$$\frac{}{no_Z(\varepsilon)}\ coind \qquad \frac{a \in A \setminus Z \qquad no_Z(x)}{no_Z(a \cdot x)}\ coind$$

Define *weak trace equivalence* $\approx \subseteq A_\tau^\infty \times A_\tau^\infty$ as follows:

$$\frac{}{\varepsilon \approx \varepsilon}\ coind \qquad \frac{x \approx y}{\tau \cdot x \approx y}\ coind \qquad \frac{x \approx y}{a \cdot x \approx a \cdot y}\ coind$$

This definition has a fairness problem as τ^ω is equivalent to any trace. This also results in the definition being termination insensitive, i.e. it allows leaks via the termination behavior of the system. Finally, for all $x, y \in A^\infty$ and some nonempty set $Z \subseteq A$, define weak trace equivalence with respect to Z as follows: $x \approx_Z y$ iff $ev_Z(x) \approx ev_Z(y)$.

We are now ready to present some security-relevant hyperproperty definitions in $\mathcal{HL}$. Consider McLean's formulation of a policy called *generalized noninterference* [60] for non-deterministic systems. Informally, the policy states that any high-level input behavior is compatible with any low level view of the system. The policy is based on partitioning A into L and H (low and high confidentiality events), and then H into sets HI and HO (high inputs and high outputs). The definition of GNI in $\mathcal{HL}$ is given as follows:

$$GNI(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \forall x_1 \in X\ \exists x_2 \in X . x_2 \approx_{\mathrm{HI}} x_0 \wedge x_2 \approx_{\mathrm{L}} x_1 .$$

If we slightly enhance our model, it is possible to express a variant of observational determinism [87]. Recall that each trace can be seen as a set of states, where the next state is obtained by taking the respective derivative. Let S be a set of states, associated with the traces in a system. Let Var be a set of variables and $L = \{l_1, \ldots, l_k\}$ be a set of low-confidentiality variables, which is a subset of Var. Let Val be a set of values and $V : S \times Var \to Val$ be a valuation. Define $=_L \subseteq S \times S$ as follows:

$$s_1 =_L s_2 \text{ iff for all } l \in L, V(s_1)(l) = V(s_2)(l).$$

Now, we can give a definition of termination insensitive observational determinism [87] as follows:

$$OD(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \forall x_1 \in X . (x_0(0) =_{\mathrm{L}} x_1(0) \to x_0 \approx_{\mathrm{L}} x_1),$$

where $\approx_L$ is an indistinguishability relation on output traces, as defined above. *OD* typically assumes the batch-job model: all variables are already initialized in the start state. Any further changes of the low part of the state should be observable in the resulting traces. The policy says that whenever any two possible executions of some candidate system T are initialized with the same low-confidentiality variables, the resulting low-observable behaviors are indistinguishable with respect to relation $\approx_L$.

Finally, consider the following definition of generalized noninference *GNF* [86]:

$$GNF(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \exists x_1 \in X.\, no_{HI}(x_1) \wedge x_0 \approx_L x_1.$$

The definition of *GNF* says that any candidate system T is secure with respect to the policy iff for every possible trace t_0 in T, there must be a trace t_1 in T with the following properties: t_1 has no high inputs and the projections of t_0 and t_1 to low-confidentiality security events are weak trace equivalent.

3.2.2 Incremental Hyperproperty Logic IL

Syntax of IL

We next present the logical language IL and use it to formalize incremental hyperproperties [63]. IL is also based on a subset of least fixed point logic (details of LFP are available in Section 2.8).

Let $var_0 = \{a, b, \ldots\}$ be a set of variables ranging over alphabet elements in A and $var_1 = \{X, Y, X_a, Y_a, \ldots\}$ be a set of variables ranging over system states (or alternatively over subtrees of the corresponding to the system unique tree). Let a range over var_0, X, Y over var_1 and I over predicate variables. The syntax of terms is given as follows:

$$T ::= Y \mid a \mid \varepsilon.$$

Formulae in IL have the following syntax:

$$\begin{aligned} \psi &::= [\mathbf{gfp}\, I(\overline{Y}).\phi](\overline{T}) \\ \phi &::= p(\overline{T}) \mid I(\overline{T}) \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \forall y.\phi \mid \exists y.\phi, \end{aligned}$$

where p ranges over a set of predicates P, I can occur only positively (under an even number of negations) in ϕ and $[\mathbf{gfp}\, I(\overline{X}).\phi](\overline{T})$ denotes the greatest fixed point. The typical least fixed point operator $[\mathbf{lfp}\, I(\overline{Y}).\phi](\overline{T})$ from LFP is not allowed in IL.

Note that in $[\mathbf{gfp}\, I(\overline{X}).\phi](\overline{T})$ I can be seen as the predicate defined as the greatest solution to the equation $I(\overline{X}) \mathrel{\widehat{=}} \phi$ where $\overline{X}$ are parameters. That is, I is a coinductively

defined tree predicate and it checks if some tuple of terms $\overline{T}$ (of type Sys) satisfies the predicate I.

Definition 3.2.3. An *incremental hyperproperty* is a coinductive predicate on a k-tuple of systems that can be specified by some formula $\phi \in \mathit{IL}$. The set of all incremental hyperproperties is $\{\overline{S} \subseteq \mathsf{Sys}^k \mid \overline{S} \models \phi\}$, where $\phi \in \mathit{IL}$.

Semantics of IL

The semantics of IL is straightforwardly derivable from the semantics of LFP, as presented in Section 2.8. The main difference to $\mathcal{HL}$ is that now there are only coinductive predicates and these are specified over trees.

We next present the models used for this logic. Consider structures of vocabulary $\sigma_1 = (P,F)$, where $P = \{=, \{p_i \mid i \in \{1,\ldots,n\}\}, \{R_a \mid a \in A\}\}$ is the set of predicates, $F = \{f_i \mid i \in \{1,\ldots,n\}\}$ is the set of functions and ε is the sole constant, here denoting the empty tree. Note that the predicates in P have the following signatures: $= : A^2 \to 2$, $p_i : A \to 2$, $R_a \subseteq \mathsf{Sys} \times \mathsf{Sys}$. A σ_1-structure (model) is $\mathcal{M} = (\overline{S}, P^{\mathcal{M}}, F^{\mathcal{M}})$, where $\overline{S} \subseteq \mathsf{Sys}^k$ and $P^{\mathcal{M}}, F^{\mathcal{M}}$ are P and F relativized to the concrete interpretation of the model. It should be noted that the transition function is converted into a transition relation parameterized by elements of A. Moreover, the transition function is implicitly relativized to a single tree in $\overline{S}$. Formally, for any trees S,T we have $(S,T) \in R_a$ (or $S \xrightarrow{a} T$) iff $\mathit{test}(a)(S) = \mathit{true}$ and $t(S)(a) = T$, where $S \in \mathit{SubT}(\overline{S}_i)$ and $T \in \mathit{SubT}(\overline{S}_i)$ for some $i \in 1,\ldots,k$. Note that $\overline{S}_i$ gives the i-th tree in $\overline{S}$.

3.2.3 Sample Incremental Hyperproperties in IL

As an example, consider the coinductive tree predicate $\mathit{FLIP}'(x,y)$, giving the incremental version of *FLIP* from Example 3.2.2 and defined as follows:

$$\mathit{FLIP}' \mathrel{\widehat{=}} \mathbf{gfp}\, I(X,Y).(\forall X_a.X \xrightarrow{a} X_a \to (\exists Y_b.Y \xrightarrow{b} Y_b \wedge \neg(b=a) \wedge I(X_a,Y_b))).$$

Definition 3.2.4. A hyperproperty $H \in \mathcal{HL}$ is *incrementalizable* iff there exists an $H' \in \mathit{IL}$ such that for all $T \subseteq \mathsf{Sys}$ we have that $H(T) \equiv H'(\bar{T})$.

The logic is general enough to capture a number of interesting incremental hyperproperties (examples will be presented in Sections 3.3 and 3.4 of this chapter, as well as in Chapters 4 and 5). In the next section we introduce the process of incrementalization — one method to arrive at incremental hyperproperties. Further, we demonstrate the incrementalization process on three classes of holistic hyperproperties.

3.3 Incrementalization of Holistic Hyperproperties

In this section, we present a syntactic approach to incrementalization for three classes of holistic hyperproperties. The first class is called PHH and it includes an existential and universal quantifiers on traces, as well as pointwise coinductive predicates, similar to the ones proposed by Niqui and Rutten [66] (with the restriction that the pointwise relation is functional). The class does not seem to contain security-relevant hyperproperties and is used mainly for illustrative purposes, as well as to establish a connection with related work [66]. The second class is called SHH and consists of security-relevant, holistic hyperproperties defined on infinite systems, e.g. *NI* from Section 2.4. The definitions in class SHH state that a system is secure if its set of traces is closed under removal of a certain type of events, e.g. confidential ones (see, for instance, [60]). Hyperproperties in this class are liveness ones. The third class is called OHH, a class of security-relevant, holistic hyperproperties, including termination-sensitive variants of observational determinism. This class contains (some) 2-safety hyperproperties, i.e. the policy can be refuted by presenting two traces that do not satisfy the respective condition. In this section, we give a detailed explanation of the incrementalization process for the first class. The details for all of them are available in Appendix A.

At a high level, incrementalization of a holistic hyperproperty H amounts to finding a coinductive tree/system predicate H' and a functional $\Psi_{H'}$ such that H' is the greatest fixed point of $\Psi_{H'}$ (i.e. $H' = \mathbf{gfp}\,\Psi_{H'}$) and for all $T \subseteq \mathsf{Sys}$, the equivalence $H(T)$ iff $H'(T,\ldots,T)$ holds. An incremental hyperproperty is based on a monotone function and hence a verification methodology can be developed (see Section 3.4). In essence, we lift the fixed point operator defining coinductive (and/or inductive) trace predicates in a holistic specification to the outermost level in an incremental specification on trees. The techniques used include generalizing the definition of H to n parameters, unfolding H and rewriting it using derivatives, unfolding the coinductive definitions, swapping quantifiers, rearranging expressions and folding the holistic definition. This process results in an incremental definition equivalent to H.

3.3.1 The Class PHH and its Incrementalization

In this section, we define the class of holistic hyperproperties PHH and then present its incrementalization. Let c_R be a pointwise, coinductive predicate parameterized by some functional relation $R \subseteq A \times A$ (i.e. R can be seen as a function) and defined as follows:

$$\frac{}{c_R(\varepsilon,\varepsilon)}\ \textit{coind} \qquad \frac{a\,R\,b \qquad c_R(x,y)}{c_R(a\cdot x, b\cdot y)}\ \textit{coind}$$

Definition 3.3.1. Let PHH_R be the class of *pointwise, holistic hyperproperties* defined in $\mathcal{HL}$ as follows:

$$\mathsf{PHH}_R(X) \mathrel{\widehat{=}} \forall x \in X\ \exists y \in X. c_R(x,y).$$

Both PHH and the coinductive predicate c are explicitly parameterized by R. However, this will be left out from the notation in the rest of this work.

We next proceed with the incrementalization of PHH. Start by generalizing PHH to take a pair of systems as parameters as follows:

$$\mathsf{PHH}^2(X,Y) \mathrel{\widehat{=}} \forall x \in X\ \exists y \in Y. c(x,y).$$

Clearly, we have that for all $T \subseteq \mathsf{Sys}$, $\mathsf{PHH}(T)$ iff $\mathsf{PHH}^2(T,T)$. Each of the following lemmas is one or more steps of the incrementalization process. First, unfold the holistic definition of PHH^2.

Lemma 3.3.2. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$(\varepsilon \in X \to \varepsilon \in Y) \wedge\ (\forall a \in A\ \forall w \in A^\infty.\ aw \in X \to \exists b \in A\ \exists u \in A^\infty. bu \in Y \wedge\ a\,R\,b \wedge c(aw,bu)).$$

Second, rewrite the definition using derivatives and unfold the coinductive definition of c once.

Lemma 3.3.3. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$(o(X) \to o(Y)) \wedge\ (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \to (\forall w \in A^\infty.\ w \in X_a \to \exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{b} Y_b \wedge a\,R\,b \wedge \exists u \in A^\infty.\ u \in Y_b \wedge c(w,u))).$$

Third, swap the quantifiers $\exists b$ and $\forall w$; this can be done as b depends only on a.

Lemma 3.3.4. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$(o(X) \to o(Y)) \wedge\ (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \to (\exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{b} Y_b \wedge a\,R\,b \wedge (\forall w \in A^\infty.\ w \in X_a \to \exists u \in A^\infty.\ u \in Y_b \wedge c(w,u)))).$$

Fourth, rearrange the resulting expression and fold the definition of PHH^2.

Lemma 3.3.5. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$o(X) \to o(Y)$$

$$\wedge\ (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \to \exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{b} Y_b \wedge a\,R\,b\ \wedge \mathsf{PHH}^2(X_a,Y_b)).$$

Finally, define the incremental hyperproperty PIH^2 as follows:

$$\mathsf{PIH}^2 \mathrel{\widehat{=}} \mathbf{gfp}\, I(X,Y).\ (o(X) \to o(Y)$$
$$\wedge\, (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \to \exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{b} Y_b \wedge a\, R\, b \wedge I(X_a, Y_b))).$$

Theorem 3.3.6 (Incrementalization of PHH^2)**.** *For all $S,T \subseteq \mathsf{Sys}$, we have that $\mathsf{PHH}^2(S,T)$ iff $\mathsf{PIH}^2(S,T)$.*

Corollary 3.3.7. *For all $T \subseteq \mathsf{Sys}$, we have that $\mathsf{PHH}(T)$* iff *$\mathsf{PIH}^2(T,T)$.*

Corollary 3.3.7 illustrates the fundamental significance of incrementalization: for verification purposes, one may ignore the holistic hyperproperty (e.g. PHH) and verify the incremental hyperproperty (e.g. PIH^2) instead.

3.3.2 The Class SHH and its Incrementalization

We next present the class SHH of security-relevant, holistic hyperproperties defined on infinite systems. The definitions in class SHH state that a system is secure if its set of traces is closed under removal of confidential (high) events (see, for instance, [60]).

Let $p : A \to 2$ be a predicate and $f : A \to A$ a function. Define the coinductive predicate $\sim_{\mathrm{p}} : A^{\omega} \times A^{\omega} \to 2$ as follows:

$$\frac{p(a) \qquad p(b) \qquad x \sim_{\mathrm{p}} y \qquad b = f(a)}{a \cdot x \sim_{\mathrm{p}} b \cdot y}\ \mathit{coind} \qquad \frac{\neg p(a) \qquad p(b) \qquad x \sim_{\mathrm{p}} b \cdot y}{a \cdot x \sim_{\mathrm{p}} b \cdot y}\ \mathit{coind}$$

$$\frac{p(a) \qquad \neg p(b) \qquad a \cdot x \sim_{\mathrm{p}} y}{a \cdot x \sim_{\mathrm{p}} b \cdot y}\ \mathit{coind}$$

Define the coinductive predicate $p_s : A^{\omega} \to 2$, generalizing no_H from Section 2.4:

$$\frac{p(a) \qquad p_s(x)}{p_s(a \cdot x)}\ \mathit{coind}$$

To define SHH, an additional restriction on Sys_{ω} is needed. First note that we can convert any predicate $p : A \to 2$ to a propositional constant as follows: for a trace t we say that $t \models p$ iff $p(t(0))$ holds. Then we will work only with systems satisfying the

property $P_{\Box\Diamond} = \{t \in A^\omega \mid t \models \Box\Diamond p\}$, based on temporal logic modalities *eventually* ($\Diamond$) and *always* ($\Box$) [68]. This means that for this section, the formal parameters X, Y are of type $P_{\Box\Diamond}$.

We next give some motivation and intuition about the restriction for systems to be in the class $P_{\Box\Diamond}$. First, note that the predicate $p : A \to 2$ could be thought of as denoting the visibility of events to agents at a certain security level. Let us call events for which p evaluates to true *p-events*, dually there are $\neg p$-events. Intuitively, many security-relevant systems (e.g. reactive systems such as servers) have infinite traces and can be characterized as follows: each trace has some p-event appearing eventually, and that happens infinitely often. These are the type of properties we expect from a server, for instance: each non-confidential request needs to be eventually serviced and that should happen infinitely often. The latter is captured by the property $P_{\Box\Diamond}$. $P_{\Box\Diamond}$ is a liveness property, as defined by Alpern and Schneider [5]: formally, such a liveness property is given as follows: $\forall \alpha \in A^* \; \exists \beta \in A^\omega . \alpha\beta \models \Box\Diamond p$.

Definition 3.3.8. Let $\mathsf{SHH}_{p,f}$ be the class of holistic hyperproperties parameterized by predicate p and function f. Recall that X ranges over $P_{\Box\Diamond}$. $\mathsf{SHH}_{p,f}$ is defined as follows:

$$\mathsf{SHH}_{p,f}(X) \mathrel{\widehat{=}} \forall x \in X \; \exists y \in X .\; p_s(y) \wedge x \sim_{\mathrm{p}} y$$

In the rest of this work we leave out the parameters and write SHH instead of $\mathsf{SHH}_{p,f}$. In order to work with only one predicate, we combine p_s and $\sim_{\mathrm{p}}$ into a new coinductive predicate c_1:

$$\frac{p(a) \qquad p(b) \qquad f(a) = b \qquad c_1(x,y)}{c_1(a \cdot x, b \cdot y)}\ \mathit{coind} \qquad \frac{\neg p(a) \qquad c_1(x,y)}{c_1(a \cdot x, y)}\ \mathit{coind}$$

Thanks to the following lemmas we can change the original definition of SHH with an equivalent definition.

Lemma 3.3.9. *For all $s,t \in A^\omega$, we have that $(p_s(t) \wedge s \sim_{\mathrm{p}} t) \to c_1(s,t)$.*

Lemma 3.3.10. *For all $s,t \in P_{\Box\Diamond}$, we have that $c_1(s,t) \to (p_s(t) \wedge s \sim_{\mathrm{p}} t)$.*

The equivalent definition of SHH by Lemmas 3.3.9 and 3.3.10 is given as follows: for all $T \in P_{\Box\Diamond}$

$$\mathsf{SHH}(T) \iff \forall x \in T \; \exists y \in T .\; c_1(x,y).$$

First, we generalize SHH to take a pair of systems as a parameter as follows:

$$\mathsf{SHH}^2(X,Y) \mathrel{\widehat{=}} \forall x \in X \; \exists y \in Y .\; c_1(x,y).$$

Clearly, for all $T \subseteq P_{\Box\Diamond}$, it is the case that $\mathsf{SHH}(T)$ iff $\mathsf{SHH}^2(T,T)$.

We next continue with similar steps as in the incrementalization of PHH from Section 3.3.1. The steps can be used to derive the following version of SIH^2 (details available in Appendix A):

$$\mathsf{SIH}^2 \mathrel{\widehat{=}} \mathbf{gfp}\, I(X,Y).$$

$$(\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ (X \xrightarrow{a} X_a \wedge p(a)) \rightarrow \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{f(a)} Y_b \wedge I(X_a, Y_b)$$

$$\wedge\, \forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ (X \xrightarrow{a} X_a \wedge \neg p(a)) \rightarrow I(X_a, Y)).$$

Theorem 3.3.11 (Incrementalization of SHH^2)**.** *For all $S,T \subseteq P_{\Box\Diamond}$, we have that* $\mathsf{SHH}^2(S,T)$ iff $\mathsf{SIH}^2(S,T)$.

Corollary 3.3.12. *For all $T \subseteq P_{\Box\Diamond}$, we have that* $\mathsf{SHH}(T)$ iff $\mathsf{SIH}^2(T,T)$.

Corollary 3.3.12 tells us that reasoning about the incremental hyperproperty SIH^2 is enough for verification of the holistic SHH. In essence, the problem of holistic hyperproperty verification is reduced to reasoning about incremental specifications. Because the incremental SIH^2 is the greatest fixed point of a monotone operator, a multitude of techniques for calculating such a fixed point are possible (such techniques will be illustrated in Section 3.4 and will be further explored in Chapters 5 and 6).

3.3.3 The Class OHH and its Incrementalization

In this section we present OHH, a class of security-relevant, holistic hyperproperties defined on systems in Sys, and its incrementalization. Given a predicate $p : A \rightarrow 2$ and a function $f : A \rightarrow A$, define $\sim_{\mathrm{pt}} : A^\infty \times A^\infty \rightarrow 2$, on traces as follows:

$$\frac{}{\varepsilon \sim_{\mathrm{pt}} \varepsilon}\ \textit{coind} \qquad \frac{p(a) \qquad p(b) \qquad b = f(a) \qquad x \sim_{\mathrm{pt}} y}{a \cdot x \sim_{\mathrm{pt}} b \cdot y}\ \textit{coind}$$

$$\frac{\neg p(a) \qquad \neg p(b) \qquad x \sim_{\mathrm{pt}} y}{a \cdot x \sim_{\mathrm{pt}} b \cdot y}\ \textit{coind}$$

Note that $\sim_{\mathrm{pt}}$ is actually parameterized by p and f, but we do not indicate it explicitly to make notation more reasonable. Let $\sim_{\mathrm{pt_i}}$ be the restriction of the relation on input elements in A_i, given as follows:

$$\frac{}{\varepsilon \sim_{\mathrm{pt_i}} \varepsilon}\ \textit{coind} \qquad \frac{p(a) \qquad p(b) \qquad a \in A_i \qquad b \in A_i \qquad b = f(a) \qquad x \sim_{\mathrm{pt_i}} y}{a \cdot x \sim_{\mathrm{pt_i}} b \cdot y}\ \textit{coind}$$

$$\frac{\neg p(a) \qquad \neg p(b) \qquad x \sim_{\mathrm{pt_i}} y}{a \cdot x \sim_{\mathrm{pt_i}} b \cdot y}\ \textit{coind}$$

$$\frac{p(a) \qquad p(b) \qquad a \in A_o \qquad b \in A_o \qquad x \sim_{\mathrm{pt_i}} y}{a \cdot x \sim_{\mathrm{pt_i}} b \cdot y}\ \textit{coind}$$

Definition 3.3.13. Let $\mathsf{OHH}_{p,f}$ be the class of holistic hyperproperties parameterized by predicate p and function f and defined as follows:

$$\mathsf{OHH}_{p,f}(X) \mathrel{\widehat{=}} \forall x \in X\ \forall y \in X.\ (x \sim_{\mathrm{pt_i}} y \to x \sim_{\mathrm{pt}} y).$$

In what follows, we will just write OHH. Again, the first step is to generalize OHH to take a pair of systems as a parameter as follows:

$$\mathsf{OHH}^2(X,Y) \mathrel{\widehat{=}} \forall x \in X\ \forall y \in Y.\ (x \sim_{\mathrm{pt_i}} y \to x \sim_{\mathrm{pt}} y).$$

The process of incrementalization is used to derive the following, incremental version of OHH^2 (details available in Appendix A):

$$\begin{aligned}
\mathsf{OIH}^2 \mathrel{\widehat{=}} \mathbf{gfp}\, I(X,Y).\ & (o(X) \leftrightarrow o(Y)) \\
& \wedge \forall a,b \in A_i\ \forall X_a, Y_b \subseteq Sys.\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{f(a)} Y_b \wedge p(a) \to I(X_a, Y_b) \\
& \wedge \forall a,b \in A_o\ \forall X_a, Y_b \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(b) \to \\
& \qquad\qquad b = f(a) \wedge I(X_a, Y_b) \\
& \wedge \forall a,b \in A\ \forall X_a, Y_b \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge \neg p(a) \wedge \neg p(b) \to I(X_a, Y_b)).
\end{aligned}$$

Theorem 3.3.14 (Incrementalization of OHH^2)**.** *For all $S,T \subseteq \mathsf{Sys}$, we have that $\mathsf{OHH}^2(S,T)$ iff $\mathsf{OIH}^2(S,T)$.*

Corollary 3.3.15. *For all $T \subseteq \mathsf{Sys}$, we have that $\mathsf{OHH}(T)$ iff $\mathsf{OIH}^2(T,T)$.*

An interesting, security-relevant hyperproperty in OHH is weak time-sensitive noninterference [4]. To formalize it, first define its key ingredient $\sim_{\mathrm{ts}}$, which is an instantiation of $\sim_{\mathrm{pt}}$ with f being the identity function and $p = low = \neg high$:

$$\frac{}{\varepsilon \sim_{\mathrm{ts}} \varepsilon}\ \textit{coind} \qquad \frac{low(a) \qquad x \sim_{\mathrm{ts}} y}{a \cdot x \sim_{\mathrm{ts}} a \cdot y}\ \textit{coind} \qquad \frac{high(a) \qquad high(b) \qquad x \sim_{\mathrm{ts}} y}{a \cdot x \sim_{\mathrm{ts}} b \cdot y}\ \textit{coind}$$

Note that this definition guarantees that both traces terminate in an equal number of steps and are low-view indistinguishable, or both diverge and are still low-view indistinguishable. Thus, the definition is also *termination-sensitive*.

Let $\sim_{\mathrm{ts_i}}$ be the same as $\sim_{\mathrm{ts}}$, but restricted to inputs. To formalize it, first define its key ingredient $\sim_{\mathrm{ts}}$, which is an instantiation of $\sim_{\mathrm{pt}}$ with f being the identity function and $p = low = \neg high$:

$$\frac{}{\varepsilon \sim_{\mathrm{ts_i}} \varepsilon}\ coind \qquad \frac{low(a) \qquad a \in A_i \qquad x \sim_{\mathrm{ts_i}} y}{a \cdot x \sim_{\mathrm{ts_i}} a \cdot y}\ coind$$

$$\frac{high(a) \qquad high(b) \qquad x \sim_{\mathrm{ts_i}} y}{a \cdot x \sim_{\mathrm{ts_i}} b \cdot y}\ coind$$

$$\frac{low(a) \qquad low(b) \qquad a \in A_o \qquad b \in A_o \qquad x \sim_{\mathrm{ts_i}} y}{a \cdot x \sim_{\mathrm{ts_i}} b \cdot y}\ coind$$

Based upon the former definitions, define *weak time-sensitive noninterference*:

$$WTSNI(X) \mathrel{\widehat{=}} \forall x \in X\ \forall y \in X.(x \sim_{\mathrm{ts_i}} y \rightarrow x \sim_{\mathrm{ts}} y).$$

It should be noted that the definitions of $\sim_{\mathrm{ts}}$ and $\sim_{\mathrm{ts_i}}$ exploit the time-sensitivity of *WTSNI*. This takes care of issues such as fairness and productivity, which are encoded in the definition itself, and allows the incremental definition to be elegant. Intuitively, the reason is that the relations are defined in a way that always requires both (related) states to advance. In contrast, this is not the case in Section 3.3.2 with relation $\sim_{\mathrm{p}}$; hence, there we needed the extra restriction $P_{\Box\Diamond}$ to guarantee the fairness and productivity of the definitions in Section 3.3.2.

3.4 Towards Verification of Incremental Hyperproperties

Incremental hyperproperties are based on monotone operators and hence enjoy a feasible verification methodology. In this section, we explore some first steps towards verification, the topic is continued in Chapters 5 and 6.

First, recall some fixed point theory. Let $\Psi : 2^{X_1 \times \ldots \times X_n} \rightarrow 2^{X_1 \times \ldots \times X_n}$ be a monotone operator over the complete lattice of set-theoretic n-ary relations on the Cartesian product of sets X_i, $i \in 1..n$, and $\mathbf{gfp}(\Psi)$ be the greatest fixed point of Ψ. For any

$\bar{x} = (x_1, \ldots, x_n) \in X_1 \times \ldots \times X_n$ and $R \subseteq X_1 \times \ldots \times X_n$ the following principle is sound:

$$\frac{R(\bar{x}) \qquad R \subseteq \Psi(R)}{\mathbf{gfp}(\Psi)(\bar{x})} \tag{3.1}$$

This is the *Knaster-Tarski coinduction principle* [48], generalized to n-ary relations.

The process of incrementalization makes the original holistic definition H irrelevant for verification. Instead, the resulting coinductive predicate H' has to be verified to show that H holds. For a coinductive predicate H', we introduce the notion of an *H'-simulation*.

Definition 3.4.1. Let H' be an incremental hyperproperty and $\Psi_{H'}$ be its respective functional, known to be monotone by definition (as H' is expressible in IL). An *H'-simulation* is an n-ary relation R such that $R \subseteq \Psi_{H'}(R)$. We define *H'-similarity* to be the union of all H'-simulations.

Corollary 3.4.2. *Finding an H'-simulation on some n-tuple of systems $\overline{T}$ is sufficient for showing that $H'(\overline{T})$ holds.*

The verification of a holistic hyperproperty H would typically take two steps. First, one needs to find an appropriate (equivalent) notion of an H'-simulation. It is typically hard to massage the definition of H into an H'-simulation; one way to tackle this problem is to use some of the incrementalization techniques presented in this chapter. The most difficult part of the problem is typically the swapping of the universal and existential quantifiers (an important property behind some well-known theorems in analysis). Fortunately, the incrementalization has to be done only once for each holistic hyperproperty class H. Second, one needs to find a specific H'-simulation for the system of interest. Showing that there is no H'-simulation implies that the coinductive predicate does not hold and hence H does not hold. The second step of searching for an H'-simulation relation on the system of interest can be automatic, adapting known model checking techniques. The soundness of this verification methodology follows directly from Knaster-Tarski's coinduction principle [48].

Note that the statement "no H'-simulation implies H does not hold" is valid only in the case when H' is equivalent to H. If we can find some H' such that $H'(\overline{T})$ only implies $H(T)$, we may only use H' to verify that $H(T)$ holds; if there is no H'-simulation relation, we do not know whether H holds for the system of interest or not. This is important to note, as sometimes it may not be possible to find an equivalent definition, however the techniques presented here may still be used to prove systems secure, provided that a proper definition of H' (i.e. $H'(\overline{T}) \rightarrow H(T)$) is found.

Theorem 3.4.3. *The predicate $H'(S_1, \ldots, S_k)$ on G-systems (S_i, α_i, x_i) for $i \in 1..k$ holds iff there exists some H'-simulation Q s.t. the k-tuple of the start states $(x_1, \ldots, x_k) \in Q$.*

It should be noted that Theorem 3.4.3 is not constructive: it does not give an algorithm for finding H'-simulations. Nevertheless, model checking techniques provide a practical means to compute or approximate the greatest fixed point of a monotone operator at least on finite-state systems. There are well-known iterative schemata for computing the greatest fixed point on a complete lattice [82]. We are usually interested in the complete lattice (2^S) of the state space S of interest. In this case, the bottom and top elements of the lattice are the empty set and the set S respectively, the partial order relation is set inclusion. When approximating (or calculating for finite state spaces) the greatest fixed point one starts with S and iteratively applies the functional Ψ. Formally $\Psi^0 = id$ and $\Psi^{n+1} = \Psi \circ \Psi^n$ where id is the identity function and operator $\circ$ denotes composition. The greatest fixed point of Ψ can be found as follows: $\mathbf{gfp}\,\Psi = \bigcap_{n\geq 0} \Psi^n(S)$. Finding algorithms for deciding whether an H'-simulations holds in general will be explored in Chapters 5 and 6.

We next illustrate the feasibility of verification of incremental hyperproperties from the classes PHH, SHH and OHH presented above.

3.4.1 Sample Hyperproperties in PHH

Consider the holistic hyperproperty *FLIP* from Section 3.2.1: it is in the class PHH. Hence, we can instantiate the incremental definition of PIH^2 and find the appropriate notion of *FLIP*′-simulation. A *FLIP*′-simulation on systems S,T is a relation $Q \subseteq S \times T$, s.t. for all $s \in S, t \in T$ we have the following: if $(s,t) \in Q$ then

$$o(s) \rightarrow o(t)$$

$$\wedge\ \forall a \in A\ \forall s_a \in \mathsf{Sys}.\ s \xrightarrow{a} s_a \rightarrow \exists b \in A\ \exists t_b \in \mathsf{Sys}.\ t \xrightarrow{b} t_b \wedge \neg(b = a) \wedge (s_a, t_b) \in Q.$$

Note that the elements of Sys may be seen as both states and sets of traces, as discussed in Section 3.1. Thus we may write both $s_a \in \mathsf{Sys}$ and $X_a \subseteq \mathsf{Sys}$.

Example 3.4.4. *Consider the system T_1, two copies of which are represented by the automata in Figure 3.3. To (attempt to) show that FLIP(T_1) holds, we have to search for a FLIP′-simulation relation, containing the pair (s_0, s_0). This can be done iteratively: by using the definition of FLIP′ we see that pairs (s_0, s_1) and (s_1, s_0) have to also be in such a relation. We arrive at relation $Q = \{(s_0, s_0), (s_0, s_1), (s_1, s_0)\}$: this is the needed FLIP′-simulation and the reader is invited to check this. Hence we conclude that FLIP(T_1) holds, i.e. $T_1 \models$ FLIP.*

Example 3.4.5. *Let T_2 be the system resulting from removing the transition from state s_1 to itself from T_1 (see Figure 3.4). The goal is again to check whether FLIP(T_2) holds. We show that there is no FLIP′-simulation Q such that $(s_0, s_0) \in Q$. To that end, assume that there were such a Q. By the assumption we have that (s_0, s_0) is*

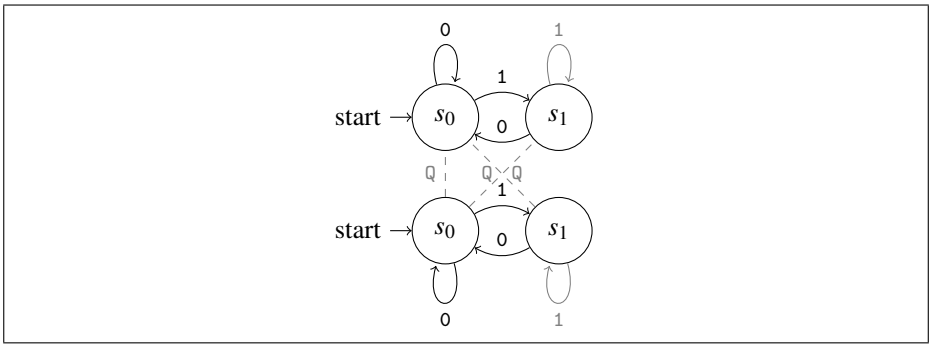


Figure 3.3: Illustration of a *FLIP′*-simulation

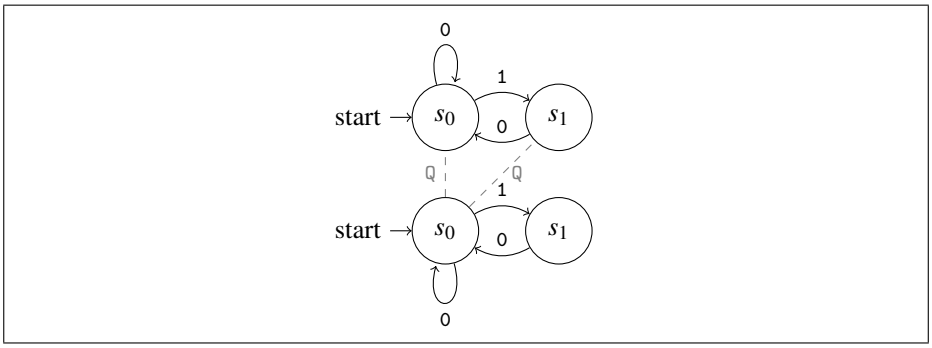


Figure 3.4: Illustration of the lack of a *FLIP′*-simulation

in Q. By the definition of FLIP′-simulation it follows that the pair (s_1, s_0)*, resulting from* $(t(s_0)(1), t(s_0)(0))$*, and* (s_0, s_1)*, resulting from* $(t(s_0)(0), t(s_0)(1))$*, should be in Q. As it should be that* $(s_1, s_0) \in Q$*, it follows that* $(t(s_1)(1), t(s_0)(0))$ *should also be in Q. However, this is not the case as* $t(s_1)(1) = \bot$*, whereas* $t(s_0)(0) = s_0$*. This is a contradiction, implying that there is no Q that is a FLIP′-simulation and* $(s_0, s_0) \in Q$*. Hence we conclude FLIP*(T_2) *does not hold, i.e.* $T_2 \not\models$ *FLIP.*

3.4.2 Sample Hyperproperties in SHH

Consider *NI* from Section 2.4 on systems satisfying $P_{\Box\Diamond}$. *NI* is in the class SHH. Hence we can instantiate the definition of SIH^2 and get an incremental definition of *NI*, called *NI′* and corresponding to an *NI′*-simulation. An *NI′*-simulation on systems S, T is a relation $Q \subseteq S \times T$, s.t. for all $s \in S, t \in T$ we have the following: if $(s,t) \in Q$

then

$$\forall a \in A\ \forall s_a \in \mathsf{Sys}\ .\ (s \xrightarrow{a} s_a \wedge \neg high(a)) \rightarrow \exists t_a \in \mathsf{Sys}\ .\ t \xrightarrow{a} t_a \wedge (s_a, t_a) \in Q$$

$$\wedge\, \forall a \in A\ \forall s_a \in \mathsf{Sys}\ .\ (s \xrightarrow{a} s_a \wedge high(a)) \rightarrow (s_a, t) \in Q.$$

Example 3.4.6. *Consider alphabet $A = \{a,b,c,d\}$ and define predicate high as $high(a)$, $high(b)$, $\neg high(c)$, $\neg high(d)$. Take the system T given by the omega-regular expression $(acbd|cd)^{\omega}$ (see Figure 3.5). To check whether $NI(T)$ holds, we have to search for an NI'-simulation relation, containing the pair (T,T). Using the same iterative approach as in Examples 3.4.4 and 3.4.5, we obtain the relation Q, defined as follows: $Q = \{(T,T),(T_a,T),(T_{ac},T_c),(T_{acb},T_c),(T_c,T_c)\}$. Q is an NI'-simulation such that $(T,T) \in Q$ and thus we conclude that $NI(T)$ holds, i.e. $T \models NI$.*

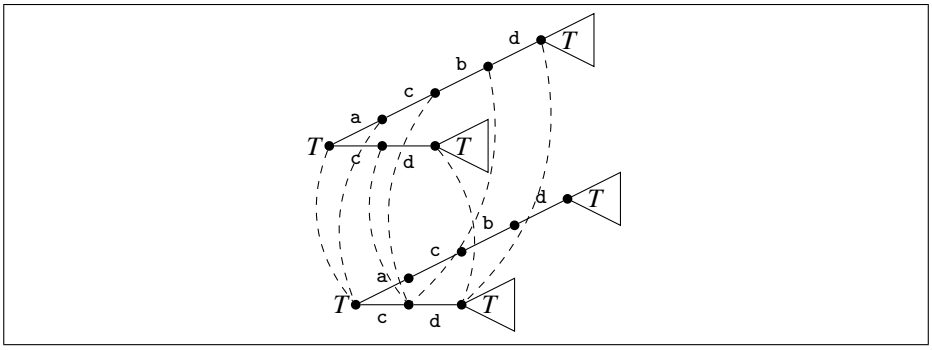


Figure 3.5: Illustration of an NI'-simulation

Note that in the last example, we implicitly switched from reasoning on states of a G-coalgebra (of the fixed functor $GX = 2 \times (1+X)^A$) to reasoning on trees representing the system behavior. This can be done, as explained in Section 3.1, thanks to the unique homomorphism from any state space S to the final G-coalgebra $\mathcal{L}$.

3.4.3 Sample Hyperproperties in OHH

Recall hyperproperty *WTSNI* from the class OHH. We can get an incremental definition, corresponding to a *WTSNI*$'$-simulation. A *WTSNI*$'$-simulation on systems S,T is a relation $Q \subseteq S \times T$ such that for all $s \in S$ and $t \in T$ we have the following: if

$(s,t) \in Q$ then

$o(s) \leftrightarrow o(t)$

$\wedge \forall a \in A_i \, \forall s_a, t_a \in \mathsf{Sys} \, . \, (s \xrightarrow{a} s_a \wedge t \xrightarrow{a} t_a \wedge \neg high(a)) \rightarrow (s_a, t_a) \in Q$

$\wedge \forall a, b \in A_o \, \forall s_a, t_a \in \mathsf{Sys} \, . \, (s \xrightarrow{a} s_a \wedge t \xrightarrow{b} t_b \wedge \neg high(a) \wedge \neg high(b)) \rightarrow (a = b \wedge (s_a, t_a) \in Q)$

$\wedge \forall a, b \in A \, \forall s_a, t_a \in \mathsf{Sys} \, . \, (s \xrightarrow{a} s_a \wedge t \xrightarrow{b} t_b \wedge high(a) \wedge high(b)) \rightarrow (s_a, t_b) \in Q.$

To illustrate the applicability of our abstract notions to programs, consider the RIMP program called P:

```
input ch_H(x) {i := x;}
input ch_L(x) {if i <= x then  output ch_L(0);
               else output ch_L(1);}
```

Note that inputs come from the set $\{0,1\}$ and outputs from $\{0,1,\tau\}$. The possible behaviors of two copies (called P_0 and P_1) of program P are presented visually in Figure 3.6. We show that there is no *WTSNI′*-simulation Q such that $(P_0, P_1) \in Q$. To that end assume there were such a Q; states that should be related in any *WTSNI′*-simulation, including the pair of start states, are connected by the dashed line in Figure 3.6. By the assumption and the definition of *WTSNI′*-simulation, it follows that $(t(P_0)(ch^i_H(1)), t(P_1)(ch^i_H(0))) \in Q$. We can continue using the definition of *WTSNI′*-simulation to find further pairs of states that have to be in the hypothetical relation Q (these are given in Figure 3.6). One of these is the pair (s_0, s_1). If $(s_0, s_1) \in Q$, then the definition requires that $ch^o_L(1) = ch^o_L(0)$, by the rule for low outputs. This is a contradiction as the outputs clearly differ, thus there is no Q that is a *WTSNI′*-simulation and $(P_0, P_1) \in Q$. Hence $P \not\models WTSNI$. Note that program P is also insecure with respect to other, less strict definitions, such as *RN* (see Section 2.4).

To make a contrast between termination-sensitive and insensitive definitions, consider program P_2:

```
input ch_H(x) {i:=x; if i = 0 then while 1 do skip};
output ch_L(0);
```

Program P_2 is termination-insensitive noninterferent (for instance, with respect to the definition of reactive noninterference *RN* from Section 2.4). To see this, let $\sigma_{in} = [ch^i_H(1)]$ and $\gamma_{in} = [ch^i_H(0)]$ be input strings. Clearly, $\sigma_{in} \approx_L \gamma_{in}$ and the resultant traces are $\sigma = [ch^i_H(1), \tau, \tau, \tau, ch^o_L(0), \tau]$ and $\gamma = [ch^i_H(0), \tau, \tau, \tau, \ldots]$. Since $\sigma \approx_L \gamma$, and σ and γ are all traces, it follows that P_2 is secure. However, this program is not secure w.r.t. *WTSNI*. The reader is invited to check that this is indeed the case.

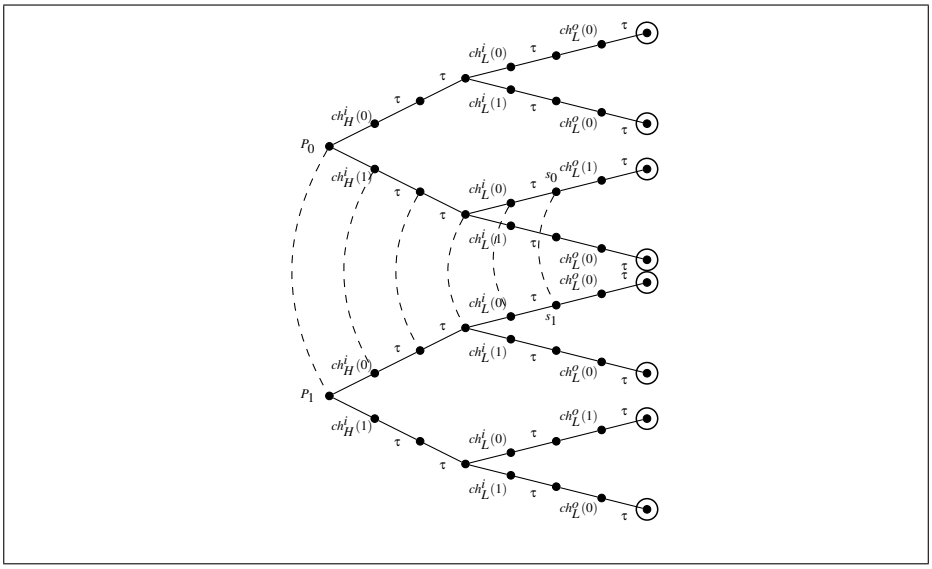


Figure 3.6: Illustration of the lack of a $WTSNI'$-simulation for program P

3.5 Discussion

The process of incrementalization of some holistic hyperproperty H may result in an incremental hyperproperty H' that only implies H. This means that although for all $T \subseteq \mathsf{Sys}$, $H'(\overline{T}) \rightarrow H(T)$, it is possible to have a system $T_e \subseteq \mathsf{Sys}$, for which $H(T_e)$ holds, however $\neg H'(\overline{T})$. In such cases we may say that H' under-approximates H. Such definitions of H' can be useful to (only) show that a system satisfies a policy. Intuitively, the reason is that H'-simulations are more fine-grained relations than holistic predicates on sets of traces. This is a generalization of the known fact that although bisimilarity implies trace equivalence, the converse is not necessarily true [27]; in other words, we may have systems that are trace equivalent but not bisimilar. Further, it is possible (in theory) that the definition of H' (seen as a set of systems) specifies the empty set and in such cases the verification methodology via H'-simulations cannot be used. However, we have not seen such a case in practice. We acknowledge that the problem needs further exploration.

The holistic hyperproperties we propose capture the known security-relevant hyperproperties (but not SLA policies on systems, such as percentage uptime, mean response time etc. [18, 19]), however holistic hyperproperties are a strict subset of hyperproperties. For instance, the hyperproperty

$$H(T) \mathrel{\widehat{=}} \forall t_0 \in T \exists t_1 \in T.(t_0, t_1) \in R,$$

where R is an arbitrary relation on traces is not necessarily a holistic hyperproperty. The reason is that R may not be definable as a coinductive (or inductive) predicate. However, we claim that holistic hyperproperties capture security-relevant hyperproperties (e.g., known variants of noninterference, information flow and secrecy, requiring uncertainty about the real state of the system), which are the most well-studied and important class of hyperproperties. The claim that holistic and respectively incremental hyperproperties capture a large class of interesting, security-relevant policies will be reaffirmed in Chapter 4.

After Clarkson and Schneider proposed a methodology for verification of k-safety hyperproperties based on invariance arguments [18, 19] (generalizing Terauchi and Aiken's work on 2-safety [84]), this is a second attempt to develop a generic verification methodology for a large class of hyperproperties. As already mentioned, our approach is based on transitioning from a trace-based holistic view of systems and hyperproperties to an incremental perspective of systems as G-coalgebras and hyperproperties as coinductive predicates. Incremental hyperproperty definitions are a generalization of bisimulation relations on the state space of systems and verification amounts to checking whether such a relation exists. In the next chapter we demonstrate that our methodology (via H'-simulations) goes beyond the state-of-the-art in being able to also handle a relatively large class of liveness hyperproperties, such as a number of variants of possibilistic information flow security definitions. The problem of determining precisely which class of hyperproperties is *incrementalizable*, and thus directly benefits from the methodology, is open. In addition, we would like to explore the possibility of abstracting even further and potentially coming up with an incrementalization technique and a notion of H'-simulation working for all of (but not limited to) the classes presented here. Finally, we plan to explore the difficulty in rearranging quantifiers, which was identified as a main problem for incrementalization.

Although the proposed methodology is generic and useful, it has some limitations. It cannot tackle arbitrary hyperproperties, such as *mean response time* and *percentage uptime* [18], which perform some operation over all traces of a system (such as taking the mean). Although one might think that these definitions would be excluded from consideration simply because the abstract representation of systems as sets of traces excludes time, time can be included by having a richer alphabet, e.g. each element being a pair of an action and a real/natural number. However, working with such hyperproperties is difficult, regardless of the methodology (holistic or incremental), even if we rule out potential divergence due to computing with infinite traces. One possible idea to dealing with such policies is via metric coinduction [44].

3.6 Related Work

Clarkson and Schneider show that labelled transition systems can be encoded as sets of traces [19]. They argue that bisimulation-based hyperproperties, notably Focardi and Gorrieri's *bisimulation nondeducibility on composition* (*BNDC*), can be converted into trace sets. This is in effect the opposite to what we suggest in this chapter. We propose to go from trace sets to trees and state-based systems because the latter are well-understood and enjoy well-established verification techniques, as well as mature verification tools (see Chapter 6 for an illustration of the use of off-the-shelf tools). It is arguable, but we believe that such an approach is more natural and generic enough to work for a large number of applications.

That incrementalization is useful can be seen from recent work on noninterference for reactive systems [11]; Bohannon et al. start with a holistic definition of reactive noninterference and convert it into a relation on program states that they call *ID-bisimulation*; they effectively make the definition incremental. The authors use that latter incremental definition in order to prove that well-typed RIMP programs are secure. They also show that an *ID-bisimulation* implies the high level, holistic policy.

It should be noted that H'-simulations are similar to coinductive predicates proposed by Niqui and Rutten [66] in a categorical setting. They essentially show that coinductive predicates can be seen as final coalgebras in a a category of relations. As concrete examples, they consider a number of coinductive predicates on streams, however they present examples that are not related to security. Niqui and Rutten's work gives a different, categorical perspective on coinductive predicates. However, our H'-simulations are coinductive predicates on systems/trees and are presented using Knaster-Tarski semantics, which we find more intuitive. Moreover, in later chapters we generalize the definition of H'-simulations to include (a restricted variant of) mixed coinductive/inductive predicates, which are more complex than what Niqui and Rutten present.

At first sight, incrementalization is somewhat similar to *unwinding* [32]. As Goguen and Meseguer describe it, unwinding is the process of translating a security policy first into local constraints on the transition system that inductively guarantee that the policy is satisfied and second in a finite set of lemmas such that any system that satisfies the lemmas is guaranteed to satisfy the policy. The main difference to our work is that unwinding is still a trace based property, whereas incrementalization results in coinductive predicates and reasoning on trees. Moreover, our H'-simulations are coinductively defined relations, which allow reasoning about infinite traces and hence hyperproperties. Finally, we present a framework for reasoning about hyperproperties and not a methodology for a concrete security-relevant hyperproperty.

It turns out that incremental hyperproperties are inherently related to Mantel's work

on unwinding of possibilistic security properties [53]. He proposes a modular framework in which most well-known security properties can be composed from a set of *basic security predicates* (BSPs); he also presents unwinding conditions for most BSPs. His unwinding conditions are specified locally on states of the system (inspired by Rushby's work [72]) as opposed to the more traditional global (trace-based) unwinding conditions. These unwinding conditions can be seen as simulation relations on system states and in that sense are similar to our incremental security hyperproperties. A major difference is that Mantel reasons about traces that are only finite, i.e. his systems are in 2^{A^*}. This implies that his framework cannot express liveness hyperproperties, whereas our framework can. Moreover, our framework is agnostic to the hyperproperty/hyperliveness classification. Interestingly, Mantel's unwinding conditions can be seen as instances of incremental hyperproperties on finite systems. This is further explored in Chapter 4.

Recent work [23] has proposed an automata-theoretic technique for model checking the possibilistic information flow hyperproperties from Mantel's framework [53] on finite state systems. The authors show how to model check Mantel's BSPs, which are the building blocks of the respective holistic hyperproperties. This is a nice theoretical result, supporting our thesis that incremental hyperproperties are amenable to model checking. On the negative side, the authors show that the model checking problem is undecidable for the class of pushdown systems. The proposed model checking approach is based on deciding set inclusion on regular languages. The latter question can be answered by standard automata-theoretic techniques. This approach is not directly applicable to hyperproperties, because the presence of infinite traces means that the languages (sets of traces) under consideration are not regular. The problem of model-checking incremental hyperproperties (via games) is explored in Chapter 5.

There has been a substantial amount of work on verifying other specific hyper-properties, most notably of secure information flow from both the language-based security [75] and process calculi security [26, 74] communities. Language-based secure information flow has traditionally relied on information flow type systems, with a recent trend to incorporate program logics or a combination of both [75, 9, 20, 36]. There have also been attempts to address noninterference using results from process algebra [74, 26]. Common for this line of work is formalizing different definitions of security and showing that they all depend on some notion of equivalence of processes, e.g. strong, weak, power bisimulation. The difference with our work is that we propose a framework for reasoning about hyperproperties and not a concrete methodology for verification of a concrete security policy.

3.7 Summary

This chapter presents a formal classification of hyperproperties into holistic and incremental ones. It furthermore motivates the shift from a holistic to an incremental approach to hyperproperty specifications, dictated by the fact that the incremental approach has a clearer verification methodology. We argue that identifying the class of hyperproperties that are incrementalizable and finding a generic methodology for incrementalization of holistic hyperproperties are important problems. We propose a generic framework and techniques to explore the process of incrementalization and the usefulness of the resulting incremental hyperproperties. We identify three classes of incrementalizable hyperproperties. We introduce the notion of H'-simulation relations and suggest a verification methodology based on these.

The following chapter explores the connection between incremental hyperproperties and unwinding relations. Further on, Chapters 5 and 6 focus on the verification of incremental hyperproperties via games.

Chapter 4

Coinductive Unwinding of Security-Relevant Hyperproperties

In this chapter we explore the connection between incremental hyperproperties and the most closely-related work, namely on the verification of security-relevant policies via unwinding. We demonstrate that combining ideas from both approaches is beneficial for the verification of hyperproperties.

Unwinding is a well-known technique used to prove that systems are secure with respect to a variety of noninterference policies. The main benefit of unwinding is that it reduces the problem of high-level policy verification to searching for an unwinding relation on the state-space of the system of interest. The idea is intuitively appealing because the connection between the transitions of the system and the higher level policy, often expressed as difficult to check relations on execution traces, is instead given by an *unwinding theorem*: the existence of a particular unwinding relation implies/is equivalent to the high-level policy. It should be noted that an unwinding relation in this sense is still a trace-based relation.

Mantel works with a different notion of unwinding in his MAKS framework [55]. This framework uses unwinding relations that have several advantages in comparison with the typical ones. The advantages are that these unwinding relations can be arbitrary (i.e. not necessarily symmetric, transitive, etc.) and they are also completely local (and thus not trace-based). These advantages are clearly shared with our H'-simulation relations from Chapter 3. In addition, the MAKS framework is modular, a feature that is also beneficial when reasoning about hyperproperties. Hence, the

following natural question arises: is it possible to use the MAKS framework directly or indirectly (and possibly in combination with the theory from Chapter 3) for the purpose of hyperproperty verification?

In this chapter we answer the former question. We show that directly using Mantel's framework for the verification of security-relevant hyperproperties is problematic, as the framework is geared towards reasoning about terminating-only behaviors (finite systems). However, hyperproperties are defined as sets of sets of (potentially) infinite traces (see Definition 2.4.3). Intuitively, infinite traces are needed to reason about liveness properties (and hyperproperties), as the good thing is "always possible" and "possibly infinite" [18, 19]; this is in contrast to the somewhat simpler verification of safety properties (and hyperproperties), as if a counter-example exists, it can be finitely determined (e.g. by computing least fixed points [35]). To enable reasoning about infinite behavior, we propose a coinductive reinterpretation of security-relevant hyperproperties, Basic Security Predicates and unwinding relations. The reason to use coinduction is that it has emerged as the technique of choice for reasoning about infinite data types and automata with infinite behavior [77]. Another strength of coinductive reasoning is that it is suitable for automation, allowing the potential reuse of enhancements, such as *up to techniques* [69]. Such techniques strive to keep the candidate relations, implying the needed high-level policy, as small as possible and thus reduce the amount of work for verification. Exploring these techniques for H'-simulations is left for future work.

4.1 Synopsis

We start by sharing some of the motivation for this work. To deal with security-relevant hyperproperties in general, we need to be able to reason about infinite systems that cannot be approximated by finite ones. Intuitively, these are systems that have, at least for some infinite traces, relevant behavior in the whole infinite trace. In contrast, there are infinite systems that can be approximated by finite ones, i.e. for each execution trace, the interesting behavior stops at some point, followed by trivial behavior (e.g. low only events or internal events). Moreover, for some policies (e.g. liveness (hyper)properties) it is not sufficient to consider finite only traces, as each finite trace needs to have a possible extension to an infinite trace, satisfying a certain condition. It is not possible to directly use the MAKS framework or typical unwinding conditions for such policies and systems (potential problems with non-well-foundedness). We thus propose to reason about infinite systems (that cannot be approximated by finite ones) by the use of coinductive variants of the definitions, which are given in the new mathematical development presented in the rest of this chapter.

The new development starts with a coinductive reinterpretation of a number of security-relevant hyperproperties (essentially by replacing the trace predicates with coinductive or coinductive/inductive ones) and their corresponding BSPs and unwinding relations. The security-relevant hyperproperties are defined based on coinductive predicates, due to the fact that, as discussed above, systems may be possibly infinite in a non-trivial way (on finite systems the hyperproperties in question have the same semantics as the original ones). This results in coinductive definitions of the BSPs and the respective unwinding relations, allowing reasoning about elements of non-well-founded sets [2]. Whereas the typical, inductive interpretation gives semantics of policies on terminating systems exclusively, the coinductive interpretation gives semantics of security policies on both finite and infinite systems. We should note that the coinductively defined BSPs have in some cases different semantics on finite systems compared to the ones presented by Mantel [55]. This is not problematic, as BSPs are typically used as a pure means to reasoning about the high level policies. What is important is that the high level policies presented in the work of Mantel and the respective coinductive interpretations that we propose have the same semantics on finite systems.

Further, we show that the basic structure of the MAKS framework is still intact, although the definitions have changed. This is indicated by the fact that the respective variants of a number of the unwinding theorems, unwinding-conditions-to-BSPs theorems and BSPs-to-holistic-hyperproperties theorems hold as expected. We show that the new unwinding relations are instances of H'-simulations. Hence, we have demonstrated that the class of incremental hyperproperties is substantial. Moreover, we can reuse the verification techniques introduced in Chapter 3 (via H'-simulations) for verification of security-relevant hyperproperties from the class of possibilistic information flow policies [55] which are instances of liveness hyperproperties. This means that our verification method is novel as it works for a class of liveness hyperproperties. Whether it works for all liveness hyperproperties is an interesting question for future work.

The chapter is based on an extended version of a recent paper of ours [62]. The relevant proofs can be found in Appendix B.

4.2 Motivation

The majority of security-relevant policies (for instance, all the ones presented in Mantel's thesis [55]) reason about finite only traces. As a result such definitions are termination-insensitive: all diverging computations are considered to be "the same" and are thus ignored. A typical argument for this is that systems satisfying such definitions are secure most of the time, i.e. as long as they terminate, which is their

expected behavior. The possibility of leaks through the termination channel, which is a type of covert channel [47] [1], is essentially ignored. Although such definitions might suffice for some types of programs, e.g. for some batch-job programs, they are clearly not satisfactory when reasoning about systems supposed to be infinite by specification. Examples are reactive systems, such as servers, embedded systems, operating systems, etc.

For instance, consider the simple system $S_1 = \{(hl)^\omega, lll\}$, where $A = L \cup H$, $L = \{l\}$ is the set of *low* confidentiality events and $H = \{h\}$ is the set of *high* confidentiality events. Clearly, termination insensitive definitions, such as NF_o introduced in Section 4.3.1, distinguish this system as trivially secure: as only one of the traces is in A^* and it has low events only. However, the system is intuitively not secure as termination is low-observable, i.e. the low observer can distinguish the two traces. In fact, an attacker having control over a similar system can in theory leak any secret to the low-level user by sending bits via the termination channel. In practice, this is particularly dangerous when the attacker can run the program multiple times, as every run can transmit one bit.

As another motivating example, consider systems with nonterminating behavior such that confidential events in all traces occur infinitely often (this is a liveness property, similar to $P_{\Box\Diamond}$ from Section 3.3.2). In such situations it is impossible to declare a system secure by examining only finite prefixes. A typical policy, for instance given by Mantel's deletion of events (based on deleting the last confidential event in any trace and requiring a particular modification of the resulting trace to be in the candidate system, see Section 4.3.2 for the formal definition) or a naive coinductive interpretation of the latter, such as DN_V from Section 4.3.2, would not be able to properly reason about such systems. Such policies would accept any system having only infinite behavior as being trivially secure. Clearly, this is not desirable.

As systems exhibiting infinite behaviors which have no last confidential event, are abundant in practice, they are important for both specification and verification. Since hyperproperties are generic system specifications, it is natural that they should be able to express and reason about such behavior. To achieve this, we give systems and hyperproperties a coinductive semantics and use coinduction as a reasoning tool for such systems. In general, theoretical machinery is needed in order to reason about potentially infinite behaviors allowed by system specifications. Such machinery, at least for reasoning about security-relevant hyperproperties in general, is currently lacking. Providing such machinery is the main motivation of this chapter.

[1] A *covert channel* is one that is not meant to be used for transmitting information at all.

4.3 Coinductive Interpretation of Security-Relevant Hyperproperties

The coinductive interpretation of well-known, holistic, security-relevant hyperproperties requires coinductively defined predicates on traces and sometimes functions (often treated as relations). We start off by giving the needed definitions. It should be noted that the holistic hyperproperties presented here are defined based on coinductive predicates and not on functions.

Definitions. The definition of no_Z is repeated here for convenience; all others are new except ev_z, which is redefined. Coinductively define *bisimulation* $\sim\colon A^\infty \times A^\infty \to 2$ as follows:

$$\frac{}{\varepsilon \sim \varepsilon}\ {\scriptstyle coind} \qquad \frac{x \sim y}{a \cdot x \sim a \cdot y}\ {\scriptstyle coind}$$

Coinductively define predicate $no_Z : A^\infty \to 2$ (parameterized by set Z), which states that there are no events from set Z in a trace:

$$\frac{}{no_Z(\varepsilon)}\ {\scriptstyle coind} \qquad \frac{a \in A \setminus Z \qquad no_Z(x)}{no_Z(a \cdot x)}\ {\scriptstyle coind}$$

Note that $no_Z(t)$ is the coinductive version of predicate $(t|_Z = \varepsilon)$, where notation $t|_Z$ is from the MAKS framework [55] and denotes the Z-projection of a trace t.

Next, inductively define $w \rightsquigarrow_Z a \cdot w'$ (w Z-reveals a with tail w'):

$$\frac{}{\varepsilon \rightsquigarrow_Z \varepsilon} \qquad \frac{a \in Z}{a \cdot w \rightsquigarrow_Z a \cdot w} \qquad \frac{b \in A \setminus Z \qquad w \rightsquigarrow_Z a \cdot w'}{b \cdot w \rightsquigarrow_Z a \cdot w'}$$

We need a coinductive relation ev_Z, relating any trace to its projection onto Z. Technically, the relation is a partial function that filters out events from set Z, and it will be defined and used as one, mainly to keep the connection to $t|_Z$, as $ev_Z(t)$ is the coinductive version of $t|_Z$.

$$\frac{}{ev_Z(\varepsilon) = \varepsilon}\ {\scriptstyle coind} \qquad \frac{w \rightsquigarrow_Z a \cdot w' \qquad ev_Z(w') = u}{ev_Z(w) = a \cdot u}\ {\scriptstyle coind}$$

Note that ev_Z can be seen as a relation: $(w,u) \in ev_Z$ iff $ev_Z(w) = u$. This definition is an improved version of the same definition from Section 2.4. The first difference is that the alphabet for the traces considered here is A instead of $A \cup \{\tau\}$. The

second difference is that, unlike the purely coinductive definition from Section 2.4, this version is productive [10] when the function is defined. The reason (and difference) is that this definition of function ev_Z only computes by calling its inductive component $\rightsquigarrow_Z$ and thus it satisfies a form of *eventually* predicate (eventually an $a \in Z$ is found as a head of the resultant stream). After finding the head, computation goes on with a recursive call on data computed in the inductive part. In essence this recursive call guarantees the *infinitely often* predicate. The idea of such eventually and infinitely often predicates comes from the literature: these were used to show how to formalize a filter function in Coq [10].

Finally, coinductively define *weak bisimulation* (parameterized by set Z) $\approx_Z : A^\infty \times A^\infty \to 2$ as follows:

$$\frac{}{\varepsilon \approx_Z \varepsilon}\ {\scriptstyle coind} \qquad \frac{w \rightsquigarrow_Z a \cdot w' \qquad u \rightsquigarrow_Z a \cdot u' \qquad w' \approx_Z u'}{w \approx_Z u}\ {\scriptstyle coind}$$

Although this definition is coinductive, it is again based on an inductive definition of $\rightsquigarrow_Z$. This is important as it helps avoiding the potential fairness problems: for instance, the definition of weak trace equivalence from Section 2.4 has the problem that τ^ω is equivalent to any trace. That is why similar definitions have been called equivalence up to stuttering and prefix [87] (i.e. it relates two traces if one is a prefix of the other) and it results in termination-insensitive definitions. The problem is that there are separate rules for both related states. Intuitively, this means that one state may get arbitrarily ahead of the other. Further details are available in Section 2.4.

In contrast, the definition of weak bisimulation above does not have a fairness problem and it is termination-sensitive. There is no fairness problem, since both related states may only proceed together, where their "moves" are given by the inductive definition of $\rightsquigarrow_Z$, or alternatively both related states are accepting, i.e. both related traces are finite.

Finally, the extension of predicate *test* to the obvious inductive definition $test^* : \mathsf{Sys} \to (A^* \to 2)$ on words is given as follows. For $a \in A$, $w \in A^*$ and ε the empty trace, $test^*$ is defined as:

$$\frac{o(X)}{test^*{}_\varepsilon(X)} \qquad \frac{test_a(X) \qquad test^*{}_w(X_a)}{test^*{}_{a\cdot w}(X)}$$

For tree T and word w, the predicate $test^*{}_w(T)$ is true whenever the finite sequence w can be performed, starting at the root of the tree T.

4.3.1 Coinductive View on some Well-known, Holistic, Security-relevant Hyperproperties

In this section we give coinductive semantics to some of the well-known, holistic, security-relevant hyperproperty definitions. The definitions presented next are known from the literature, however, for each original definition we present its respective variant on potentially infinite systems (i.e. as a hyperproperty). The difference between each pair of definitions lies in the core definitions of the ingredients: in the new versions, these definitions are coinductive. This is needed to lift the original definitions to potentially infinite systems and thus convert them into hyperproperties. The old definitions have subscript o, whereas the new, coinductive definitions are presented in a box. It is noteworthy that the new and old definitions coincide on finite systems.

Noninference

This policy will be called NF_o (the original version of NF) and it is originally defined on finite systems as follows [86]:

$$NF_o(X) \mathrel{\widehat{=}} \forall x \in X.\, x|_L \in X.$$

The definition says that any candidate system T is secure with respect to noninference iff the projection of each trace t in T to low-confidentiality security events is a possible trace of the candidate system T.

The coinductive variant of *noninference* NF is given using the coinductive analogue of $x|_L$ called ev_L:

$$\boxed{NF(X) \mathrel{\widehat{=}} \forall x \in X.\; ev_L(x) \in X.}$$

Generalized Noninference

This policy was originally proposed by Zakinthinos and Lee [86] and it was given as follows:

$$GNF_o(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \exists x_1 \in X.\, (x_1|_{HI} = \varepsilon \wedge\ x_1|_L = x_0|_L).$$

The definition says that any candidate system T is secure with respect to generalized noninference iff the following holds: for every possible trace t_0 in T, there must be a trace t_1 in T such that t_1 has no high inputs and the projections of t_0 and t_1 to low-confidentiality security events are equal.

The coinductive interpretation of *generalized noninference GNF* is:

$$GNF(X) \mathrel{\widehat{=}} \forall x_0 \in X\ \exists x_1 \in X.\ (no_{HI}(x_1) \wedge x_1 \approx_L x_0).$$

Generalized Noninterference

This policy will be called GNI_o. It was originally defined as follows [58]:

$$GNI_o(X) \mathrel{\widehat{=}} \forall x_1, x_2, x_3 \in A^*\ .\ [(x_1 \cdot x_2 \in X \wedge x_3|_{A \setminus HI} = x_2|_{A \setminus HI}) \rightarrow$$
$$\exists x_4 \in A^*.\ (x_1 \cdot x_4 \in X \wedge x_4|_{L \cup HI} = x_3|_{L \cup HI})].$$

The definition says that any candidate system T is secure with respect to generalized noninterference iff the following holds: for every trace $t_1 \cdot t_2$ in T and every possible trace $t_1 \cdot t_3$ that differs from $t_1 \cdot t_2$ only in high level input events, there exists a trace $t_1 \cdot t_4$ in T that differs from $t_1 \cdot t_3$ only in high-level output and high-level internal events.

Using coinductive relations on traces, we define *generalized noninterference GNI* as a hyperproperty:

$$GNI(X) \mathrel{\widehat{=}} \forall x_1 \in A^*\ \forall x_2, x_3 \in A^\infty\ .\ [(x_1 \cdot x_2 \in X \wedge x_3 \approx_{A \setminus HI} x_2) \rightarrow$$
$$\exists x_4 \in A^\infty.\ (x_1 \cdot x_4 \in X \wedge x_4 \approx_{L \cup HI} x_3)].$$

Note that the equality of projections is replaced by the coinductively defined relation $\approx_Z$ (parameterized by $L \cup HI$).

Perfect Security Property

The policy presented next was originally proposed by Zakinthinos and Lee [86]:

$$PSP_o(X) \mathrel{\widehat{=}} (\forall x \in X.x|_L \in X) \wedge (\forall \alpha \in A^* \forall \beta \in A^*.\ [(\beta \cdot \alpha \in X \wedge \alpha|_H = \varepsilon) \rightarrow$$
$$\forall h \in H.\ (\beta \cdot h \in X \rightarrow \beta \cdot h \cdot \alpha \in X)]).$$

The definition says that any candidate system T is secure with respect to the perfect security property iff the following conditions hold: for every possible trace t in T, its

projection to low-confidentiality security events must be a possible trace in T. Further, any finite trace $t \in A^*$ that can be formed from a possible trace in T by adding a high-level event h, not followed by any other high-level event in t, should also be a possible trace of T.

Finally, our coinductive interpretation of the *perfect security property PSP* is:

$$PSP(X) \mathrel{\widehat{=}} (\forall x \in X.\, ev_L(x) \in X) \wedge (\forall \alpha \in A^{\infty} \forall \beta \in A^*. \\ [(\beta \cdot \alpha \in X \wedge no_H(\alpha)) \rightarrow \forall h \in H.\, (\beta \cdot h \in X \rightarrow \beta \cdot h \cdot \alpha \in X)]).$$

We have presented only a few information flow definitions in order to illustrate what is needed to represent them as hyperproperties. It is relatively straightforward to convert any of the ones not presented here. Nevertheless, the examples are enough to cover a number of important BSPs and unwinding relations, as well as to raise some interesting questions, which will be considered in the following sections.

The examples suggest a possible technique to adapt Mantel's unwinding relations to reason about security-relevant hyperproperties and a connection with incremental hyperproperties, namely that an H'-simulation (see Chapter 3), implying that an incremental hyperproperty H' holds, is a(n) (conjunction of) unwinding relation(s). This is further explored in Section 4.4.

4.3.2 Coinductive View on BSPs

Mantel introduced the MAKS framework [55, 52], which can represent most well-known possibilistic security policies as conjunctions of Basic Security Predicates. Mantel classified his BSPs in two dimensions. In the *first dimension* fall BSPs that essentially hide the *occurrence* of confidential events (A_c-events), whereas in the *second dimension* are BSPs that hide the *non-occurrence* of A_c-events. This means roughly the following: in the first dimension, whenever a confidential event occurs, it should be also possible that it did not occur, i.e. a respective trace without the event should exist. In the second dimension, whenever a confidential event did not occur, there should be an alternative trace in which it did occur.

Mantel's BSPs and security policies are defined on finite traces only. In this section we present a coinductive perspective on a number of the BSPs, parameterized by a *security view* (a partition of the alphabet A, as introduced in Section 2.5). Whereas the coinductively-defined security policies from Section 4.3.1 work on both finite and infinite systems and on finite systems have the same semantics as the originals, the situation is fundamentally different with the BSPs. The BSPs presented here

(except the first one, removal of events R_V) have different semantics on finite systems compared with the Mantel's BSPs. The reason is that lifting Mantel's BSPs to infinite systems is not simply a matter of coinductive interpretation of the definitions. More precisely, often it does not make sense to refer to the last confidential event in an infinite trace, as there is no such event, at least by specification; this will be further clarified by Example 4.3.1. Instead, we need to change the rules but still try to keep the structure of the framework. Although we have changed the definitions, we have kept their original names in order to illustrate that the structure of the framework (see Figure 2.1) is preserved. The original definitions are again presented for comparison. The old definitions have subscript o, whereas the new, coinductive definitions are presented in a box.

Note, and this applies to all definitions presented in this section, that these are closure operators. They take some set T and say what other traces should be in T. Although it is important what the application of the operator does for every particular trace, we are also interested in what the operator collectively does for all traces in the set T.

We start with some BSPs from the first dimension.

Removal of Events R_V

Predicate $R_V(T)$ requires for any trace $\sigma \in T$ the existence of another trace γ which has no A_c-events and which has the same A_v-events (essentially allowing "corrections" of A_n-events). The original definition [55] is:

$$R_V{}^o(T) \mathrel{\widehat{=}} \forall \sigma \in T \; \exists \gamma \in T \; . \; (\gamma|_{A_c} = \varepsilon \wedge \sigma|_{A_v} = \gamma|_{A_v}).$$

Our definition is:

$$\boxed{R_V(T) \mathrel{\widehat{=}} \forall \sigma \in T \; \exists \gamma \in T \; . \; (no_{A_c}(\gamma) \wedge \sigma \approx_{A_v} \gamma).}$$

Note that we have replaced the relations on traces in the original work with coinductive ones, similarly to the modifications of the definitions from Section 4.3.1. Interestingly, such a straightforward modification will not be possible for the rest of the BSP definitions we explore.

Stepwise Deletion of Events D_V

The original definition [55] changes any trace σ in a candidate set T by deleting the *last* occurrence of a confidential event and requires that the resulting trace can be

corrected (by possibly inserting/deleting events from A_n if it is not empty) resulting in a possible trace γ in T.

$$D_V{}^o(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot c \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon) \rightarrow$$
$$\exists \alpha', \beta' \in A^* \, . \, (\beta' \cdot \alpha' \in T \wedge \alpha'|_{A_v} = \alpha|_{A_v} \wedge \alpha'|_{A_c} = \varepsilon \wedge \beta'|_{A_v \cup A_c} = \beta|_{A_v \cup A_c})].$$

If we *naively* convert Mantel's definition to potentially infinite traces, we get the following:

$$DN_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot c \cdot \alpha \in T \wedge no_{A_c}(\alpha)) \rightarrow$$
$$\exists \alpha' \in A^\infty, \beta' \in A^* \, . \, (\beta' \cdot \alpha' \in T \wedge \alpha' \approx_{A_v} \alpha \wedge no_{A_c}(\alpha') \wedge \beta' \approx_{A_v \cup A_c} \beta)].$$

This definition would work as expected on finite traces. Unfortunately, it is not well-suited for infinite traces. To illustrate this, consider the following example:

Example 4.3.1. *Let $V = (A_v, A_n, A_c)$ be a view such that $A_v = \{l_1, l_2\}$, $A_n = \emptyset$ and $A_c = \{h_1, h_2\}$. Consider system $S_1 = \{(l_1 h_1 h_2 l_2)^\omega\}$. Intuitively, system S_1 is not secure, as every time l_2 is observed it is clear that h_1 and h_2 must have occurred. Unfortunately, the definition of DN_V does not capture this intuition, as system S_1 is trivially secure with respect to the definition. The reason for this problem is that confidential events appear infinitely often, thus there is no suffix t for which $no_{A_c}(t)$ holds.*

Potentially infinite traces are allowed in many useful systems (operating systems, servers, embedded systems etc.) and often there is no last confidential event, as confidential events occur infinitely often. To further demonstrate that such systems and policies exist, recall that one of the incrementalizable classes from Chapter 3, namely SHH, deals with such systems. Thus the definition of DN_V needs to be changed. We propose the following definition to fix the problem:

$$D_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [\beta \cdot c \cdot \alpha \in T \rightarrow$$
$$\exists \alpha' \in A^\infty, \beta' \in A^* . \, (\beta' \cdot \alpha' \in T \wedge \beta' \approx_{A_v \cup A_c} \beta \wedge \alpha' \approx_{A_v \cup A_c} \alpha)].$$

This definition deletes *some* occurrence of a confidential event in a trace and then perturbs the resulting trace. Unfortunately, as already mentioned, on finite traces the definition is not semantically equivalent to the original one. To see this consider the following:

Example 4.3.2. *Let $V = (A_v, A_n, A_c)$ be a view such that $A_v = \{l_1, l_2\}$, $A_n = \emptyset$ and $A_c = \{h_1, h_2\}$. Consider system $S_2 = \{l_1h_1h_2l_2, l_1h_1l_2, l_1l_2\}$. It is easy to check that $D_V(S_2)$ does not hold — as $l_1h_2l_2$ has to be in S_2, but it is not. Nevertheless S_2 is secure with respect to Mantel's original definition $D_V{}^o$, as well as with respect to our naive definition DN_V.*

It should be noted that the definition D_V, proposed here and used throughout the chapter, is stronger — it requires more possible traces and hence higher uncertainty for the attacker than DN_V. In other words, $D_V(X) \rightarrow DN_V(X)$. Moreover, D_V properly rejects systems exhibiting the pattern of S_1 as insecure; to see one reason why, note that $l_1h_2l_2(l_1h_1h_2l_2)^\omega \notin S_1$.

More importantly, the BSP definitions presented in this chapter are not simply Mantel's BSPs lifted to potentially infinite traces (we have shown that doing this would not be meaningful). The BSP definitions presented here may have different semantics (compared to Mantel's) on finite systems, in fact they are more restrictive and guarantee higher uncertainty. This is not problematic as they still fulfill their purpose: they allow us to reason about the high level policies (i.e. holistic hyperproperties) in a very convenient way. Moreover, the holistic hyperproperties presented in this chapter, if restricted to finite systems, have the same semantics as the respective policies presented by Mantel. The proof of this statement is straightforward and based on the following observation. The coinductive definitions of all the relations on traces used to build the holistic hyperproperties if instantiated on finite only traces are equivalent to Mantel's definitions of trace predicates. For instance, $no_{HI}(t_1)$ is equivalent to $t_1|_{HI} = \varepsilon$ and $t_1|_L = t_0|_L$ is equivalent to $t_1 \approx_L t_0$ for all $t_0, t_1 \in A^*$.

Backwards Strict Deletion *BSD*

The next BSP is called *BSD*. The intuitive idea is that the occurrence of an A_c-event should not be deducible. *BSD* is very similar to D_V. The only difference to D_V is that the part of the trace that has already occurred (β) cannot be changed, i.e. the "past" of the trace cannot be changed. The original definition [55] is:

$$BSD_V^o(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^*\ \forall c \in A_c\ .\ [\beta \cdot c \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon \rightarrow$$
$$\exists \alpha' \in A^*.\ (\beta \cdot \alpha' \in T \wedge \alpha'|_{A_v} = \alpha|_{A_v} \wedge \alpha'|_{A_c} = \varepsilon)].$$

Our definition of $BSD_V(T)$ is given next:

$$BSD_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [\beta \cdot c \cdot \alpha \in T \rightarrow \exists \alpha' \in A^\infty . \, (\beta \cdot \alpha' \in T \wedge \alpha' \approx_{A_v \cup A_c} \alpha]$$

It should be noted that a similar modification as to D_V was used here and the reason again is to enable tackling systems which do not have a last confidential event.

Although the BSP definitions have changed, the following theorem reestablishes a connection between the BSPs, known from Mantel's framework [55].

Theorem 4.3.3. *Let $V = (A_v, A_n, A_c)$ be a view and T be a set of traces. Then the following implications hold: $BSD_V(T) \rightarrow D_V(T)$ and $D_V(T) \rightarrow R_V(T)$.*

This theorem gives convenience when reasoning about the high-level policies. For instance, it is enough to prove $BSD_V(T)$, then $D_V(T)$ and $R_V(T)$ hold "for free".

Strict Deletion SD_V

Mantel's original definition of *strict deletion* [55] is given first:

$$SD_V^o(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot c \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon) \rightarrow \beta \cdot \alpha \in T].$$

This BSP is similar to BSD_V presented above, but it is even stricter. It claims that whenever we look at the last confidential event c of a trace, it should be possible to delete c, and the rest of the trace ($\beta \cdot \alpha$) should be a possible trace of T. In other words, the "past" of the trace and its "future" should remain unchanged when deleting c and then the rest of the trace should be a possible trace of T.

Our version of the BSP is again different than Mantel's: as in the previous definition, it does not search for the last A_c-event. Our coinductive version of $SD_V(T)$ is given next:

$$SD_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [\beta \cdot c \cdot \alpha \in T \rightarrow \beta \cdot \alpha \in T].$$

The rest of the presented BSPs are from the second dimension, hiding the *non-occurrence* of A_c-events.

Backwards Strict Insertion BSI_V

Mantel's original definition [55] is given first:

$$BSI_V^o(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon) \rightarrow \exists \alpha' \in A^* . \, (\beta \cdot c \cdot \alpha' \in T \wedge \alpha'|_{A_v} = \alpha|_{A_v} \wedge \alpha|_{A_c} = \varepsilon)].$$

This BSP is in a sense dual to BSD_V— instead of deleting an A_c-event, it requires the possible insertion of such an event before any possible finite suffix α having no confidential events. Of course, we have again modified the definition to a coinductive one and it does not search for the last A_c-event, hence it works on infinite traces. The same also holds for all the following definitions. Our coinductive version of $BSI_V(T)$ is given as follows:

$$BSI_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [\beta \cdot \alpha \in T \rightarrow \exists \alpha' \in A^\infty . \, (\beta \cdot c \cdot \alpha' \in T \wedge \alpha' \approx_{A_v \cup A_c} \alpha)].$$

Backwards Strict Insertion of Admissible Events $BSIA_V^{\rho o}$

Mantel's original definition [55] is given first:

$$BSIA_V^{\rho o}(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon \wedge Adm_V^\rho(T, \beta, c)) \rightarrow \exists \alpha' \in A^* . \, (\beta \cdot c \cdot \alpha' \in T \wedge \alpha'|_{A_v} = \alpha|_{A_v} \wedge \alpha'|_{A_c} = \varepsilon)].$$

This BSP is similar to BSI_V, but it hides the non-occurrence of *admissible* events only. This is useful, as hiding the occurrence of all possible confidential events A_c might be too extreme in practice. Instead, we might need to hide the occurrence of only some subset of A_c. Our coinductive version of $BSIA_V^\rho(T)$ is given as follows:

$$BSIA_V^\rho(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [(\beta \cdot \alpha \in T \wedge Adm_V^\rho(T, \beta, c)) \rightarrow \exists \alpha' \in A^\infty . \, (\beta \cdot c \cdot \alpha' \in T \wedge \alpha' \approx_{A_v \cup A_c} \alpha)].$$

Strict Insertion SI_V

Mantel's original definition [55] is given first:

$$SI_V^o(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \; \forall c \in A_c \; . \; [(\beta \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon) \rightarrow \beta \cdot c \cdot \alpha \in T].$$

This BSP requires the possibility to insert any A_c-event at any place in a stream, it is strict because neither the past nor the future part of the trace may be changed by perturbing A_n-events. Our coinductive version of $SI_V(T)$ is given as follows:

$$SI_V(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \; \forall \beta \in A^* \; \forall c \in A_c \; . \; [\beta \cdot \alpha \in T \rightarrow \beta \cdot c \cdot \alpha \in T].$$

Strict Insertion of ρ-admissible Events SIA_V^ρ

Mantel's original definition [55] is given first:

$$SIA_V^{\rho o}(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \; \forall c \in A_c \; .$$
$$[(\beta \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon \wedge Adm_V^\rho(T, \beta, c)) \rightarrow \beta \cdot c \cdot \alpha \in T].$$

This BSP requires the possibility to insert any ρ-admissible A_c-event at any place (where admissible) in a trace such that it would be the last confidential event in the trace. Our coinductive version of $SIA_V^\rho(T)$ is given as follows:

$$SIA_V^\rho(T) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \; \forall \beta \in A^* \; \forall c \in A_c \; .$$
$$[(\beta \cdot \alpha \in T \wedge Adm_V^\rho(T, \beta, c)) \rightarrow \beta \cdot c \cdot \alpha \in T].$$

Note again that the definition is lifted to potentially infinite systems. More importantly, we drop the restriction of necessarily dealing with the last A_c-event.

Insertion of ρ-admissible Events IA_V^{ρ}

Mantel's original definition [55] is given first:

$$IA_V^{\rho o}(T) \mathrel{\widehat{=}} \forall \alpha, \beta \in A^* \, \forall c \in A_c.$$
$$[(\beta \cdot \alpha \in T \wedge \alpha|_{A_c} = \varepsilon \wedge Adm_V^{\rho}(T, \beta, c)) \rightarrow$$
$$\exists \alpha', \beta' \in A^*. \, (\beta' \cdot c \cdot \alpha' \in T \wedge \alpha'|_{A_v} = \alpha|_{A_v} \wedge \alpha'|_{A_c} = \varepsilon \wedge \beta'|_{A_v \cup A_c} = \beta|_{A_v \cup A_c})].$$

This BSP is similar to SIA_V^{ρ}, except that the definition is not strict (perturbations of the front and back parts of the trace are possible). Our *f*uctive version of $IA_V^{\rho}(T)$ is given as follows:

$$IA_V^{\rho}(T) \mathrel{\widehat{=}} \forall \alpha \in A^{\infty} \, \forall \beta \in A^* \, \forall c \in A_c.$$
$$[(\beta \cdot \alpha \in T \wedge Adm_V^{\rho}(T, \beta, c)) \rightarrow$$
$$\exists \alpha' \in A^{\infty}. \exists \beta' \in A^*. \, (\beta' \cdot c \cdot \alpha' \in T \wedge \alpha' \approx_{A_v \cup A_c} \alpha \wedge \beta' \approx_{A_v \cup A_c} \beta)].$$

4.4 Coinductive Interpretation of Unwinding Relations

So far we have explored the coinductive interpretation of security-relevant hyperproperties and their building block BSPs. We now focus on coinductive unwinding relations. Instead of verifying BSPs directly or via global unwinding conditions, Mantel uses *local* unwinding conditions in his work [52]. Essentially, in order to prove that a number of BSPs hold and hence a particular policy is respected (see Figure 2.1) the existence of an unwinding relation satisfying a set of unwinding conditions has to be shown. In this section, we present a coinductive reinterpretation of Mantel's unwinding relations, which is needed in order for them to be suitable for non-terminating systems. We also show that the relations are instances of our H'-simulations.

4.4.1 Defining osc_V-*simulation*

The first relation is historically called *output-step consistency* and denoted osc_V. The original definition [55] is given as follows:

$$osc_V^o(T,R) \mathrel{\widehat{=}} \forall s_1, s_1', s_2' \in S \,.\, \forall a \in A \setminus A_c \,.$$

$$(reachable(SES, s_1) \wedge reachable(SES, s_1') \wedge s_1' \xrightarrow{a}_T s_2' \wedge (s_1', s_1) \in R) \rightarrow$$

$$\exists s_2 \in S \; \exists \alpha \in (A \setminus A_c)^* \,.\, (\alpha|_{A_v} = a|_{A_v} \wedge s_1 \xrightarrow{\alpha}_T s_2 \wedge (s_2', s_2) \in R).$$

Recall that *SES* is a state-event system given as a sextuple $(S, s_0, A, A_i, A_o, Tr)$, where S is the state-space, A is the alphabet, A_i and A_o are input and output alphabets respectively and Tr is a transition relation. Predicate $reachable(SES, s)$ means that state s is connected to the initial state s_0. Note that this definition is inductive in nature [1]. The use of predicate *reachable* means that the definition only refers to states that can be reached by the transition relation of the *SES* (and thus the set considered is *well-founded*[2]). This is a way to guarantee that there are no infinite decreasing sequences (of pairs) in R. As a result of this, the recursive calls are always applied to smaller arguments.

An osc_V-*simulation* on systems S, T is:

a relation $R \subseteq S \times T$ defined as follows: for all states $s \in S$, $t \in T$ if $(s,t) \in R$, then

$$o(s) \leftrightarrow o(t) \;\wedge\; \forall a \in A \setminus A_c \; \forall s_a \in S \,.\, (s \xrightarrow{a} s_a \rightarrow$$

$$\exists \sigma \in (A \setminus A_c)^* \; \exists t_\sigma \in T \,.\, (t \xrightarrow{\sigma} t_\sigma \wedge \sigma \approx_{A_v} a \wedge (s_a, t_\sigma) \in R)).$$

The predicate $osc_V(S,T,R)$ holds iff R is an osc_V-simulation on systems S and T. The gist of the definition of osc_V is very similar to the definition of osc_V^o, but note that it needs to be coinductive, namely taking the greatest fixed point interpretation of the respective rules, otherwise it would be potentially *non-well-founded* on infinite systems. Moreover, *reachable* is not in the definition, because the latter is coinductive: this means that there is no hierarchy of states, i.e. all pairs are on par for such a relation, and the relations are completely local [76]. Further note that because in trees there might be superimposed finite traces in addition to infinite ones (see Section 2.6),

[2] A set is well-founded [2] with respect to some accessibility relation R if all its elements are reachable by R.

we have the condition $o(s) \leftrightarrow o(t)$. This condition imposes termination-sensitivity in the definition. Finally, the definition of osc_V is more general than osc^o_V, as it needs not be on two copies of the same system; thus, it can be used for hyperproperty-preserving refinement.

4.4.2 Defining lrf_V-*simulation*

This relation is historically denoted lrf_V (locally respects forwards). It means that the state after performing a confidential event is related to the state before: the intuitive idea is that removing a confidential event should not have any effect on the low-observations, where the latter are typically formalized by osc_V (conditions on) relations. The original definition [55] is given as follows:

$$lrf^o_V(T,R) \mathrel{\widehat{=}} \forall s,s' \in S\,.\,\forall a \in A_c.$$
$$((reachable(SES,s) \wedge s \xrightarrow{a}_T s') \rightarrow (s',s) \in R).$$

We define an lrf_V-*simulation* coinductively as follows:

a relation $R \subseteq S \times T$ such that for all $s \in S, t \in T$ if $(s,t) \in R$, then

$$o(s) \leftrightarrow o(t) \;\wedge\; \forall a \in A_c\; \forall s_a \in S.\; (s \xrightarrow{a} s_a \rightarrow (s_a,t) \in R).$$

The predicate $lrf_V(S,T,R)$ holds iff R is an lrf_V-simulation on systems S and T. Again the rules of the relation are similar (modulo differences in the models), but we take a coinductive interpretation and will show that it is meaningful for reasoning about potentially infinite systems. As before, the predicate *reachable* is not needed for the coinductive version.

4.4.3 Defining lrb_V-*simulation*

This relation is historically denoted lrb_V (locally respects backwards). It means that at any state it is possible to perform any confidential event and the former state is related to the resulting state: the idea is that adding a confidential event should not have any effect on the low-observations, where the latter are typically formalized by osc_V (conditions on) relations.

The original definition [55] is given as follows:

$$lrb_V^o(T,R) \mathrel{\widehat{=}} \forall s \in S \,.\, \forall a \in A_c$$
$$(reachable(SES,s) \rightarrow \exists s' \in S \,.\, (s \xrightarrow{a}_T s' \wedge (s,s') \in R).$$

Next, define an *lrb_V-simulation* coinductively as follows:

a relation $R \subseteq S \times T$ such that for all $s \in S, t \in T$ if $(s,t) \in R$, then

$$o(s) \leftrightarrow o(t) \;\wedge\; \forall a \in A_c \; \forall t_a \in T.\, t \xrightarrow{a} t_a \wedge (s,t_a) \in R.$$

The predicate $lrb_V(S,T,R)$ holds iff R is an lrb_V-simulation on systems S and T. Again the rules of the relation are similar except for the coinductive interpretation and the predicate *reachable* is not needed for the coinductive version.

4.4.4 Defining $lrbe_V^{\rho}$-simulation

Intuitively, this relation is similar to lrb_V, except that we are only interested in hiding some subset of enabled (confidential) events (and not all confidential events). First, we define our version (i.e. for our model of systems) of $En_V^{\rho}(X,s,a)$, specifying whether event a is enabled in state s of system X with respect to a set of admissible events given by function ρ (see Sections 2.5.1 and 4.3). This is essentially the definition of $Adm_V^{\rho}(T,\beta,e)$ from Sections 2.5.1, but lifted to partial automata/trees (see Section 3.1).

$$En_V^{\rho}(X,s,a) \mathrel{\widehat{=}} \exists \beta,\gamma \in A^*.[test^*{}_{\beta}(X) \;\wedge\; X_{\beta} = s \;\wedge\; \gamma \approx_{\rho(V)} \beta \;\wedge test^*{}_{\gamma \cdot a}(X)].$$

The original definition [55] is given as follows:

$$lrbe_V^{\rho o}(T,R) \mathrel{\widehat{=}} \forall s \in S \,.\, \forall a \in A_c.$$
$$(reachable(SES,s) \wedge En_V^{\rho}(T,s,a) \rightarrow \exists s' \in S \,.\, (s \xrightarrow{a}_T s' \wedge (s,s') \in R)).$$

Next, define an *$lrbe_V^{\rho}$-simulation* relation as follows:

a relation $R \subseteq S \times T$ such that for all $s \in S, t \in T$ if $(s,t) \in R$, then

$$o(s) \leftrightarrow o(t) \wedge \forall a \in A_c.\ (En_V^{\rho}(T,t,a) \rightarrow (\exists t_a \in T\ .\ t \xrightarrow{a} t_a \wedge (s,t_a) \in R)).$$

The predicate $lrbe_V^{\rho}(S,T,R)$ holds iff R is an $lrbe_V^{\rho}$-simulation on systems S and T. The rules of the relation are similar except for the coinductive interpretation and the predicate *reachable* is not needed for the coinductive version.

4.4.5 Coinductive Unwinding Relations as H'-simulations

Next, we show that the coinductive unwinding relations defined so far are indeed H'-simulations. First, recall that incremental hyperproperties are coinductive predicates on trees [63]. Formally, an *H'-simulation* is an n-ary relation R such that $R \subseteq \Psi_{H'}(R)$. An H'-simulation corresponds to a monotone operator $\Psi_{H'}$ whose greatest fixed point is the coinductive predicate H'. Hence showing the existence of such a relation is sufficient to show that H' holds [63]. Because of the way the coinductive unwinding relations presented above are defined, it is obvious that $R \subseteq \Psi_{H'}(R)$ holds and $\Psi_{H'}$ is monotone (the defined relation occurs only positively in the formula); informally, $\Psi_{H'}$ is the "step" of the relation. Thus, the relations are indeed *H'-simulation* relations.

4.5 Coinductive Unwinding Framework

We have taken a coinductive perspective on Mantel's unwinding relations [55]. The high-level goal is to properly incorporate the unwinding relations in our framework in order to facilitate the verification of security-relevant hyperproperties. To show that we have succeeded in this endeavor, we present three types of theorems, similar to the ones initially introduced by Mantel in his framework: firstly, theorems connecting unwinding conditions and BSPs, secondly, ones connecting BSPs and holistic hyperproperties and finally, a version of Mantel's unwinding theorems.

The fact that we can prove these theorems implies that our definitions of unwinding relations, BSPs and holistic hyperproperties are reasonable and, more importantly, that unwinding in our framework is suitable for the verification of a number of security-relevant hyperproperties.

4.5.1 Unwinding Conditions for BSPs Theorems

The following two lemmas prove the property that states related by an osc_V-simulation relation are A_v-indistinguishable.

Lemma 4.5.1. *Let $T,S \subseteq \mathsf{Sys}$ be arbitrary systems and $R \subseteq T \times S$ be a relation. If $osc_V(T,S,R)$ holds then we have*

$$\forall \alpha_1 \in (A \setminus A_c)^*.(test^*{}_{\alpha_1}(T) \rightarrow$$

$$\exists \alpha_2 \in (A \setminus A_c)^*.\ (test^*{}_{\alpha_2}(S) \wedge \alpha_1 \approx_{A_v} \alpha_2 \wedge (T_{\alpha_1}, S_{\alpha_2}) \in R)).$$

This is a helper lemma, used to prove the following Lemma 4.5.2.

Lemma 4.5.2. *For all $T,S \subseteq \mathsf{Sys}$ if there exists $R \subseteq T \times S$ s.t. $osc_V(T,S,R)$ holds, then the following is valid:*

$$\forall \alpha_1 \in (A \setminus A_c)^{\infty}.(\alpha_1 \in T \rightarrow$$

$$\exists \alpha_2 \in (A \setminus A_c)^{\infty}.(\alpha_2 \in S \wedge \alpha_1 \approx_{A_v} \alpha_2)).$$

This lemma tells us that whenever $osc_V(T,S,R)$ holds, we have the following. For every trace t_1 in T, there must be a trace s_2 in S such that t_1 and s_2 are indistinguishable with respect to A_v.

The next result gives logically sufficient conditions (conjunctions of unwinding relations) for a number of BSPs. This is not surprising (Mantel presents a similar result), but it is nevertheless important, as the underlying definitions have changed.

Theorem 4.5.3. *Let T be an arbitrary system and $R \subseteq T \times T$ be a relation on T. The following implications are valid:*

1. $lrf_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow BSD_V(T)$
2. $lrf_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow D_V(T)$
3. $lrf_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow R_V(T)$
4. $lrbe^{\rho}_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow BSIA^{\rho}_V(T)$
5. $lrb_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow BSI_V(T)$.

The following theorem gives a conditional completeness result (when $A_n = \emptyset$) for some BSPs.

Theorem 4.5.4. *Consider a view (A_v, A_n, A_c) s.t. $A_n = \emptyset$ and some function ρ from views to subsets of A (see Section 2.5.1). The following are valid:*

1. *$BSD_V(T)$ implies there exists a relation $R \subseteq T \times T$ s.t. $lrf_V(T,T,R)$ and $osc_V(T,T,R)$ hold.*
2. *$BSIA_V^{\rho}(T)$ implies there exists a relation $R \subseteq T \times T$ s.t. $lrbe_V^{\rho}(T,T,R)$ and $osc_V(T,T,R)$ hold.*

4.5.2 Coinductive Version for BSPs to Holistic Hyperproperties Theorems

This section presents interesting results about using BSPs to compose the known, security-relevant hyperproperties introduced in Section 4.3.1. First, recall that $\mathcal{HI} = (L, H \setminus HI, HI)$. The instantiation of BSD_V with view $\mathcal{HI}$ is given as follows:

$$BSD_{\mathcal{HI}}(T) \mathrel{\widehat{=}} \forall \alpha \in A^{\infty}\, \forall \beta \in A^{*}\, \forall c \in HI\, .\, [(\beta \cdot c \cdot \alpha \in T \rightarrow \exists \alpha' \in A^{\infty}.\, (\beta \cdot \alpha' \in T \wedge \alpha' \approx_{L \cup HI} \alpha)].$$

The instantiation of BSI_V with view $\mathcal{HI}$ is given as follows:

$$BSI_{\mathcal{HI}}(T) \mathrel{\widehat{=}} \forall \alpha \in A^{\infty}\, \forall \beta \in A^{*}\, \forall c \in HI\, .\, [(\beta \cdot \alpha \in T \rightarrow \exists \alpha' \in A^{\infty}.\, (\beta \cdot c \cdot \alpha' \in T \wedge \alpha' \approx_{L \cup HI} \alpha)].$$

The following result establishes the connection between certain BSPs and *GNI*.

Theorem 4.5.5. *For all $T \subseteq \mathsf{Sys}$ we have $BSD_{\mathcal{HI}}(T) \wedge BSI_{\mathcal{HI}}(T)$ iff $GNI(T)$.*

This means that generalized noninterference *GNI* can be composed from the BSPs *BSD* and *BSI*, both instantiated with view $\mathcal{HI}$.

The next theorem establishes that the holistic hyperproperty *NF* is equivalent to the BSP *removal of events* instantiated with view $\mathcal{H}$.

Theorem 4.5.6. *For all $T \subseteq \mathsf{Sys}$ we have $R_{\mathcal{H}}(T)$ iff $NF(T)$.*

The following result is for the holistic hyperproperty *GNF*: *GNF* is equivalent to the BSP *removal of events*, when instantiated with view $\mathcal{HI}$.

Theorem 4.5.7. *For all $T \subseteq \mathsf{Sys}$ we have $R_{\mathcal{HI}}(T)$ iff $GNF(T)$.*

The following lemma is crucial for proving Theorem 4.5.11.

Lemma 4.5.8. *For all $T \subseteq \mathsf{Sys}$, the following hold:*
1. $SD_V(T) \rightarrow BSD_V(T)$
2. $SIA_V^{\rho}(T) \rightarrow BSIA_V^{\rho}(T)$
3. $BSD_V(T) \rightarrow R_V(T)$
4. $BSIA_V^{\rho}(T) \rightarrow IA_V^{\rho}(T)$.

The following two lemmas are important for proving a result (Theorem 4.5.11) about *PSP*.

Lemma 4.5.9. *For all $T \in \mathsf{Sys}$, the following holds: $PSP(T) \rightarrow (SD_{\mathcal{H}}(T) \wedge SIA_{\mathcal{H}}^{\rho_A}(T))$.*

Lemma 4.5.10. *For all $T \in \mathsf{Sys}$, the following holds: $(R_{\mathcal{H}}(T) \wedge IA_{\mathcal{H}}^{\rho_A}(T)) \rightarrow PSP(T)$.*

Finally, we have proven the following main *PSP* Theorem 4.5.11.

Theorem 4.5.11. *For all $T \subseteq \mathsf{Sys}$ we have $BSD_{\mathcal{H}}(T) \wedge BSIA_{\mathcal{H}}^{\rho_A}(T)$ iff $PSP(T)$.*

Proof. By mutual implication.
($\Rightarrow$) From the assumption and Lemma 4.5.8 (3 and 4) we get $(BSD_{\mathcal{H}}(T) \wedge BSIA_{\mathcal{H}}^{\rho_C}(T)) \rightarrow (\mathcal{R}_{\mathcal{H}}(T) \wedge IA_{\mathcal{H}}^{\rho_A}(T))$. Then from Lemma 4.5.10 we get $PSP(T)$, as needed.
($\Leftarrow$) From the assumption and Lemma 4.5.9 we get $PSP(T) \rightarrow (SD_{\mathcal{H}}(T) \wedge SIA_{\mathcal{H}}^{\rho_A}(T))$. Then from Lemma 4.5.8 (1 and 2) we get $BSD_{\mathcal{H}}(T) \wedge BSIA_{\mathcal{H}}^{\rho_C}(T)$, as needed. □

The theorem tells us that $PSP(T)$ is equivalent to $BSD_{\mathcal{H}}(T) \wedge BSIA_{\mathcal{H}}^{\rho_A}(T)$ for all $T \subseteq \mathsf{Sys}$. This theorem is important, because it significantly simplifies the problem of reasoning about *PSP*. We can instead reason about two BSPs, namely $BSD_{\mathcal{H}}(T)$ and $BSIA_{\mathcal{H}}^{\rho_A}(T)$. The latter are connected to specific unwinding conditions, as discussed in Section 4.5.1.

4.5.3 Coinductive Version of Mantel's Unwinding Theorems

Finally, we present the coinductive unwinding theorems for a number of known security-relevant hyperproperties. These unwinding theorems allow the specification and verification of the high-level policy by reasoning about the local states of the candidate system. Interestingly, there is a completeness result only for the definition of *PSP*. This means that only *PSP* is equivalent to a conjunction of unwinding relations; in the other cases, the proposed conjunctions of unwinding relations

imply the high-level policy (but they are not equivalence relations). Not having an equivalent incremental definition H' to a holistic hyperproperty H means that the non-existence of an H'-simulation does not imply that H does not hold for the system of interest. Similar theorems have been shown before [55], but for different unwinding relation, BSP and hyperproperty definitions and a different model (e.g. in the MAKS framework [55]).

Noninference *NF*

We have proven an unwinding theorem for *NF* that gives logically sufficient conditions for when the high-level policy *NF* holds for a system.

Theorem 4.5.12 (Unwinding of *NF*)**.** *If there exists a relation $R \subseteq T \times T$ such that $lrf_{\mathcal{H}}(T,T,R) \wedge osc_{\mathcal{H}}(T,T,R)$, we have that $NF(T)$ holds.*

Proof. Assume there exists a relation $R \subseteq T \times T$ such that $lrf_{\mathcal{H}}(T,T,R)$ and $osc_{\mathcal{H}}(T,T,R)$. From the unwinding conditions for BSPs Theorem 4.5.3 (3), it follows that $R_{\mathcal{H}}(T)$. Then $R_{\mathcal{H}}(T)$ implies $NF(T)$ by Theorem 4.5.6. □

Generalized Noninference *GNF*

Further, we have proven an unwinding theorem for *GNF*, again giving logically sufficient conditions for when the high-level policy *GNF* holds for a system.

Theorem 4.5.13 (Unwinding of *GNF*)**.** *If there exists a relation $R \subseteq T \times T$ such that $lrf_{\mathcal{H}\mathcal{I}}(T,T,R) \wedge osc_{\mathcal{H}\mathcal{I}}(T,T,R)$, we have that $GNF(T)$ holds.*

Proof. Assume there exists a relation $R \subseteq T \times T$ such that $lrf_{\mathcal{H}\mathcal{I}}(T,T,R)$ and $osc_{\mathcal{H}\mathcal{I}}(T,T,R)$. From the unwinding conditions for BSPs Theorem 4.5.3 (3), it follows that $R_{\mathcal{H}\mathcal{I}}(T)$. Then $R_{\mathcal{H}\mathcal{I}}(T)$ implies $GNF(T)$ by Theorem 4.5.7. □

Perfect Security Property *PSP*

The *Perfect Security Property* is special, as there are equivalent to it necessary as well as sufficient unwinding conditions. We have been able to show this in our framework as well. First, recall that $\rho_A(A_v,A_n,A_c) = A$.

Theorem 4.5.14 (Unwinding of *PSP*)**.** *If there exist relations $R,Q \subseteq T \times T$ such that $lrf_{\mathcal{H}}(T,T,R) \wedge osc_{\mathcal{H}}(T,T,R)$, as well as $lrbe^{\rho_A}_{\mathcal{H}}(T,T,Q) \wedge osc_{\mathcal{H}}(T,T,Q)$, we have that $PSP(T)$ holds.*

Proof. Assume there exist relations $R, Q \subseteq T \times T$ such that $lrf_{\mathcal{H}}(T,T,R)$ and $osc_{\mathcal{H}}(T,T,R)$, as well as $lrbe^{\rho_A}_{\mathcal{H}}(T,T,Q)$ and $osc_{\mathcal{H}}(T,T,Q)$ hold. By Theorem 4.5.3 (1 and 4), it follows that $BSD_{\mathcal{H}}(T) \wedge BSIA^{\rho_A}_{\mathcal{H}}(T)$. By Theorem 4.5.11 it follows that $PSP(T)$ holds. □

Theorem 4.5.15 (Completeness of Unwinding for *PSP*)**.** *If $PSP(T)$ holds then there exist relations $R, Q \subseteq T \times T$ such that $lrf_{\mathcal{H}}(T,T,R) \wedge osc_{\mathcal{H}}(T,T,R)$, as well as $lrbe^{\rho_A}_{\mathcal{H}}(T,T,Q) \wedge osc_{\mathcal{H}}(T,T,Q)$.*

Proof. Assume that $PSP(T)$ holds. From Theorem 4.5.11 we get $BSD_{\mathcal{H}}(T) \wedge BSIA^{\rho_A}_{\mathcal{H}}(T)$. From Theorem 4.5.4 it follows that there exist relations $R, Q \subseteq T \times T$ s.t. $lrf_{\mathcal{H}}(T,T,R)$, $osc_{\mathcal{H}}(T,T,R)$, $lrbe^{\rho_A}_{\mathcal{H}}(T,T,Q)$ and $osc_{\mathcal{H}}(T,T,Q)$. Note that for $\mathcal{H}$ we have $A_n = \emptyset$. □

Generalized Noninterference *GNI*

We have also been able to prove an unwinding theorem for *GNI*, giving logically sufficient conditions for when the high-level policy *GNI* holds for a system. Unfortunately, such conditions are sufficient but not necessary, which can be shown by counterexamples.

Theorem 4.5.16 (Unwinding of *GNI*)**.** *If there exist relations $R, Q \subseteq T \times T$ s.t. $lrf_{\mathcal{H}I}(T,T,R) \wedge osc_{\mathcal{H}I}(T,T,R)$, as well as $lrb_{\mathcal{H}I}(T,T,Q) \wedge osc_{\mathcal{H}I}(T,T,Q)$, we have that $GNI(T)$ holds.*

Proof. Assume there exist relations $R, Q \subseteq T \times T$ such that $lrf_{\mathcal{H}I}(T,T,R)$, $osc_{\mathcal{H}I}(T,T,R)$, also $lrb_{\mathcal{H}I}(T,T,Q)$ and $osc_{\mathcal{H}I}(T,T,Q)$. By Theorem 4.5.3 (1 and 5) it follows that $BSD_{\mathcal{H}I}(T)$ and $BSI_{\mathcal{H}I}(T)$. Then Theorem 4.5.5 gives us $GNI(T)$. □

The meaning of Theorem 4.5.16 in the context of the presented framework is illustrated in Figure 4.1. The different planes show respectively unwinding relations, BSPs and security-relevant hyperproperties. The arrows capture implications vs. equivalence statements, but these are also interacting with hidden conjunctions. For instance, there is a hidden conjunction present in Figure 4.1: $BSI_{\mathcal{H}I}(T) \wedge BSD_{\mathcal{H}I}(T)$ is equivalent to $GNI(T)$. Similarly, there is a hidden implication in the figure: $lrf_{\mathcal{H}I}(T,T,R) \wedge osc_{\mathcal{H}I}(T,T,R)$ implies $BSD_{\mathcal{H}I}(T)$.

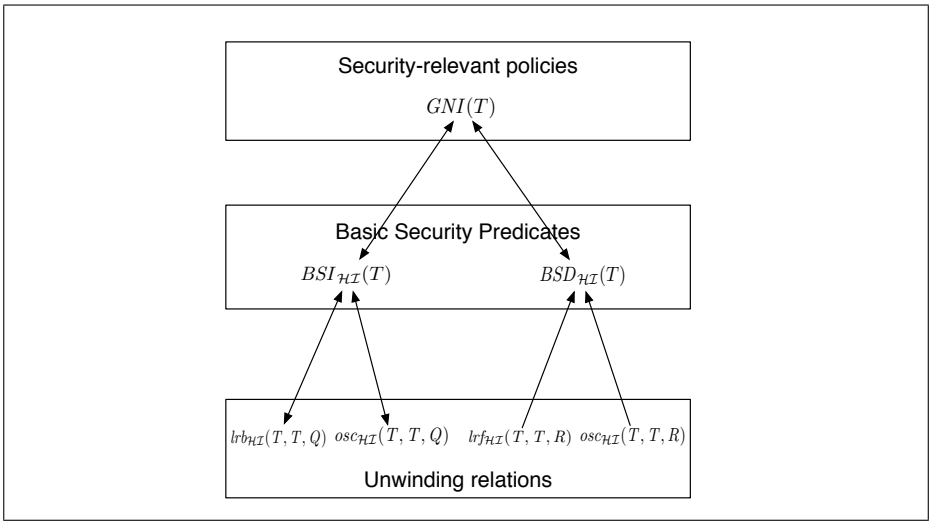


Figure 4.1: Meaning of Theorem 4.5.16

4.6 Coinductive Unwinding Theorems in Practice

This section demonstrates the potential use of the coinductive unwinding framework and particularly the unwinding theorems for the verification of concrete systems w.r.t. different policies. Before exploring these, recall the significance of H'-simulations for verification (see Chapter 3 and our recent paper [63]). In order to demonstrate that two systems T and S are H'-similar, it suffices to find an H'-simulation relation containing the pair (T,S). In order to show that $H(T)$ holds for some system T one needs to find an H'-simulation on $\overline{T}$ (see Section 3.4 for details).

Define the following sets: $L = \{l_1,l_2\}$, $H = \{h_1,h_2\}$, $HI = \{hi_1,hi_2\}$, $HO = \{ho_1,ho_2\}$. Recall that $\mathcal{HI} = (L,H \setminus HI,HI)$ whereas $\mathcal{H} = (L,\emptyset,H)$. Now we are ready to explore the use of the theory for the verification of sample systems with respect to security-relevant hyperproperties.

4.6.1 Noninference *NF*

Consider the security-relevant hyperproperty *noninference NF* from Section 4.3.1 and system $S = \{(l_2h_1l_1l_2 \mid l_2l_1l_2)^\omega\}$ with respect to view $\mathcal{H}$. It turns out that there exists relation R given as

$$R \quad = \quad \{(S,S),(S_{l_2h_1},S_{l_2}),(S_{l_2h_1l_1},S_{l_2l_1}),(S_{l_2l_1},S_{l_2l_1}),(S_{l_2},S_{l_2})\}$$

and illustrated in Figure 4.2, such that $osc_{\mathcal{H}}(R)$, $lrf_{\mathcal{H}}(R)$ and $(S,S) \in R$. The reader is invited to check that this is indeed the case. Hence, by Theorem 4.5.12 we have that system S is *secure* with respect to policy *NF*. Note that to make the figures more readable we use the following convention: whenever a pair of states in R resulted from an $osc_{\mathcal{H}}$ step, the states are connected with a dashed line. Alternatively, if the pair resulted from an $lrf_{\mathcal{H}}$ step, the states are connected with a solid line. Initial states are connected with a dotted line. This convention about relations is respected throughout this section.

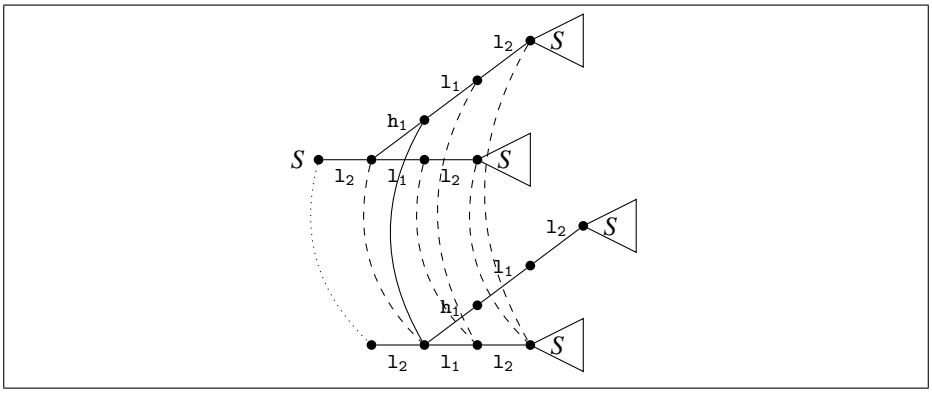


Figure 4.2: Illustration of an *NF*-simulation

Note that the definition of *NF* is very similar to the definition of *NI* from Section 2.4. However, our definition of *NF* includes the fairness constraint in itself, namely in the definition of ev_L, whereas the formalization of *NI* (as a part of class SHH) in Section 3.3.2 relied on the extra condition of systems to be in $L_{\Box\Diamond}$ (eventually a low event occurs in each trace and that happens infinitely often). It is also noteworthy that we had an equivalence of the incremental and holistic hyperproperty (i.e. $\mathsf{SHH}(T)$ iff $\mathsf{SIH}^2(T,T)$). The reason is that the view we considered did not have neutral events, i.e. $A_n = \emptyset$. Such a result (about completeness of the unwinding of *NF* when $A_n = \emptyset$) is also presented by Mantel [55] and can be easily adapted to our definition of *NF*.

4.6.2 Generalized Noninference *GNF*

The next security-relevant hyperproperty we consider is *generalized noninference GNF* from Section 4.3.1. The only difference between *GNF* and *NF* is the use of view $\mathcal{HI}$ instead of view $\mathcal{H}$. Recall that $\mathcal{HI} = (L, H \setminus HI, HI)$ whereas $\mathcal{H} = (L, \emptyset, H)$. Now consider system $S = \{(l_1 l_2 ho_1 ho_2 l_2 \mid l_1 l_2 ho_1 hi_1 ho_2 l_2)^\omega\}$. The relation R, as illustrated

in Figure 4.3, is given as follows

$$\begin{aligned} R \;=\; & \{(S,S),(S_{l_1},S_{l_1}),(S_{l_1l_2},S_{l_1l_2}),(S_{l_1l_2ho_1},S_{l_1l_2ho_1}),(S_{l_1l_2ho_1ho_2},S_{l_1l_2ho_1}), \\ & (S_{l_1l_2ho_1ho_2},S_{l_1l_2ho_1ho_2}),(S_{l_1l_2ho_1hi_1},S_{l_1l_2ho_1}),(S_{l_1l_2ho_1hi_1ho_2},S_{l_1l_2ho_1ho_2})\}. \end{aligned}$$

R satisfies the conjunction $lrf_{\mathcal{HI}}(R) \wedge osc_{\mathcal{HI}}(R)$. Again, the reader is invited to check that this is indeed the case. Hence by Theorem 4.5.13 system S is *secure* with respect to *GNF*.

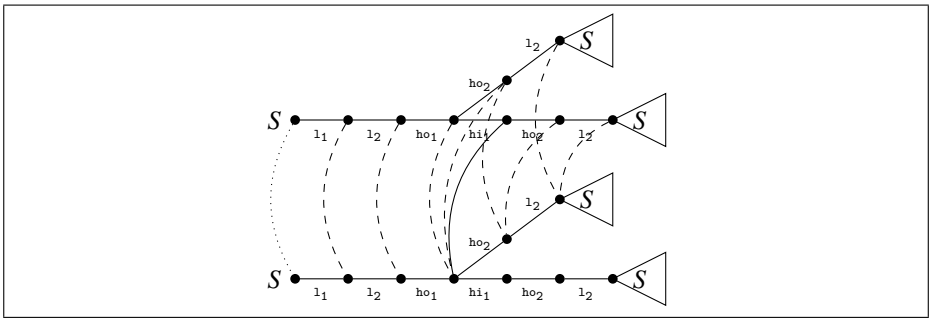


Figure 4.3: Illustration of a *GNF*-simulation

4.6.3 Perfect Security Property *PSP*

The next security-relevant hyperproperty we consider is the Perfect Security Property *PSP* from Section 4.3.1. Consider system $S = \{(l_2l_1l_2 \mid l_2h_1h_2l_1l_2 \mid l_2h_1l_1l_2)^\omega\}$. The relation R is given as follows:

$$\begin{aligned} R \;=\; & \{(S,S),(S_{l_2},S_{l_2}),(S_{l_2l_1},S_{l_2l_1}),(S_{l_2h_1},S_{l_2}), \\ & (S_{l_2h_1h_2},S_{l_2}),(S_{l_2h_1h_2l_1},S_{l_2l_1}),(S_{l_2h_1l_1},S_{l_2l_1})\}. \end{aligned}$$

R satisfies the conjunction $osc_{\mathcal{H}}(S,S,R) \wedge lrf_{\mathcal{H}}(S,S,R)$. The relation can be seen in Figure 4.4.

The relation Q is given as follows:

$$\begin{aligned} Q \;=\; & \{(S,S),(S_{l_2},S_{l_2}),(S_{l_2l_1},S_{l_2l_1}),(S_{l_2},S_{l_2h_1}), \\ & (S_{l_2},S_{l_2h_1h_2}),(S_{l_2l_1},S_{l_2h_1h_2l_1}),(S_{l_2l_1},S_{l_2h_1l_1})\}. \end{aligned}$$

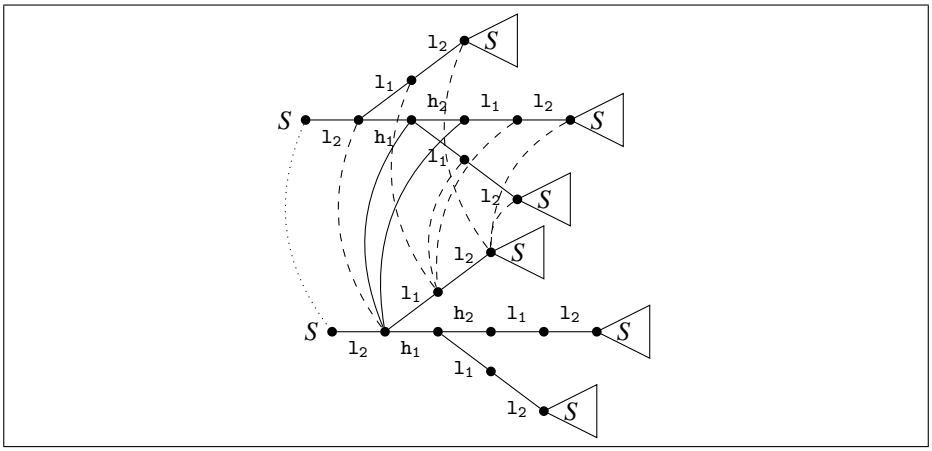


Figure 4.4: Illustration of a relation R such that $osc_{\mathcal{H}}(S,S,R)$ and $lrf_{\mathcal{H}}(S,S,R)$

Relation Q, as depicted in Figure 4.5, satisfies the conjunction $osc_{\mathcal{H}}(S,S,Q) \wedge lrbe^{\rho A}_{\mathcal{H}}(S,S,Q)$. Note that the part of Q resulting from $osc_{\mathcal{H}}$ is dashed and the part from $lrbe^{\rho A}_{\mathcal{H}}$ is solid. The existence of R and Q, together with Theorem 4.5.14 imply that system S is *secure* with respect to *PSP*.

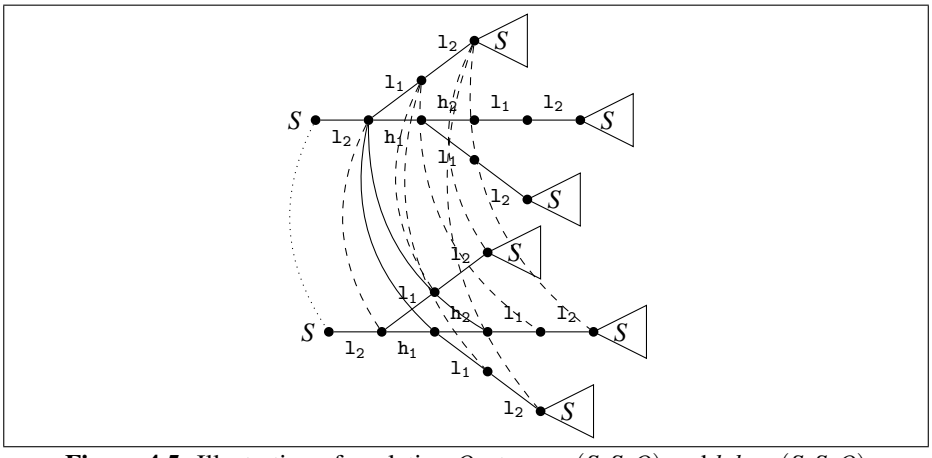


Figure 4.5: Illustration of a relation Q s.t. $osc_{\mathcal{H}}(S,S,Q)$ and $lrbe_{\mathcal{H}}(S,S,Q)$

4.6.4 Generalized Noninterference *GNI*

By Theorems 4.5.16 and 4.5.13, we have that *GNI* is stronger that *GNF*. That is, in order for *GNI* to hold for some S it has to be that *GNF* holds and there must also exist some Q s.t. $lrb_{\mathcal{HI}}(S,S,Q)$ and $osc_{\mathcal{HI}}(S,S,Q)$. That means that in order to check *GNI*, one has to first check *GNF* as in Section 4.6.2 and then in a similar way try to check the existence of a relation Q with $lrb_{\mathcal{HI}}(S,S,Q)$ and $osc_{\mathcal{HI}}(S,S,Q)$ satisfied. This demonstrates that the framework is still modular.

4.7 Discussion and Related Work

We have presented a new mathematical development enabling the application of unwinding relations to the verification of security-relevant hyperproperties. We have also demonstrated the potential of a modular framework for coinductive reasoning about hyperproperties. This is achieved by combining the framework from Chapter 3 with a coinductive reinterpretation of a class of possibilistic security-relevant hyperproperties, Mantel's BSPs and unwinding relations. It should be reiterated that our proposed coinductive variants of the BSPs are not equivalent to Mantel's on finite systems, nevertheless their conjunctions still imply the desired high-level hyperproperties.

The coinductive reinterpretation of Mantel's unwinding relations are instances of our H'-simulations (see Chapter 3). More precisely, an H'-simulation is a (conjunction of) unwinding relation(s). This realizes a connection between unwinding relations and our incremental hyperproperties: incremental security hyperproperties can be seen as conjunctions of coinductively-defined unwinding relations, or alternatively as (conjunctions of) H'-simulations, implying the respective high level policies. This is obvious if we consider some of the incrementalizable classes from Chapter 3, particularly SHH and OHH. Their equivalent simulation relations can be seen as a conjunction of unwinding relations.

We have demonstrated that the class of incremental hyperproperties is substantial. Moreover, we can reuse the techniques introduced in Chapter 3 (via H'-simulations) and further developed in Chapters 5 and 6 for verification of security-relevant hyperproperties from the class of possibilistic information flow policies [55] (instances of liveness hyperproperties). As a result, our verification method is the first one that works on a class of liveness hyperproperties. Finding the precise class for which the methodology works is an interesting direction for future work.

To the best of our knowledge, the only related paper that explores nonterminating behaviors and identifies the need for a coinductive interpretation of noninterference for potentially nonterminating systems is by Bohannon et. al [11]. They introduce

the notion of *reactive noninterference* and explore variants suitable for reactive systems. The main similarity with our work is that they use coinductive and inductive/coinductive definitions in order to define relations on finite and infinite streams. They convert their high-level, holistic definition into a relation (called *ID-bisimulation*) on program states; they show that their ID-bisimulation implies the high-level, holistic policy. Their ID-bisimulation is an incremental hyperproperty.

It should be noted that many of the theorems presented in this chapter are not constructive: they do not give an algorithm for finding unwinding relations/H'-simulations. Nevertheless, as we argued in Chapter 3, there are model checking techniques giving a practical means to compute or approximate the greatest fixed point of a monotone operator. Such model checking techniques will be explored in Chapters 5 and 6.

4.8 Summary

We have proposed a framework suitable for coinductive reasoning about holistic hyperproperties in general and illustrated its usefulness by exploring a new coinductive reinterpretation of known noninterference policies as hyperproperties. The framework is modular, as it permits the expression of a number of security-relevant hyperproperties as conjunctions of variants of Mantel's BSPs. We have demonstrated the usefulness of coinductive unwinding relations for reasoning about hyperproperties. In particular, we have presented unwinding theorems for generalized noninterference [58], noninference [86], generalized noninference [86] and the perfect security property [86]. Moreover, we have proven results connecting unwinding relations and BSPs, relating different BSPs and relating BSPs and holistic hyperproperties.

To the best of our knowledge, the results are novel in several ways. First they further extend our recently proposed framework for reasoning about hyperproperties (presented in Chapter 3) and establish a connection with the most relevant (in our opinion) work on verification via unwinding [52]. They also identify and illustrate the potential of unwinding relations (which turn out to be instances of our H'-simulations) for generic verification of hyperproperties. In particular, H'-simulations can be used to verify a class of liveness hyperproperties (so far, there was a verification methodology for k-safety hyperproperties only [18, 19]). Further, the results show that coinductively defined hyperproperties are important not only from a theoretical standpoint, but also in practice, due to the abundance of reactive systems. Finally, the results shed light on the significance of incremental hyperproperties.

In the next chapter, we present a verification methodology for incremental hyperproperties via games.

Chapter 5

Incremental Hyperproperties as Games

In this chapter we zoom in on the problem of incremental hyperproperty verification. The main contribution is a game semantics for incremental hyperproperties. We demonstrate that interpreting incremental hyperproperties as games is important not only theoretically, but also in practice: it can be used as a basis for model checking incremental hyperproperties.

In Chapter 3 we defined an incremental hyperproperty H' to be the greatest fixed point of a monotone function over k-tuples of systems. The respective notion of an H'-simulation relation is convenient for verification: showing the existence of such a relation is sufficient to show that H' holds. Furthermore, searching for an H'-simulation can be done using the notion of fixed point approximants [82]. In practice, however, calculating fixed points is tedious and error prone, especially when alternation of fixed point operators is present [82]. To deal with this problem we propose to interpret incremental hyperproperties as games. This enables the conversion of the search for an H'-simulation into the search for a history-free, winning strategy in the respective incremental hyperproperty game. That search can be performed using game-based model checking techniques.

5.1 Synopsis

We start by revisiting incremental hyperproperty logic IL from Chapter 3 and demonstrate that the logic is not expressive enough for certain incremental hyperproperties.

We next introduce H'-*simulation games*, which can be seen as a generalization of Stirling's equivalence games [82]. H'-simulation games result directly from definitions of H'-simulations. The approach offers a very intuitive way of thinking about incremental hyperproperties and coinductive predicates in general. It is also a good demonstration of both the type of results we would like to have for incremental hyperproperty games and some of the respective shortcomings (e.g. the lack of generality).

Further, we show that Andersen's polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [8], presented in Section 2.9, is a suitable logic for expressing all known incremental hyperproperties and H'-simulation relations. The main contribution of this chapter is then presented: a novel characterization of the satisfaction relation between a system and an incremental hyperproperty given in $\mathcal{L}_\mu^k$, in terms of playing a game. To be as general as possible, we present *incremental hyperproperty checking games* for the full $\mathcal{L}_\mu^k$. This is useful, as some incremental hyperproperties happen to fall outside of our initially proposed LFP fragment $I\mathcal{L}$, but are expressible in $\mathcal{L}_\mu^k$. Instances of such relations include some of our coinductive unwinding relations from Chapter 4.

It turns out that a fragment of $\mathcal{L}_\mu^k$ suffices for expressing the known notions of incremental hyperproperties, including the instances from Chapter 4, as well as the relevant fairness constraints. The new logic is called $I\mathcal{L}_\mu^k$ and allows (a restricted variant of) coinductive/inductive definitions or in other words both greatest and least fixed points in the logic. As the proposed logic is expressive enough to capture the known H'-simulations including our coinductive unwinding relations and thus the coinductive reinterpretation of Mantel's framework, we argue that it captures a large class of interesting security policies. The chapter is based on extended versions of recent papers of ours [65, 64]. The relevant proofs can be found in Appendix C.

5.2 Revisiting Incremental Hyperproperty Logic

In Chapter 3 we presented the logical language $I\mathcal{L}$ and used it to formalize incremental hyperproperties. Unfortunately, it was later realized that $I\mathcal{L}$ is not expressive enough to capture all incremental hyperproperties of interest. For instance, some of the H'-simulations presented in Chapter 4 are not expressible in $I\mathcal{L}$. The relations in question require not only least and greatest fixed point operators, but also alternation of the fixed points. To illustrate the problem we will present one such example based on definitions from Chapter 4.

Recall that the extension of predicate *test* to the obvious inductive definition $test^* : \mathsf{Sys} \to (A^* \to 2)$ on words is given as follows. For $a \in A$, $w \in A^*$ and ε the empty trace, $test^*$ is defined as:

$$\frac{o(X)}{test^*{}_{\varepsilon}(X)} \qquad \frac{test_a(X) \qquad test^*{}_w(X_a)}{test^*{}_{a\cdot w}(X)}$$

For tree T and word w, the predicate $test^*{}_w(T)$ is true whenever the finite sequence w can be performed, starting at the root of the tree T.

We also recall the inductively defined relation $w \rightsquigarrow_Z a \cdot w'$ — for $Z \subseteq A$, w Z-reveals a with tail w', is defined as:

$$\frac{}{\varepsilon \rightsquigarrow_Z \varepsilon} \qquad \frac{a \in Z}{a\cdot w \rightsquigarrow_Z a \cdot w} \qquad \frac{b \in A \setminus Z \qquad w \rightsquigarrow_Z a\cdot w'}{b\cdot w \rightsquigarrow_Z a \cdot w'}$$

Finally, recall the definition of *weak bisimulation* $\approx_Z : A^\infty \times A^\infty \to 2$ (parameterized by set Z):

$$\frac{}{\varepsilon \approx_Z \varepsilon}\ {\scriptstyle coind} \qquad \frac{w \rightsquigarrow_Z a\cdot w' \qquad u \rightsquigarrow_Z a \cdot u' \qquad w' \approx_Z u'}{w \approx_Z u}\ {\scriptstyle coind}$$

We are now ready to present the promised incremental hyperproperty that is not expressible in IL. This is a variant (identical to our original definition, but using *test* and *test**) of the coinductively defined *osc*$_V$*-similarity* — the union of all *osc*$_V$-simulations (see Chapter 4 for a definition of *osc*$_V$-simulation):

$$osc_V(X,Y) \mathrel{\widehat{=}} \mathbf{gfp}\, R(X,Y)\,.\,(o(X) \leftrightarrow o(Y)) \bigwedge \forall a \in A \setminus A_c.\,(test_a(X) \rightarrow$$
$$\exists \sigma \in (A \setminus A_c)^*.(test^*{}_\sigma(Y) \wedge a \approx_{A_v} \sigma \wedge R(X_a, Y_\sigma)))).$$

The definition cannot be expressed in IL. Intuitively, the reason is the alternation of the greatest and least fixed point operators, which arise, for instance, in the definition of $\approx_{A_v}$, but also from the interaction of *test** with the outermost **gfp** operator. Unfortunately, such alternation is not available in IL.

In order to preempt the problem of discovering definitions that require further extension to some more expressive LFP fragment, we choose to work with a relatively large but still tractable fragment of LFP: the polyadic modal mu-calculus $\mathcal{L}^k_\mu$ [8]. Most importantly though, $\mathcal{L}^k_\mu$ is able to express all incremental hyperproperties we have considered.

Here is a short outline of the rest of this chapter. First, we motivate the work by presenting a direct approach to converting H'-simulations into games. Second, we propose our *incremental hyperproperty checking games* for the full polyadic modal mu-calculus $\mathcal{L}^k_\mu$. Finally, we motivate and present the logic IL^k_μ.

5.3 From H'-simulations to H'-simulation Games

In Chapter 3 we identified several (classes of) incremental hyperproperties and their respective simulation relations. One way to represent an H'-simulation relation is in terms of playing a game. The prototypical example of such a game is Stirling's *equivalence game* [82] — an interactive game played by two players, namely R (refuter) and V (verifier). The *arena* of the game is a pair of systems; the rules require that for every turn, first R chooses one system and performs an action $a \in A$, then V has to respond with the same action a in the other system. A player wins when the other one cannot move. Additionally, V wins all infinite plays.

Definition 5.3.1. A game is *determined* if there is a winning strategy for one of the players from each position.

Stirling shows that equivalence games are determined and that two processes are bisimilar iff they are game equivalent [82], i.e. iff V has a history-free winning strategy for the respective equivalence game. The H'-simulation games we introduce here are different, though, as they are far more general and need not correspond to equivalence relations. Furthermore, *H'-simulation games* generalize equivalence games.

Definition 5.3.2. An *H'-simulation game*, denoted $G_{H'}(X^1, \ldots, X^k)$, is an interactive game played by players R and V, making choices in turns what the next transition is. The arena is given by a k-tuple of trees $T^1, \ldots, T^k$. A *play* of an H'-simulation game is a finite or infinite sequence of k-tuples of trees

$$(T_0^1, \ldots, T_0^k), \ldots, (T_i^1, \ldots, T_i^k), \ldots.$$

In an H'-simulation game player R tries to show that a certain coinductive predicate $H'(T^1, \ldots, T^k)$, given in IL and corresponding to the game played $G_{H'}(T^1, \ldots, T^k)$, is false. V tries to show that $H'(T^1, \ldots, T^k)$ holds. The rules of the game depend on the concrete definition of H'-simulation. They can be roughly seen as coming from the model checking games for first-order logic [33]. The rules are not formalized here, as IL is not the right logic for incremental hyperproperties, as argued in Section 5.2. Nevertheless, the potential approach is outlined to give some intuition about the connection between H'-simulations and games.

Definition 5.3.3. Any position $(T_i^1, \ldots, T_j^k)$ where R can perform an action and V *cannot* respond to the R-move, but is required to do so by the rules of the game, is called an *R-win*.

We have chosen to model the fact that player V is not always forced to make a move, by introducing the notion of *null move*: a *null move* does not change the current position in a game, however it changes whose turn it is.

Initially, all trees have an R-token or a V-token placed on their roots. The refuter makes the first move(s) and the verifier responds with a transition(s), obeying the rules of the game. A play goes on until one of the players wins. A play is won by R if it reaches an R-win position. Any other play is won by V, i.e. the verifier wins all infinite plays and any finite play that reaches a position $(T_i^1, \ldots, T_j^k)$ in which the refuter cannot move.

We continue by recalling some definitions from Section 2.10. A *strategy* is a family of rules prescribing how the players move. A *history-free strategy* is a strategy that does not depend on what happened previously in the play. A player P *uses* a strategy if all her moves conform to the rules of the strategy. A *winning strategy* for player P is a strategy that guarantees that player P wins all plays using the strategy.

Next, we present a sample H'-simulation game and its related results.

Example 5.3.4. *We first recall some definitions and notation from Section 3.3.2. Let $p : A \to 2$ be an arbitrary predicate. We can convert any predicate $p : A \to 2$ to a propositional constant as follows: for a trace t we say that $t \models p$ iff $p(t(0))$ holds. Let $P_{\Box\Diamond} = \{t \in A^\omega \mid t \models \Box\Diamond p\}$ be the set of infinite traces satisfying the temporal logic formula $\Box\Diamond p$, based on the temporal logic modalities* eventually *($\Diamond$) and* always *($\Box$). The hyperproperty class SIH^2 from Section 3.3.2, which is only defined for systems in $P_{\Box\Diamond}$, can be given as follows:*

$$\mathsf{SIH}^2 \mathrel{\widehat{=}} \textbf{\textit{gfp}}\, I(X,Y).$$

$$(\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ X \xrightarrow{a} X_a \wedge p(a) \to \exists Y_b \subseteq \mathsf{Sys}.\ Y \xrightarrow{f(a)} Y_b \wedge I(X_a, Y_b)$$

$$\wedge\ \forall a \in A\ \forall X_a \subseteq \mathsf{Sys}.\ \neg p(a) \to I(X_a, Y)).$$

where p is a predicate on A and f is an endofunction on A (i.e. $f : A \to A$). Instantiating H' with SIH^2, we first get the notion of a SIH^2-simulation relation and then its corresponding SIH^2-simulation game $G_{\mathsf{SIH}^2}(X,Y)$. Recall that a SIH^2-simulation on systems S,T is a relation $Q \subseteq S \times T$, s.t. for all $s \in S$, $t \in T$ we have the following: if $(s,t) \in Q$ then

$$\forall a \in A\ \forall s_a \in \mathsf{Sys}\ .\ s \xrightarrow{a} s_a \wedge p(a) \to (\exists b \in A\ \exists t_b \in \mathsf{Sys}\ .\ t \xrightarrow{b} t_b \wedge b = f(a) \wedge (s_a, t_b) \in Q$$

$$\wedge\ \forall a \in A\ \forall s_a \in \mathsf{Sys}\ .\ s \xrightarrow{a} s_a \wedge \neg p(a) \to (s_a, t) \in Q).$$

Recall also that two systems S and T are SIH^2-similar iff there exists a SIH-simulation relation Q on the state spaces of S and T, such that $(s_0, t_0) \in Q$, where s_0 and t_0 are the respective start states.

Rules of the SIH^2-simulation game $G_{\mathsf{SIH}^2}(X,Y)$.
The rules of $G_{\mathsf{SIH}^2}(X,Y)$ are based on the definition of SIH^2-simulation (or alternatively on the definition of SIH^2). At position (T_i^1, T_j^2), player R makes a move by

choosing some $a \in A$ such that $test_a(T_i^1)$. Then player V has to respond with move $f(a)$, but only if $p(a)$ holds. Otherwise, V responds with a null move and R has her turn again. The verifier always knows the move of the refuter. Any position (T_i^1, T_j^2) where R can perform an action $a \in A$ such that $p(a)$ holds and V cannot respond to the R-move is an R-win. A play goes on until one of the players wins. Any play that is not won by R is won by V, i.e. the verifier wins all infinite plays and any finite play that reaches a position (T_i^1, T_j^2), from which the refuter cannot move.

A substantial difference between H'-simulation games and Stirling's equivalence games [82] is that in the former each player has their own token and uses it to traverse their own tree individually; in equivalence games, however, both players share the tokens and at each round R may choose in which tree (i.e. with which token) to make the next move, whereas V has to respond in the other tree(s). This is a consequence of the fact that only equivalence relations are considered. Another difference is that in H'-simulation games it is possible that R goes arbitrarily ahead of V, because player V may make a null move. This results in potential fairness problems, which will be dealt with in the next section.

Proposition 5.3.5. *For any SIH^2-simulation game $G_{\mathsf{SIH}^2}(S,T)$, where $S,T \subseteq P_{\Box\Diamond}$, either player R or player V has a history-free winning strategy.*

This proposition says two things:

1. Any SIH^2-simulation game $G_{\mathsf{SIH}^2}(S,T)$ is determined.
2. It is sufficient to consider history-free strategies, which considerably simplifies the problem of searching for a strategy.

Of course we have to prove that SIH^2-simulation games indeed capture what we hope, namely the meaning of the coinductive predicate $\mathsf{SIH}^2(S,T)$. The following theorem guarantees this.

Theorem 5.3.6 (Correctness of SIH^2-simulation games)**.** *The coinductive predicate $\mathsf{SIH}^2(S,T)$, where $S,T \subseteq P_{\Box\Diamond}$, holds iff V has a history-free winning strategy for $G_{\mathsf{SIH}^2}(S,T)$.*

This theorem gives a game theoretic characterization of when a coinductive predicate from the class SIH^2 holds. Next, we further explore the repercussions of Theorem 5.3.6. Recall the holistic hyperproperty class SHH, which gives rise to SIH^2. SHH was defined as follows:

$$\mathsf{SHH}(X) \mathrel{\widehat{=}} \forall x \in X\ \exists y \in X.\ p_s(y) \wedge x \sim_{\mathrm{p}} y,$$

where $\sim_{\mathrm{p}} : A^\omega \times A^\omega \to 2$ and $p_s : A^\omega \to 2$ are as defined in Section 3.3.2.

In Chapter 3, we showed that for all $T \subseteq P_{\Box\Diamond}$, we have that $\mathsf{SHH}(T)$ iff $\mathsf{SIH}^2(T,T)$. As a consequence of this and Theorem 5.3.6, we have the following result:

Corollary 5.3.7. *A system $T \subseteq P_{\Box\Diamond}$ satisfies a holistic hyperproperty from class* SHH*, i.e.* $\mathsf{SHH}(T)$*, iff V has a history-free winning strategy for $G_{\mathsf{SIH}^2}(T,T)$.*

Example 5.3.8. *To further illustrate the usefulness of Theorem 5.3.6, consider the definition of an NI$'$-simulation relation from Chapter 3. An NI$'$-simulation on systems S,T is a relation $Q \subseteq S \times T$, s.t. for all $s \in S, t \in T$ we have the following: if $(s,t) \in Q$ then*

$$\forall a \in A\, \forall s_a \in \mathsf{Sys}\,.\, s \xrightarrow{a} s_a \wedge \neg high(a) \to \exists t_a \in \mathsf{Sys}\,.\, t \xrightarrow{a} t_a \wedge Q(s_a,t_a)$$

$$\wedge\, \forall a \in A\, \forall s_a \in \mathsf{Sys}\,.\, s \xrightarrow{a} s_a \wedge high(a) \to Q(s_a,t),$$

where low and high are predicates on A s.t. $low = \neg high$. Note that NI$'$ is in the syntactic class SIH^2.

Rules of the NI$'$-simulation game $G_{NI'}(X,Y)$.
The NI$'$-simulation game $G_{NI'}(X,Y)$ is an instantiation of $G_{\mathsf{SIH}^2}(X,Y)$ with function f being identity on A and p being low. At any position (T_i^1, T_j^2), player R choses some move $a \in A$ such that $T_i^1 \xrightarrow{a} T_{i+1}^1$. Then player V has to respond with move a, but only if $low(a)$ holds. Otherwise V responds with a null move and R has her turn again. Any position (T_i^1, T_j^2) where R can perform an action a such that $low(a)$ holds and V cannot respond is an R-win. As usual, a play goes on until one of the players wins. Any play that does not lead to an R-win is won by V, including all infinite plays.

Next, we illustrate the game on a concrete system. Let $A = \{a,b,c,d\}$ be the alphabet and define predicates high and low on elements of A as follows: $high(a)$, $high(b)$, $low(c)$, $low(d)$. Consider the infinite tree T in Figure 5.1, containing $(acbd)^\omega$, $(cda)^\omega$ and all streams resulting from some interleaving of acbd and cda. Formally, the traces of the tree T can be given as $(acbd \mid cda)^\omega$. Alternatively, the tree can be given as the coinductive interpretation of the following grammar:

$$T = acbdT \mid cdaT.$$

One possible play of $G_{NI'}(T,T)$, displayed in Figure 5.2, is given as follows:

$$(T,T),(T_a,T),(T_{ac},T_c),(T_{acb},T_c),(T,T_{cd}),(T_a,T_{cd}).$$

Note that the last position (T_a,T_{cd}) is an R-win, because R can move c which cannot be matched by V. Another possible play is:

$$(T,T),(T_c,T_c),(T_{cd},T_{cd}),(T,T_{cd}).$$

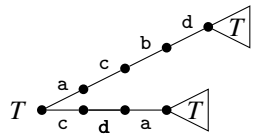


Figure 5.1: Tree T

The last position is also an R-win, because R can move c but this cannot be matched by V. These are the only possible nontrivial[1] *plays. The reader is invited to check that verifier does not have a winning strategy for* $G_{NI'}(T,T)$*; the reasons behind such an argument have been presented above. Hence, we can conclude that* $\neg NI'(T,T)$ *holds. By Corollary 5.3.7 and because NI is in class* SHH*, we also have that* $\neg NI(T)$.

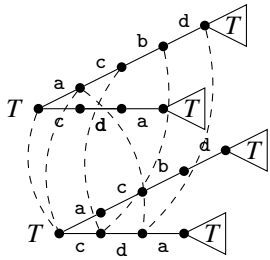


Figure 5.2: A play of the NI'-simulation game on two copies of T.

The approach outlined here is very intuitive and might form the underpinning of efficient algorithms (as is the case for bisimulation [81]). However, we consider it insufficient, because every instance of an H'-simulation requires the construction of a specific game and hence one would need to prove the correctness of the respective game for each case, in a result similar to Theorem 5.3.6. As there are many different security-relevant incremental hyperproperties (and corresponding H'-simulations), this is not optimal.

Moreover, we need a more expressive logic than IL. As we argued in Section 5.2 and demonstrated in Chapter 4, in practice we want to reason about H'-simulations that also have alternation between the least and greatest fixed point operators. Constructing such games and reasoning about them is not as intuitive and straightforward as in the case for purely coinductive predicates (such as SIH^2 and NI' from Example 5.3.8). Finally, the encoding of fairness also requires a least fixed point operator and

[1] Trivial plays are modifications of the presented ones, for which R immediately wins, e.g. when V moves a c at position (T, T_{cd}).

alternation. For instance, the fairness constraint for SIH^2 was encoded into condition $P_{\Box\Diamond}$, which needs to be checked separately. We would like to use a logic that is expressive enough to include similar constraints in the incremental hyperproperty definition itself, and at the same time be tractable enough so that we can perform model checking. Such logics are presented in Sections 2.9 and 5.6. Obviously, the existence of tool support would be a significant asset. This is further explored in Chapter 6.

Nevertheless, H'-simulation games can be considered important in their own right in the same way that equivalence games are considered important. H'-simulation games also give a very intuitive account of the satisfaction relation between a system and an incremental hyperproperty, facilitating reasoning about the latter. Moreover, H'-simulation games have given us an intuition of what results would be nice to have, namely a generic incremental hyperproperty checking game (IHP checking game), such that the incremental hyperproperty H' would hold iff V has a winning strategy for the respective IHP checking game. Finding such games and a logic that is tractable and expressive enough for IHPs is the main motivation and accomplishment of this chapter.

5.4 Incremental Hyperproperty Checking Games

Andersen's polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [8] was presented in Section 2.9. In this section, we propose a game theoretic characterization of when an incremental hyperproperty expressed in $\mathcal{L}_\mu^k$ holds for a particular k-tuple of trees $\overline{\mathcal{T}}$, relative to some second-order valuation V. As in Section 5.3, the games presented here will be played by two players: refuter (R) and verifier (V). Player R attempts to disprove that a particular k-tuple of trees $\overline{\mathcal{T}}$ satisfies an incremental hyperproperty H' expressed in $\mathcal{L}_\mu^k$, whereas player V attempts to prove that H' holds for $\overline{\mathcal{T}}$. The idea is similar to the property-checking games for the modal mu-calculus [82]: we present a game for hyperproperty checking, such that a system satisfies a hyperproperty whenever player V has a winning strategy for the respective game.

We start by giving some necessary definitions. Let $\sigma = \{\mu, \nu\}$.

Definition 5.4.1. Let Φ be a normal formula (see Definition 2.9.2) expressing an incremental hyperproperty. A *play* of the incremental hyperproperty checking game $HG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ is a finite or infinite sequence of pairs of k-tuples of trees and $\mathcal{L}_\mu^k$ formulae:

$$((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0),\ldots,((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n),\ldots.$$

Note that each formula Φ_i is a subformula of Φ_0 and each tree T_i^j is a subtree of T_0^j. Also note that $T_0^1,\ldots,T_0^m,\ldots,T_0^k$ are not necessarily the same tree. The next move(s) in a play from any position $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)$, as well as who makes them, depends on the main connective in Φ_n. The possible moves are given in Figure 5.3.

- If $\Phi_n = \Psi_1 \wedge \Psi_2$, then R choses one of the conjuncts Ψ_i ($i \in \{1,2\}$), the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged and formula $\Phi_{n+1} = \Psi_i$.
- If $\Phi_n = \Psi_1 \vee \Psi_2$, then V choses one of the disjuncts Ψ_i ($i \in \{1,2\}$), the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged and formula $\Phi_{n+1} = \Psi_i$.
- If $\Phi_n = [a]_m\Psi$, then R has to move along the transition $T_j^m \xrightarrow{a} T_{j+1}^m$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ becomes $(T_i^0,\ldots,T_{j+1}^m,\ldots,T_l^k)$ and formula $\Phi_{n+1} = \Psi$.
- If $\Phi_n = \langle a\rangle_m\Psi$, then V has to move along the transition $T_j^m \xrightarrow{a} T_{j+1}^m$, the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ becomes $(T_i^0,\ldots,T_{j+1}^m,\ldots,T_l^k)$ and formula $\Phi_{n+1} = \Psi$.
- If $\Phi_n = \sigma Z.\Psi$, then formula Φ_{n+1} becomes Z and the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged.
- If $\Phi_n = Z$ and the subformula of Φ_0 identified by Z is $\sigma Z.\Psi$, then formula $\Phi_{n+1} = \Psi$ and the k-tuple of trees $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)$ remains unchanged.

Figure 5.3: Rules specifying the next possible move(s) from some position $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)$

Unlike in the case with H'-simulation games, players are not strictly required to take turns. Player R has her turn when the next main connective is $\wedge$ or $[a]$. Dually, player V has her turn when the connective is $\vee$ or $\langle a\rangle$. When the position is $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\sigma Z.\Psi)$, the next two positions are known: the following one is $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),Z)$ and the one after is $((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Psi)$. Neither player is responsible for the former two moves.

5.4.1 Winning Conditions for Players V and R

We next present the winning conditions for both players in Figure 5.4.

First of all, consider finite length plays. Player R wins if a false configuration is reached: the evaluated formula Φ_n is ff, or position $((T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k),Z)$ is reached where Z is free in Φ_0 and $(T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k) \notin \mathsf{V}(Z)$, or V is supposed

Winning conditions for player R:

1. The play so far is $((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)\ldots((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)$ and
   - $\Phi_n = ff$ or
   - $\Phi_n = Z$ and Z is free in Φ_0 and $(T_i^1,\ldots,T_j^m,\ldots,T_l^k) \notin \mathsf{V}(Z)$ or
   - $\Phi_n = \langle a\rangle_m \Psi$ for some $a \in A$, $m \in \{1,\ldots,k\}$ but $T_j^m \overset{a}{\not\to}$.
2. The play $((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)\ldots((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)\ldots$ has infinite length. There is a unique variable X, occurring infinitely often and subsuming (see Definition 2.9.4) all other variables occurring infinitely often, and this variable identifies a least fixed point formula (i.e. $\mu X.\Psi$).

Winning conditions for player V:

1. The play so far is $((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)\ldots((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)$ and
   - $\Phi_n = tt$ or
   - $\Phi_n = Z$ and Z is free in Φ_0 and Φ_0 and $(T_i^1,\ldots,T_j^m,\ldots,T_l^k) \in \mathsf{V}(Z)$ or
   - $\Phi_n = [a]_m \Psi$ for some $a \in A$, $m \in \{1,\ldots,k\}$ but $T_j^m \overset{a}{\not\to}$.
2. The play $((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)\ldots((T_i^1,\ldots,T_j^m,\ldots,T_l^k),\Phi_n)\ldots$ has infinite length. There is a unique variable X, occurring infinitely often and subsuming all other variables occurring infinitely often, and this variable identifies a greatest fixed point formula (i.e. $\nu X.\Psi$).

Figure 5.4: Winning conditions for players R and V

to move, but such a move is impossible. The rules for V are dual to the ones for R. V wins if a true configuration is reached: the evaluated formula Φ_n is tt, or position $((T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k),Z)$ is reached where Z is free in Φ_0 and $(T_i^1,\ldots,T_{j+1}^m,\ldots,T_l^k) \in \mathsf{V}(Z)$, or R has to move, but such a move is impossible.

Second, consider infinite length plays. The winner in such games depends on the outermost fixed point subformula that gets unfolded infinitely often: whenever it is a least fixed point formula, the refuter wins; dually, whenever this is a greatest fixed point formula, the verifier wins.

5.4.2 Sample Incremental Hyperproperty Checking Games

To illustrate IHP checking games, we present two examples. The correctness theorems, guaranteeing the results presented, can be found in Section 5.4.3.

Example 5.4.2. *Consider the IHP NI' from Chapter 4, which is the greatest fixed point of an NI'-simulation. NI' can be given in $\mathcal{L}^2_\mu$ as:*

$$NI' \mathrel{\widehat{=}} \nu X.\ \bigwedge_{a\in A_v} [a]_1\langle a\rangle_2 X \wedge \bigwedge_{a\in A_c} [a]_1 X.$$

Let view $V_0 = (A_v, A_n, A_c)$, where $A_v = \{l_1, l_2\}$, $A_n = \emptyset$ and $A_c = \{h\}$. The definition of NI' then becomes:

$$NI'_{V_0} \mathrel{\widehat{=}} \nu X.[l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X.$$

Consider the tree T, given in Figure 5.5: it is the infinite tree containing $(l_1 h l_2)^\omega$, $(l_1 l_2)^\omega$ and all streams resulting from some interleaving of $l_1 h l_2$ and $l_1 l_2$. The traces of tree T can be given by the omega regular expression $(l_1 h l_2 \mid l_1 l_2)^\omega$

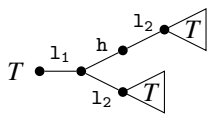
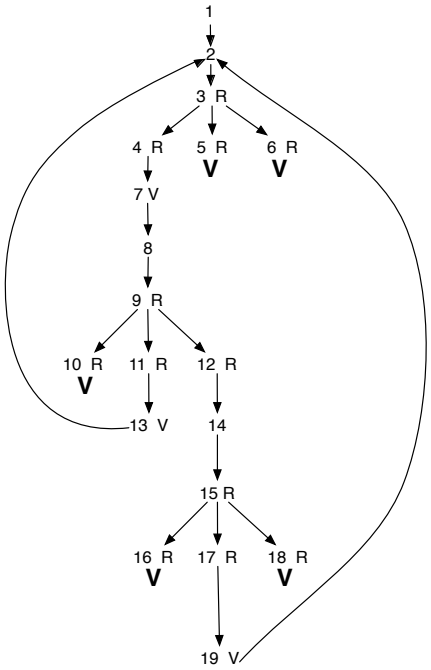


Figure 5.5: Tree T

The game $HG_{\mathbf{V}}((T,T), NI')$ has the positions given in Figure 5.6. Arrows from a position indicate the possible subsequent positions. Certain positions are labeled, indicating which player has to make the next move. Final positions are also labeled below the node, signifying the player who wins the game at the respective positions. For instance, position 5 is labeled **V** because it is a V-win. Informally, a play of the game can be seen as a sequence of positions, starting from the root and ending at a leaf of the game graph or looping forever. At position 16, V wins the game by the third part of rule 1 for player V (in Figure 5.4): in essence, R has a turn, but is unable to move. The situation is similar at positions $5, 6, 10$ and 18: V wins the game as R has a turn, but is unable to move. Positions 13 and 19 are again V wins, but for a different reason: as these positions result in infinite plays and the variable X identifies a greatest fixed point (see rule 2 for V in Figure 5.4). Thus, it is easy to see that V wins all plays of the game $HG_{\mathbf{V}}((T,T), NI')$. Hence, we may conclude that $(T,T) \models NI'$. By Corollary 5.3.7, we also have that $NI(T)$.

The previous example used only a greatest fixed point operator. Incremental hyperproperty definitions are not always that simple. In the following example, we show the game for a definition having alternation of least and greatest fixed point quantifiers, needed to express a notion of fairness.



$1 : ((T,T), \nu X.[l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X)$
$2 : ((T,T), X)$
$3 : ((T,T), [l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X)$
$4 : ((T,T), [l_1]_1\langle l_1\rangle_2 X)$
$5 : ((T,T), [l_2]_1\langle l_2\rangle_2 X)$
$6 : ((T,T), [h]_1 X)$
$7 : ((T_{l_1},T), \langle l_1\rangle_2 X)$
$8 : ((T_{l_1},T_{l_1}), X)$
$9 : ((T_{l_1},T_{l_1}), [l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X)$
$10 : ((T_{l_1},T_{l_1}), [l_1]_1\langle l_1\rangle_2 X)$
$11 : ((T_{l_1},T_{l_1}), [l_2]_1\langle l_2\rangle_2 X)$
$12 : ((T_{l_1},T_{l_1}), [h]_1 X)$
$13 : ((T,T_{l_1}), \langle l_2\rangle_2 X)$
$14 : ((T_{l_1 h},T_{l_1}), X)$
$15 : ((T_{l_1 h},T_{l_1}), [l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X)$
$16 : ((T_{l_1 h},T_{l_1}), [l_1]_1\langle l_1\rangle_2 X)$
$17 : ((T_{l_1 h},T_{l_1}), [l_2]_1\langle l_2\rangle_2 X)$
$18 : ((T_{l_1 h},T_{l_1}), [h]_1 X)$
$19 : ((T,T_{l_1}), \langle l_2\rangle_2 X)$

Figure 5.6: The game $HG_{\mathsf{V}}((T,T), NI')$

Example 5.4.3. *Let view $V_0 = (A_v, A_n, A_c)$, where $A_v = \{l\}$, $A_n = \{\tau\}$ and $A_c = \{h\}$. Consider our coinductive version of hyperproperty* noninference *NF' from Chapter 4, given in $\mathcal{L}^2_\mu$ as follows:*

$$NF' \mathrel{\widehat{=}} \nu X.\, O_1 \leftrightarrow O_2 \wedge \bigwedge_{a\in A_v} [a]_1 \mu Z.(\langle a\rangle_2 X \vee \langle\tau\rangle_2 Z) \wedge \bigwedge_{a\in A_c} [a]_1 X.$$

This definition does not have a fairness problem: in fact the fairness constraint is

encoded in the definition itself. The definition guarantees for any pair of systems satisfying it, that whenever player R can make a *visible a* move in the first system, player V can eventually get a turn to respond with the same move in the second system.

The definition of NI' with respect to view V_0 is:

$$NF'_{V_0} \mathrel{\widehat{=}} \nu X.\, O_1 \leftrightarrow O_2 \wedge ([l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)) \wedge [h]_1 X.$$

Now consider the concrete system T, presented in Figure 5.7.

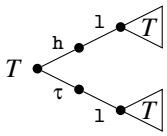


Figure 5.7: Tree T

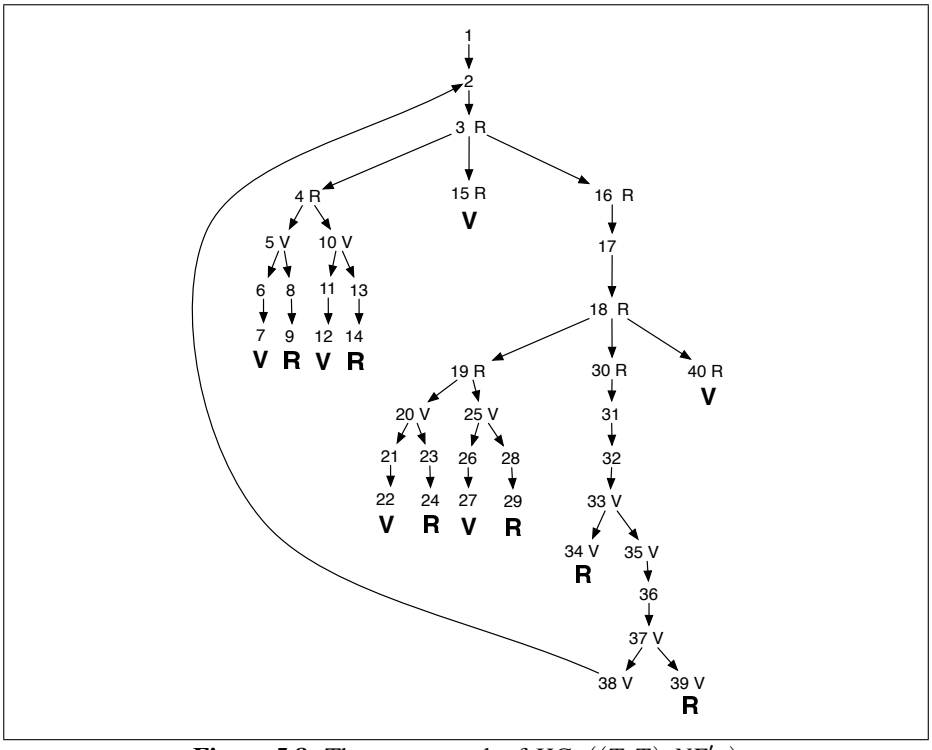


Figure 5.8: The game graph of $HG_{\mathsf{V}}((T,T), NF'_{V_0})$

$1 : ((T,T), \nu X.\ O_1 \leftrightarrow O_2 \wedge \wedge [l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z) \wedge [h]_1 X)$
$2 : ((T,T), X)$
$3 : ((T,T), O_1 \leftrightarrow O_2 \wedge \wedge [l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z) \wedge [h]_1 X)$
$4 : ((T,T), (\neg O_1 \vee O_2) \wedge (\neg O_2 \vee O_1))$
$5 : ((T,T), \neg O_1 \vee O_2)$
$6 : ((T,T), \neg O_1)$
$7 : ((T,T), tt)$
$8 : ((T,T), O_2)$
$9 : ((T,T), ff)$
$10 : ((T,T), \neg O_2 \vee O_1)$
$11 : ((T,T), \neg O_2)$
$12 : ((T,T), tt)$
$13 : ((T,T), O_1)$
$14 : ((T,T), ff)$
$15 : ((T,T), [l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z))$
$16 : ((T,T), [h]_1 X)$
$17 : ((T_h,T), X)$
$18 : ((T_h,T), O_1 \leftrightarrow O_2 \wedge [l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z) \wedge [h]_1 X)$
$19 : ((T_h,T), (\neg O_1 \vee O_2) \wedge (\neg O_2 \vee O_1))$
$20 : ((T_h,T), \neg O_1 \vee O_2)$
$21 : ((T,T), \neg O_1)$
$22 : ((T,T), tt)$
$23 : ((T,T), O_2)$
$24 : ((T,T), ff)$
$25 : ((T_h,T), \neg O_2 \vee O_1)$
$26 : ((T,T), \neg O_2)$
$27 : ((T,T), tt)$
$28 : ((T,T), O_1)$
$29 : ((T,T), ff)$
$30 : ((T_h,T), [l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z))$
$31 : ((T,T), \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z))$
$32 : ((T,T), Z)$
$33 : ((T,T), \langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)$
$34 : ((T,T), \langle l \rangle_2 X)$
$35 : ((T,T), \langle \tau \rangle_2 Z)$
$36 : ((T,T_\tau), Z)$
$37 : ((T,T_\tau), \langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)$
$38 : ((T,T_\tau), \langle l \rangle_2 X)$
$39 : ((T,T_\tau), \langle \tau \rangle_2 Z)$
$40 : ((T_h,T), [h]_1 X)$

Figure 5.9: Positions of the game $HG_{\mathsf{V}}((T,T), NF')$

The game graph of $HG_{\mathsf{V}}((T,T), NF'_{V_0})$ is presented in Figure 5.8 and the game positions in Figure 5.9. For this game, as is obvious from Figure 5.8, we have that some plays are won by V and some by R. For instance, V wins configurations 7, 12, 22 and 27 — these are all true configurations. Dually, the false configurations 9, 14, 24 and 29 are won by R. Position 15 is won by V, because it is R's turn but she cannot move, position 40 is similar. Positions 34 and 39 are dual: it is V's turn but she cannot move, hence R wins. Observing the game graph in Figure 5.8, it is straightforward to see that player V has a winning strategy given in the following table:

Position	5	10	20	25	33	37
Choice	6	11	21	26	35	38

5.4.3 Correctness of Hyperproperty Checking Games

Next, we will argue that the incremental hyperproperty checking games are correct. That is, formula Φ satisfied at state $(T^1,\ldots,T^m,\ldots,T^k)$ is equivalent to player V having a history-free winning strategy for the game $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

Dually, formula Φ not satisfied at state $(T^1,\ldots,T^m,\ldots,T^k)$ is equivalent to player R having a history-free winning strategy for the game $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

Theorem 5.4.4. *The following equivalences are valid:*

1. $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$ *iff player V has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.
2. $(T^1,\ldots,T^m,\ldots,T^k) \not\models_{\mathsf{V}} \Phi$ *iff player R has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

Proof. Essentially by lifting Stirling's proof for property checking games in the modal mu-calculus to IHP checking games in $\mathcal{L}^k_\mu$. The details of the proof are available in Appendix C. □

Corollary 5.4.5. *Incremental hyperproperty checking games are determined.*

Proof. Follows directly from Theorem 5.4.4. □

As a result of Theorem 5.4.4 and Corollary 5.4.5, we know that it is enough to reason about the existence of a history-free winning strategy for V to verify (or refute) an incremental hyperproperty expressed in $\mathcal{L}^k_\mu$. We can actually use tools to find such a strategy, by converting the IHP checking game into a parity game — a two player infinite game with perfect information [82]. This should not come as a surprise, as it is known that the solving of a parity game has equivalent complexity to the model checking problem for the modal mu-calculus [80]. This method of conversion is similar to the one used by Stirling [82] to convert property checking games into parity games and will be presented in the next section. The conversion is important as results and tools developed for parity games may be reused for solving IHP checking games. On the other hand, it does not make IHP checking games obsolete, because they are sinificantly more intuitive to reason about.

5.5 From Incremental Hyperproperty Checking Games to Parity Games

In this section, we show how to convert IHP checking games into parity games. The exposition is based on Stirling's conversion of his property checking games into parity games [82]. The proof that the conversion is correct follows directly from the analysis of the conversion process.

5.5.1 Parity Games

We start by recalling some definitions from Section 2.10. A game of *perfect information* is a game in which players move alternately and each player is aware of all previous moves in the game. A *parity game* is a game of perfect information, played by two players on a directed graph $G = (N, \rightarrow, L)$. N is a finite set of vertices from the set of natural numbers $\mathbb{N}$, $\rightarrow \subseteq N \times N$ is a binary relation on the vertices and $L : N \rightarrow \{V, R\}$ is a labeling function, assigning elements from the set of players $\{V, R\}$ to vertices. Note that the elements of N are the positions of the game and function L assigns labels to the vertices, essentially signifying which player is supposed to move at the position in question. We write $j \rightarrow k$ instead of $(j,k) \in \rightarrow$. All plays in a parity game have infinite length. A play always starts with a token on the least vertex. The winner of a play is determined by the label of the least vertex i occurring infinitely often. The notion of a history-free strategy for a parity game is standard (see Section 2.10). A player P's *strategy* in a parity game is a set of rules that prescribe what the next move j s.t. $i \rightarrow j$ should be at each position i, for which player P has a turn. A history is *winning* if P wins any parity game that conforms to the strategy.

5.5.2 Conversion of Incremental Hyperproperty Checking Games into Parity Games

As mentioned above, every play in a parity game is infinite. In practice, this is not problematic for the conversion. The solution is simple: make finite plays of the IHP checking games infinite by introducing a loop in the final position.

We next detail the conversion of IHP checking games into parity games. Recall that for any tree T, $SubT(T)$ is the set of subtrees of T. Let $HG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ be some IHP checking game. Let $E_1 = (T_1^1, \ldots, T_1^k)$, where $T_1^1 = T^1$, $\ldots$, $T_1^k = T^k$, be the starting position. Let $E_1, \ldots, E_m$ be the list of all possible k-tuples of trees reachable (independent of the formula Φ) from the initial k-tuple $(T_1^1, \ldots, T_1^k)$. Let Φ be a normal formula. Let $Z_1, \ldots, Z_k$ be a list of the bound variables in Φ. Game $HG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ will be converted into parity game $PG_{\mathsf{V}}((T^1, \ldots, T^k), \Phi)$ using the following steps:

1. Create a list of all subformulae in Φ in decreasing order of size, except that all Z_i are inserted into special places. The list is constructed as follows. Let $\Phi_1, \ldots, \Phi_l$ be a list of the formulae in $Sub(\Phi) \setminus \{Z_1, \ldots, Z_k\}$ in decreasing size, starting with $\Phi_1 = \Phi$. Extend the list by inserting each Z_i after its respective fixed point formula. For instance, the list may look as follows: $\Phi_1, \ldots, \sigma_i Z_i.\Psi_i, Z_i, \ldots, \Phi_l$. The final list of formulae is $\Phi_1, \ldots, \Phi_n$. All positions in the IHP checking game

$HG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ can be given as follows:

$$(E_1,\Phi_1),\ldots,(E_m,\Phi_1),(E_1,\Phi_2),\ldots,(E_1,\Phi_n),\ldots,(E_m,\Phi_n).$$

2. Specify the set N. The set N of the parity game $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ will be $\{1,\ldots,m\times n\}$. Any position (E_j,Φ_k) of the IHP checking game has a corresponding vertex i (in the parity game) given as follows: $i = m\times(k-1)+j$.

3. Specify the relation $\rightarrow$ and the labeling function L as follows: for any vertex i, we define its label and edges by case analysis of the respective to i position (F,Ψ), specified in Figure 5.10.

4. Remove all positions not reachable from (E,Φ), where $E = E_1 = (T_1^1,\ldots,T_1^k)$.

We have described the parity game $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$. The next step is to show that the winner of $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ and $HG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ are always the same. Observe that the least vertex i, appearing infinitely often in game $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ (and determining the winner), corresponds to one of the following positions:

- (F,Z) where Z is free in Φ
- (F,tt)
- (F,ff)
- $(F,[a]_k\Psi)$ where $F \overset{a}{\not\rightarrow}_k$
- $(F,\langle a\rangle_k\Psi)$ where $F \overset{a}{\not\rightarrow}_k$
- (F,Z_j)

In the first five cases the winner of the parity game is clearly the same as in the IHP checking game, as specified in Figure 5.4. In the sixth case we have (F,Z_j). It is possible to have another vertex, say j', which occurs infinitely often and corresponds to another position (F',Z_l). As i is the least vertex that occurs infinitely often, (F',Z_l) appears later than (F,Z_j) in the ordering. Hence, there are two options: the two positions coincide or the formula corresponding to Z_l is strictly smaller than the one corresponding to Z_j. In the former case, the winner is the same as in the IHP checking game. In the latter case, it has to be that Z_j subsumes Z_l. Thus, we may conclude that, in both possible cases, if $\nu Z_j.\Psi_j$ is in $Sub(\Phi)$, then player V wins the game. Otherwise, $\mu Z_j.\Psi_j$ is in $Sub(\Phi)$ and thus player R wins. This is guaranteed to be in accordance with the outcome of the corresponding IHP checking game.

Case analysis of position (F, Ψ):

- If $\Psi = Z$, where Z is free in Φ and $F \in \bigvee(Z)$, then $L(i) = V$ and $i \to i$.
- If $\Psi = Z$, where Z is free in Φ and $F \notin \bigvee(Z)$, then $L(i) = R$ and $i \to i$.
- If $\Psi = tt$, then $L(i) = V$ and $i \to i$.
- If $\Psi = ff$, then $L(i) = R$ and $i \to i$.
- If $\Psi = \Psi_1 \wedge \Psi_2$, then $L(i) = R$ and there exist edges $i \to j_1$, $i \to j_2$ s.t. j_1 corresponds to (F, Ψ_1) and j_2 corresponds to (F, Ψ_2).
- If $\Psi = \Psi_1 \vee \Psi_2$, then $L(i) = V$ and there exist edges $i \to j_1$, $i \to j_2$ s.t. j_1 corresponds to (F, Ψ_1) and j_2 corresponds to (F, Ψ_2).
- If $\Psi = [a]_k \Psi'$ and $F \overset{a}{\not\to}_k$, then $L(i) = V$ and there exists an edge $i \to i$.
- If $\Psi = [a]_k \Psi'$ and $F \overset{a}{\to}_k F'$, then $L(i) = R$ and there exists an edge $i \to j$ s.t. j corresponds to (F', Ψ').
- If $\Psi = \langle a \rangle_k \Psi'$ and $F \overset{a}{\not\to}_k$, then $L(i) = R$ and there exists an edge $i \to i$.
- If $\Psi = \langle a \rangle_k \Psi'$ and $F \overset{a}{\to}_k F'$, then $L(i) = V$ and there exists an edge $i \to j$ s.t. j corresponds to (F', Ψ').
- If $\Psi = \nu Z_j.\Psi_j$, then $L(i) = V$. Moreover, there is an edge $i \to j'$ s.t. j' corresponds to (F, Z_j).
- If $\Psi = \mu Z_j.\Psi_j$, then $L(i) = R$. Moreover, there is an edge $i \to j'$ s.t. j' corresponds to (F, Z_j).
- If $\Psi = Z_j$ and $\nu Z_j.\Psi_j \in Sub(\Phi)$, then $L(i) = V$. Moreover, there is an edge $i \to j'$ s.t. j' corresponds to (F, Ψ_j).
- If $\Psi = Z_j$ and $\mu Z_j.\Psi_j \in Sub(\Phi)$, then $L(i) = R$. Moreover, there is an edge $i \to j'$ s.t. j' corresponds to (F, Ψ_j).

Figure 5.10: Case analysis of positions

Proposition 5.5.1. *There exists a winning strategy for player $P \in \{V, R\}$ in an IHP checking game $HG_{\bigvee}((T^1, \ldots, T^k), \Phi)$ iff there exists a winning strategy for player P in the corresponding parity game $PG_{\bigvee}((T^1, \ldots, T^k), \Phi)$.*

Proof. Straightforward based on the above observations. The main proof idea is that the winner of both games is always the same. □

Example 5.5.2. *Consider the game $HG_{\forall}((T^1,\ldots,T^k),NI')$ from Example 5.4.2. To illustrate the conversion process, we shall construct the resultant parity game $PG_{\forall}((T^1,\ldots,T^k),NI')$ here. First, create the list of states $E_1,\ldots,E_9$:*

$$(T,T),(T,T_{l_1}),(T,T_{l_1h}),(T_{l_1},T),(T_{l_1},T_{l_1}),(T_{l_1},T_{l_1h}),(T_{l_1h},T),(T_{l_1h},T_{l_1}),(T_{l_1h},T_{l_1h}).$$

Then create the list of subformulae in NI' denoted $\Phi_1,\ldots,\Phi_8$ in decreasing order of size:

$$\nu X.[l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X,\ X,\ [l_1]_1\langle l_1\rangle_2 X \wedge [l_2]_1\langle l_2\rangle_2 X \wedge [h]_1 X,\ [l_1]_1\langle l_1\rangle_2 X,$$

$$[l_2]_1\langle l_2\rangle_2 X,\ \langle l_1\rangle_2 X,\ \langle l_2\rangle_2 X,\ [h]_1 X.$$

The number of possible positions of the property checking game is 72 ($n \times m$), but not all of them are reachable. The positions are given in the following table:

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
Φ_1	1	2	3	4	5	6	7	8	9
Φ_2	10	11	12	13	14	15	16	17	18
Φ_3	19	20	21	22	23	24	25	26	27
Φ_4	28	29	30	31	32	33	34	35	36
Φ_5	37	38	39	40	41	42	43	44	45
Φ_6	46	47	48	49	50	51	52	53	54
Φ_7	55	56	57	58	59	60	61	62	63
Φ_8	64	65	66	67	68	69	70	71	72

The resulting parity game is depicted in Figure 5.11. Positions of the verifier V are in a box, positions of the refuter R in a circle. V wins all possible plays of this game and thus has a trivial winning strategy: no matter what V does, she will win the game. We next explain why this is the case. At each of positions 32, 35, 37, 64, 71, V wins because they are labeled V and each of them is the least i occurring infinitely often in the plays. In all other infinite paths (the one passing through positions 41 and 44), the least i occurring infinitely often is 10 and it is labelled V, i.e. $L(10) = V$.

This small example demonstrates that creating and analyzing game graphs similar to the one from Figure 5.11 is tedious and error prone. To address these problems, in Chapter 6 we explore the use of game-based model checking tools. It should be noted, however, that even small systems are bound to produce large state spaces. For instance any system with state space S and formula Φ (interpreted on a k-tuple of systems) will have a state space of the resulting parity game at most $|S|^k \times |\Phi|$. Thus finding useful, well-behaving fragments of the logic and/or reusing existing (or finding new) efficient algorithms for model checking is an important area for future work.

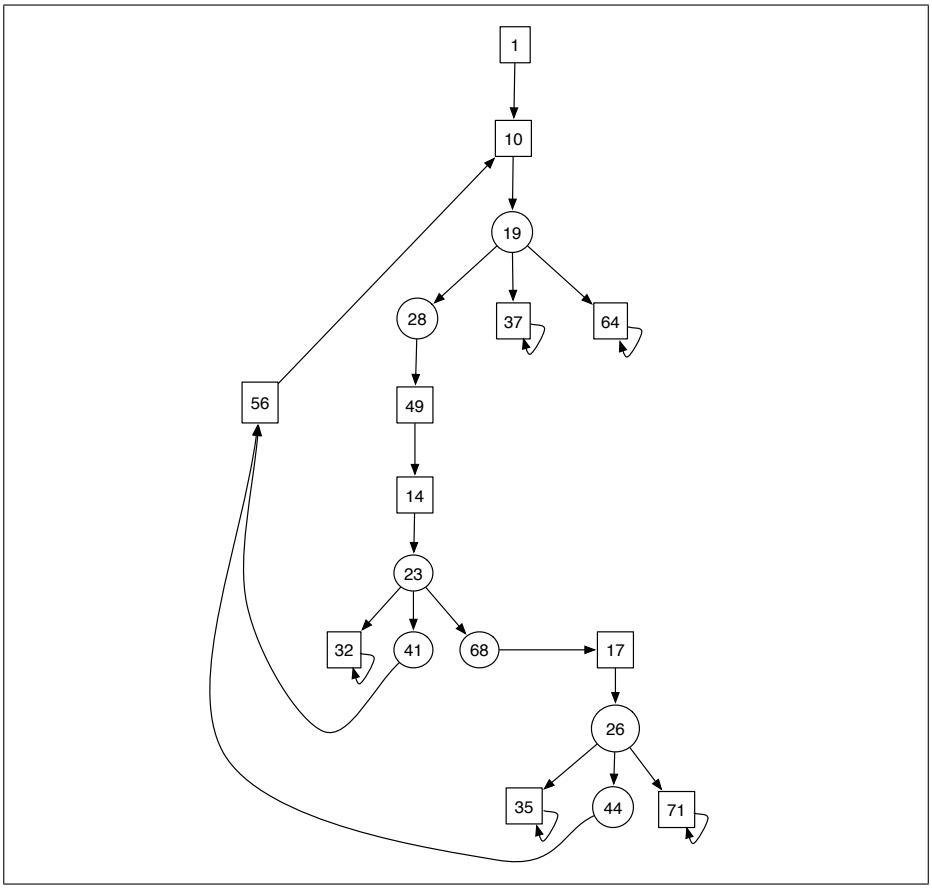


Figure 5.11: Parity game corresponding to $HG_V((T,T),NI')$.

5.6 A New Logic for Incremental Hyperproperties

This section presents the logic IL_μ^k, which is a fragment of the polyadic modal mu-calculus $\mathcal{L}_\mu^k$. Naturally, this logic is less expressive than the full $\mathcal{L}_\mu^k$, but we find it expressive enough for a large class of useful, security-relevant, incremental hyperproperties.

Formulae in IL_μ^k have the following syntax:

$$\Psi ::= \nu Z.\Phi \qquad \Phi ::= tt \mid ff \mid Z \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [a]_i\Phi \mid \langle a\rangle_i\Phi \mid \mu Z.\Phi.$$

The games we propose for IL_μ^k are essentially the same as the ones for $\mathcal{L}_\mu^k$. It is

noteworthy that the maximal alternation depth of any formula in IL_μ^k is 2, which results in lower complexity than $\mathcal{L}_\mu^k$. This is reflected in the following result about the complexity of model checking for IL_μ^k, based on Theorem 2.9.1 :

Corollary 5.6.1. *There exists an algorithm for deciding* $\overline{\mathcal{T}} \models \Phi$*, running in time* $O(|\Phi|^2(|S_1|\dots|S_k|)|\mathcal{T}_1|\dots|\mathcal{T}_k|)$*, where* Φ *is closed and* $\overline{\mathcal{T}}$ *a k-tuple of finite transition systems with state spaces* $S_1,\dots,S_k$*.*

The logic allows (a restricted variant of) coinductive/inductive definitions, which are reminiscent of the idea that any hyperproperty is the intersection of a safety hyperproperty and a liveness hyperproperty [18]: the latter are discussed in more detail in Chapter 2, but essentially they can be seen as generalizations of safety and liveness properties respectively. In essence, hypersafety can be expressed as a greatest fixed point formula and hyperliveness as a least fixed point formula. $\mathcal{L}_\mu^k$ and IL_μ^k seem suitable for expressing these, based on the examples we considered. The characterization of hypersafety and hyperliveness in IL_μ^k (or in $\mathcal{L}_\mu^k$) is an interesting thread for future work.

5.6.1 Sample Incremental Hyperproperties in IL_μ^k

As an illustration of the logic, we first present an incremental hyperproperty class from Chapter 3. The class is SIH^2 (see its definition in Section 3.3.2) and it contains incremental versions of notions of noninterference, such as *NI* from Chapter 3. SIH^2 is defined in IL_μ^k as follows:

$$\mathsf{SIH}^2 \mathrel{\widehat{=}} \nu X.\ \bigwedge_{a\in A\wedge p(a)} [a]_1\langle f(a)\rangle_2 X \wedge \bigwedge_{a\in A\wedge\neg p(a)} [a]_1 X.$$

In an attempt to show that the logic can express a large class of other, useful, security-relevant IHPs, we show that the coinductive unwinding relations from Chapter 4 can be expressed in IL_μ^k. As a result, we argue that a large number of security-relevant IHPs can be expressed in the logic, because those can be seen as conjunction of the unwinding relations.

The first relation we present here is osc_V-simulation. First, recall that the union of all osc_V-simulations, namely *osc_V-similarity*, can be given (in the proper extension of IL, allowing alternation of the fixed point operators) as follows:

$$\mathbf{gfp}\, Q(x,y)\ .\ o(x)\leftrightarrow o(y)\ \wedge\ \forall a\in A\setminus A_c.\ (x\xrightarrow{a} x_a \rightarrow$$

$$\exists \sigma\in(A\setminus A_c)^*.(y\xrightarrow{\sigma} y_\sigma \wedge a\approx_{A_v}\sigma \wedge Q(x_a,y_\sigma))).$$

Using the previous definition, two trees (systems) S and T are osc_V-similar iff

$$[\mathbf{gfp}\, Q(x,y)\ .\ o(x) \leftrightarrow o(y)\ \wedge\ \forall a \in A \setminus A_c.\ (test_a(x) \rightarrow \exists \sigma \in (A \setminus A_c)^*.(test^*{}_\sigma(y) \wedge a \approx_{A_v} \sigma \wedge Q(x_a, y_\sigma)))](S,T).$$

In IL^k_μ, *osc_V-similarity* can be represented as follows:

$$osc_V \mathrel{\widehat{=}} \nu X.\ O_1 \leftrightarrow O_2 \wedge \bigwedge_{a \in A \setminus A_c} [a]_1 \mu Z.(\langle a \rangle_2 X \vee \langle A_n \rangle_2 Z).$$

The fact that two trees (systems) S, T are osc_V-similar can be expressed in IL^k_μ as follows:

$$(S,T) \models \nu X.\ O_1 \leftrightarrow O_2 \wedge \bigwedge_{a \in A \setminus A_c} [a]_1 \mu Z.(\langle a \rangle_2 X \vee \langle A_n \rangle_2 Z).$$

lrf_V-similarity, the union of all lrf_V-simulations, is defined coinductively in IL as:

$$\mathbf{gfp}\, Q(x,y)\ .\ o(x) \leftrightarrow o(y)\ \wedge\ \forall a \in A_c.\ (test_a(x) \rightarrow Q(x_a, y)).$$

In IL^k_μ, *lrf_V-similarity* can be represented as follows:

$$lrf_V \mathrel{\widehat{=}} \nu X.\ O_1 \leftrightarrow O_2 \wedge [A_c]_1 X.$$

Finally, *lrb_V-similarity*, the union of all lrb_V-simulations, is defined coinductively in IL as:

$$\mathbf{gfp}\, Q(x,y)\ .\ o(x) \leftrightarrow o(y)\ \wedge\ \forall a \in A_c.\ test_a(y) \wedge Q(x, y_a).$$

In IL^k_μ, *lrb_V-similarity* can be presented as follows:

$$lrb_V \mathrel{\widehat{=}} \nu X.\ O_1 \leftrightarrow O_2 \wedge [A_c]_2 tt \wedge [A_c]_2 X.$$

5.7 Discussion and Related Work

To the best of our knowledge, we are the first to propose and explore the idea of representing hyperproperties as games. More concretely, this chapter attempts to bridge the gap between the problem of generic verification of incremental hyperproperties and the extensive work on model checking games for fixed point logics [80, 81, 82]. Thus, it shares features with work in both the security and game theory communities.

It is well-known that the modal mu-calculus and other tree logics, such as CTL and CTL*, are not expressive enough for hyperproperties. Intuitively, the reason is that they cannot express specifications relating several paths in a tree, yet the original notion of hyperproperties is based on the existence of such relations (see [18, 63]). Alur et al. [7] present a proof that secrecy, defined as uncertainty whether a particular property is true or not (also expressible in our framework), is not expressible in the modal mu-calculus. In a subsequent paper, Alur et al. [6] introduce two new logics, enriching CTL and the modal mu-calculus with path equivalencies, making them expressive enough for secrecy. The major difference to our work is in the generality: definitions in our framework are not limited to secrecy and not necessarily based on equivalence relations. Indeed, many H'-simulations are not equivalence relations.

Standard fixed point logics and model checking techniques may be used for specific, determinism-based definitions of noninterference. For instance, Huisman and Blondeel [39] give a modal mu-calculus characterization of two such notions of information flow: observational determinism and eager trace equivalence [71]. Their characterization is based on self-composition [9, 20] of the transition system induced by the program of interest. The major difference to our work is that our IHPs and IHP checking games are more general. There is no restriction to deterministic systems and equivalence relations. Thus we can handle a much larger class of system and policy specifications. Due to the nature of our approach and the fact that IHPs need not be defined on k copies of the same system, but on k different systems instead, our framework may be used for reasoning about hyperproperty-preserving refinement. Exploring this idea is left for future work. In addition, H'-simulations go beyond the state-of-the-art in giving a generic method to reason about (at least) a number of interesting security-relevant liveness hyperproperties. Characterizing the class of hyperproperties that benefit from H'-simulations is an important direction for future work.

In recent work, D'Souza et al. [23] have proposed an automata-theoretic technique for model checking the possibilistic information flow hyperproperties from Mantel's framework [53] on finite state systems, to model check Mantel's BSPs. This is a nice theoretical result, supporting our thesis that incremental hyperproperties are amenable to model checking. The proposed model checking approach is based on

deciding set inclusion on regular languages. Their approach is not directly applicable to hyperproperties, because the presence of infinite traces means that the languages under consideration are not regular. Nevertheless, it seems worthwhile to adapt these techniques to deal with omega-regular languages or arbitrary streams.

Hyperproperties generalize properties and it would be intuitively appealing if checking hyperproperties would generalize checking properties. This is what we have shown in this work for incremental hyperproperties. First, we have proposed to express incremental hyperproperties in $\mathcal{L}_\mu^k$ — a logic that generalizes the modal mu-calculus, in which (regular) properties are typically expressed. The games we have presented here can be seen as a generalization of the model checking games for the modal mu-calculus [82]. Hence, our work is related to the abundant work on model checking games for fixed point logics [80, 81, 82, for example]. The crucial contribution is that we are the first to propose games for $\mathcal{L}_\mu^k$ and its fragment $I\mathcal{L}_\mu^k$ with an interesting, security-relevant application.

5.8 Summary

This chapter proposes a clear and feasible verification methodology for incremental hyperproperties. The major contribution is a novel characterization of the satisfaction relation between a system and an incremental hyperproperty in terms of playing a game. To that end, we first introduce H'-simulation games, an intuitive approach of directly converting H'-simulation relations into games. Further, we are the first to propose the more general and sophisticated incremental hyperproperty checking games for the polyadic modal mu-calculus $\mathcal{L}_\mu^k$. We have also argued that $\mathcal{L}_\mu^k$ is expressive enough for the known incremental hyperproperties and for some fairness constraints. The proposed approach is feasible, because checking of incremental hyperproperties expressible in $\mathcal{L}_\mu^k$ is decidable and can be done in polynomial time. A fragment of $\mathcal{L}_\mu^k$ (called $I\mathcal{L}_\mu^k$), which has lower complexity and might be sufficient for a large class of incremental hyperproperties, has also been proposed. However, we also envision the possibility of devising more efficient tableau algorithms [12] for $I\mathcal{L}_\mu^k$. This is a promising direction for future work.

In the next chapter we consider the use of tools to automate the game-based verification of incremental hyperproperties.

Chapter 6

Model Checking Incremental Hyperproperties via Games

In this chapter we explore the significance of incremental hyperproperty checking games and their corresponding parity games for practical model checking of incremental hyperproperties. In particular, we present three different approaches for model checking incremental hyperproperties and investigate some of their advantages and disadvantages. The fact that these approaches can be automated is illustrated by using particular off-the-shelf tools for model checking sample hyperproperties. Two of the approaches are game-based and we show that this is advantageous, as winning strategies can give useful intuition about the system-policy interaction. Most importantly, this work demonstrates how to reuse algorithms and tools for practical model checking mu-calculus games for model checking incremental hyperproperties.

6.1 Synopsis

The chapter starts by illustrating the problem of model checking systems via parity games and thus motivates the need of model checking tools for IHP checking games. Then the tools used in this chapter, namely mCRL2 [34], PGSolver [29] and MLSolver [30], are briefly introduced. This is followed by the presentation of three different approaches for model checking incremental hyperproperties.

The first approach discussed in this work was originally proposed by Andersen himself to model check his polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [8]. The approach is based on a reduction of the problem of model checking a formula ϕ in $\mathcal{L}_\mu^k$ on a system S to model

checking a modified formula (called $prod(\phi)$) in the ordinary modal mu-calculus [43] on a modified version of the original system (called $prod(S)$). This means that one can use off-the-shelf model checking tools for verification of incremental hyperproperties, as long as there is a convenient way to create the modified version of the original system (we present one such method using mCRL2).

The second approach is based on converting an IHP checking game into a parity game (as shown in Section 5.5) and solving the parity game using an off-the-shelf tool such as PGSolver. The third approach offers more automation and is again based on model checking of a modified system with respect to a modified specification formula. The advantage of game-based model checking is that the algorithms generate winning strategies [82]. This is beneficial, as one may use such a winning strategy to get intuition about the interaction between a system and a policy and, even more importantly, to get a precise reason as to what goes wrong and why a formula does not hold on a system. In case the formula holds, insight as to why this is the case on the concrete system can be given too. Using such techniques and visualizations of games may result in tools with intuitive debugging functionality.

To further explore the advantages of model checking via games we propose two different graph views, namely an *extended game graph view* and a *tree view*, that can be used for better visualization of the interactions between a policy and a system. An extended game graph view enhances the game graph with the winning strategy of the winner from the initial node. Such a view also presents at each winning position the winner and the complete position itself. A tree view is a list of the different positions in the history of the play so far. It turns out that tree views can be visualized in the same way as the H'-simulation games from Section 5.3, which are an intuitive representation of the game. In addition, we present a known idea for a strategy-based interactive tool [83] (here based on our extended game graph and tree views) that could be useful in our setting for understanding the interaction of systems and security policies. We note that the main difference between the two game-based approaches explored in this chapter is the degree of automation: the third one has the potential for being fully automatic.

6.2 Illustration and Motivation

In Chapter 5 we introduced model checking incremental hyperproperties via IHP games. However, reasoning about them "by hand" seems to be tedious and error-prone (see, for instance, Examples 5.4.2 and 5.4.3). We also showed that IHP checking games can be straightforwardly converted into parity games. In this section we argue that as systems get larger, reasoning about parity games "by hand" is also tedious and

error prone. This motivates the use of model checking tools, as explored in further sections.

Example 6.2.1 (From an IHP Checking Game to a Parity Game)**.** *Let view V_0 be given as: $A_v = \{l\}$, $A_n = \{\tau\}$ and $A_c = \{h\}$. Recall our coinductive version of hyperproperty* noninference *NF' from Chapter 5:*

$$NF' \mathrel{\widehat{=}} \nu X.\, O_1 \leftrightarrow O_2 \wedge \bigwedge_{a \in A_v} [a]_1 \mu Z.(\langle a \rangle_2 X \vee \langle A_n \rangle_2 Z) \wedge \bigwedge_{a \in A_c} [a]_1 X.$$

We will consider a termination insensitive definition here. Implicitly taking into account the view V_0, the definition becomes:

$$NF'_{ti} \mathrel{\widehat{=}} \nu X.\, ([l]_1 \mu Z.(\langle l \rangle_2 X \vee \langle \tau \rangle_2 Z)) \wedge [h]_1 X.$$

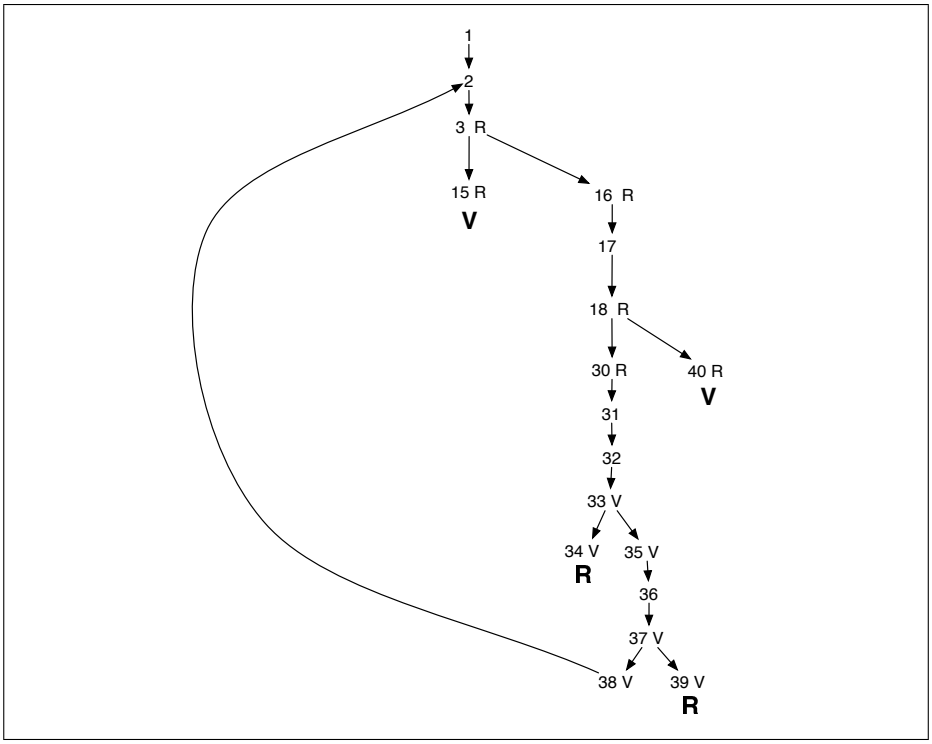


Figure 6.1: The game graph of $HG_{\mathsf{V}}((T,T), NF'_{ti})$

Recall system T from Figure 5.7, also specified by the omega regular expression $(hl \mid \tau l)^{\omega}$. Before constructing the game graph of parity game $PG_{\mathsf{V}}((T,T), NF'_{ti})$,

we consider what is already known for this graph. First, the game graph of $PG_{\mathsf{V}}((T,T),NF'_{ti})$ *can be obtained from the game graph of the respective IHP checking game* $HG_{\mathsf{V}}((T,T),NF'_{V_0})$*, presented in Figure 6.1.*

Note that the only difference in the new version of $HG_{\mathsf{V}}((T,T),NF'_{V_0})$ *as compared to the one in Figure 5.8 results from the fact that we have to remove the subtrees starting at vertices* 4 *and* 19 *that correspond to formula* $O_1 \leftrightarrow O_2$*.*

It is relatively straightforward to convert the game graph of $HG_{\mathsf{V}}((T,T),NF'_{ti})$ *into parity game* $PG_{\mathsf{V}}((T,T),NF'_{ti})$*, by making the final positions loop and labeling or relabeling positions. In particular, nodes corresponding to greatest fixed point formulae are labeled as V nodes, and dually, nodes corresponding to least fixed point formulae are labeled as F nodes. In addition each final node is (re)labeled with the winner of the* IHP *checking game and gets a loop to itself.*

In this particular case, to convert the game graph from Figure 6.1 to the game graph of its equivalent parity game, the following additional labels are needed. Positions 1 *and* 2 *should be labelled V as they correspond to a greatest fixed point formula and its respective bound variable, similarly* 17 *should be labelled V. Positions* 31 *and* 32 *should be labelled R as they correspond to a least fixed point formula and the respective bound variable. Node* 15 *should have label V, whereas* 34 *and* 39 *should be labelled R, because these are the respective winners of the final positions. In addition, positions* $28, 64, 75$ *and* 85 *get self-loops. It is also possible to convert back the parity game into an IHP checking game isomorphic to the original one. Note that any parity game, constructed as detailed in Section 5.5.2, can be converted into an IHP checking game.*

Example 6.2.2 (Creating a Parity Game from Scratch)**.** *We next use the procedure for creating the parity game* $PG_{\mathsf{V}}((T,T),NF'_{ti})$ *from scratch, as detailed in Section 5.5.2. First, create the list of states* $E_1,\ldots,E_9$*:*

$$(T,T),\ (T,T_h),\ (T,T_\tau),(T_h,T),\ (T_h,T_\tau),\ (T_h,T_h),\ (T_\tau,T),\ (T_\tau,T_h),\ (T_\tau,T_\tau).$$

Then create a list of the subformulae in NF'_{ti}*, denoted* $\Phi_1,\ldots,\Phi_{10}$*, in decreasing order of size:*

$$\nu X.\ ([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)\wedge[h]_1 X),\ X,\ [l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)\wedge[h]_1 X,$$

$$[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z),\ \mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z),\ Z,\ \langle l\rangle_2 X \vee \langle\tau\rangle_2 Z,\ \langle l\rangle_2 X,\ \langle\tau\rangle_2 Z,\ [h]_1 X.$$

There are 90 *possible positions of the property checking game, but of course not all of them are reachable. The positions are given in the following table:*

	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9
Φ_1	1	2	3	4	5	6	7	8	9
Φ_2	10	11	12	13	14	15	16	17	18
Φ_3	19	20	21	22	23	24	25	26	27
Φ_4	28	29	30	31	32	33	34	35	36
Φ_5	37	38	39	40	41	42	43	44	45
Φ_6	46	47	48	49	50	51	52	53	54
Φ_7	55	56	57	58	59	60	61	62	63
Φ_8	64	65	66	67	68	69	70	71	72
Φ_9	73	74	75	76	77	78	79	80	81
Φ_{10}	82	83	84	85	86	87	88	89	90

The resulting parity game $PG_{\mathsf{V}}((T,T),NF'_{ti})$ is depicted in Figure 6.2. Positions of the verifier V are surrounded by a box, positions of the refuter R by a circle. V has a history-free winning strategy for the game $PG_{\mathsf{V}}((T,T),NF'_{ti})$. The history-free winning strategy is given in the following table:

Position	1	10	28	13	85	55	57	73	66
Choice	10	19	28	22	85	73	66	48	10

Hence, we may conclude that $(T,T) \models NF^{ti}_{V_0}$. This claim, as well as the precise winning strategy are verified in Example 6.5.1 with the help of PGSolver.

The previous examples are based on a very simple system, having only 3 states. Nevertheless, it should be clear that as the number of states gets larger, to create and reason about parity games by hand becomes cumbersome and error-prone. Fortunately, we can reuse parity game solvers for the verification of incremental hyperproperties. We next present and later investigate the use of three different tools for model checking IHPs.

6.3 The Tools

This section gives a brief introduction to the tools used in the chapter.

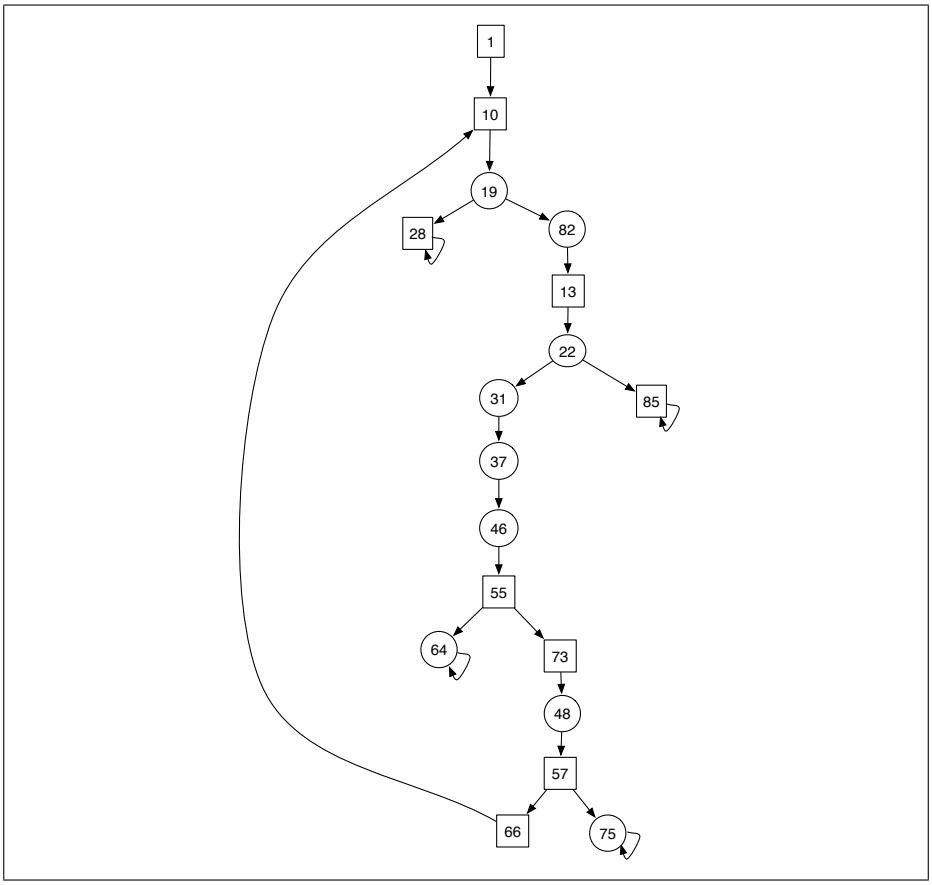


Figure 6.2: Parity game $PG_{\forall}((T,T),NF'_{ti})$.

6.3.1 PGSolver

PGSolver is a toolset consisting of tools for generating, manipulating and solving parity games [29]. It implements most of the known algorithms for solving parity games, such as the recursive algorithm by Zielonka [88], the strategy improvement algorithm by Vöge and Jurdziński [85], the optimal strategy improvement method by Schewe [78], etc., as well as heuristics for making the solving faster. Depending on the algorithm chosen, PGSolver solves the games either locally with respect to a concrete vertex or globally with respect to the whole game graph. The typical use of PGSolver is illustrated in Section 6.5.1.

6.3.2 MLSolver

MLSolver is a tool targeting the satisfiability and validity problems for modal fixed point logics [30]. In essence, the tool reduces the satisfiability and validity problem to that of solving a parity game. The respective parity game is then passed on to PGSolver. More details about our use of the tool can be seen in Section 6.5.2.

6.3.3 mCRL2

mCRL2 is a a formal specification language together with a toolset [34]. The typical use of mCRL2 is for modelling and verification of concurrent systems and protocols. We will use the language for specifying transformed systems. In addition, some of the tools in the toolset will be used for performing the system transformation and the actual model checking. More details about the use of mCRL2 are presented in Section 6.4.1.

6.4 Traditional Model Checking of $\mathcal{L}_\mu^k$

This section presents the use of traditional model checking techniques for incremental hyperproperties. Andersen proposes a model checking approach for the polyadic modal mu-calculus $\mathcal{L}_\mu^k$ [8]. The approach is a reduction of the problem of model checking $\mathcal{L}_\mu^k$ to model checking the ordinary modal mu-calculus $\mathcal{L}_\mu$ on a product of the original system. One tool that can be used to implement this model checking approach is mCRL2. This is illustrated in Section 6.4.1, but we first present Andersen's approach.

Definition 6.4.1. Given an n-ary tuple of transition systems $(T_1,\ldots,T_n)$, define the product $prod(\overline{T})$ of these to be the labelled transition system $(S,\rightarrow,i)$, where S is the state space given as $S \mathrel{\widehat{=}} S_1 \times \ldots \times S_n$, $\rightarrow$ is the transition relation and i the tuple of the start states. Relation $\rightarrow \subseteq S \times (A \times \mathbb{N}) \times S$ is defined as follows:

$$(s_1,\ldots,s_n) \xrightarrow{a,i} (s'_1,\ldots,s'_n) \text{ iff}$$

$$s_i \xrightarrow{a} s'_i \text{ and } \forall j.(1 \leq j \leq n \wedge j \neq i) \rightarrow s_j = s'_j.$$

Next, define $prod(\Phi)$ as the homomorphic map on formulae in $\mathcal{L}_\mu^k$ such that $prod(\langle a\rangle_i \Phi) = \langle (a,i)\rangle prod(\Phi)$. Note that instead of $\langle (a,i)\rangle\Phi$ we typically write $\langle a_i\rangle\Phi$. It is clear that $prod(\overline{T})$ is a single system (vs. tuple of systems) and $prod(\Phi)$ is defined over such systems.

Theorem 6.4.2 (Reduction of $\mathcal{L}_\mu^k$ to $\mathcal{L}_\mu$ [8])**.** *Consider an n-tuple of transition systems $\overline{T}$ and an $\mathcal{L}_\mu^k$ formula Φ, as well as the respective $prod(\overline{T})$ and $prod(\Phi)$ as defined above. Then the following equivalence is valid:*

$$\overline{T} \models \Phi \textit{ iff } prod(\overline{T}) \models prod(\Phi).$$

This result is important, as it suggests the use of standard model checking techniques for verification of IHPs expressed in the polyadic modal mu-calculus $\mathcal{L}_\mu^k$.

6.4.1 Model Checking with mCRL2

In this section we illustrate the use of mCRL2 for model checking incremental hyperproperties. Let us start with a formula Φ in $\mathcal{L}_\mu^k$ expressing an IHP defined on a tuple of systems $\overline{T}$. We perform the following transformations (in accordance with Andersen's approach presented above) to reduce the problem of model checking $\overline{T}$ with respect to Φ to an ordinary model checking problem of the product $prod(\overline{T})$ with respect to the formula $prod(\Phi)$.

1. Make a product of systems $prod(\overline{T})$. In practice, we typically create systems $T_1, T_2, \ldots, T_k$ that are the same as T, except that each action has as subscript $1, 2, \ldots, k$, indicating which copy it belongs to. Then, we build the product $prod(T_1, T_2, \ldots, T_k)$. Tools such as mCRL2 can be used to facilitate the creation of such a product; to achieve this, we use parallel composition but disable the simultaneous occurrence of multiple actions, as they are not interesting for the product. The result is an interleaving parallel composition of $T_1, T_2, \ldots, T_k$. In essence, we specify that only multi-actions consisting of a single action are allowed in the product. In mCRL2, this is done using the *allow* operator. For the hyperproperties considered in this work the binary product $prod(T_1, T_2)$, as presented in Figure 6.4, is enough.
2. Convert formula Φ into $prod(\Phi)$, where Φ works on a k-tuple of systems $(\overline{T})$ and $prod(\Phi)$ on one system, namely on the product $prod(T_1, T_2, \ldots, T_k)$. Each action in the formula $prod(\Phi)$ is given a subscript, signifying to which system it belongs.

Then, we can use mCRL2's model checking tools to find out whether $prod(T,T) \models prod(\Phi)$, and hence by Theorem 6.4.2 whether $(T,T) \models \Phi$ holds.

Example 6.4.3. *Consider again the game $HG_V((T,T), NF'_{ti})$, where T is specified by the omega regular expression $(hl \mid \tau l)^\omega$ (visualized in Figure 6.3).*

The first step is creating $prod(T_1, T_2)$. The transition system of the product is specified in mCRL2 as follows:

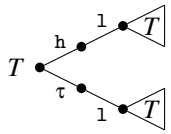


Figure 6.3: System T

```
act
    h1,h2,l1,l2,τ1,τ2;
proc
    T1 = h1.l1.T1 + τ1.l1.T1;
    T2 = h2.l2.T2 + τ2.l2.T2;
    S = T1||T2;
init allow({h1,h2,l1,l2,τ1,τ2},S);
```

The `act` *keyword is followed by a list of allowed actions, the* `proc` *keyword is followed by a list of process definitions. The* `init` *keyword gives the initial process. Here, this is process S, which is the interleaving parallel composition of the original system copies T_1 and T_2. The* `allow` *keyword is used to specify the actions that should be synchronized (i.e. happen in parallel). Our specification effectively says that no actions should be synchronized. This specification results in the transition system presented in Figure 6.4.*

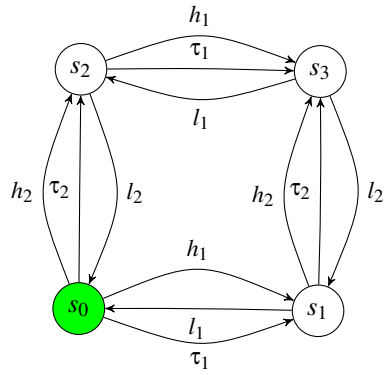


Figure 6.4: The product $prod(T_1, T_2)$ produced by mCRL2

The second step is converting the IHP specification. In this case, the definition of NF'_{ti}, namely

$$NF'_{ti} \mathrel{\widehat{=}} \nu X.\,([l]_1 \mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z) \wedge [h]_1 X),$$

is converted into $prod(NF'_{ti})$*, given as*

$$prod(NF'_{ti}) = \nu X.\ ([l_1]\mu Z.(\langle l_2\rangle X \vee \langle \tau_2\rangle Z) \wedge [h_1]X).$$

Using mCRL2 in a standard way, we model check the product $prod(T_1,T_2)$ *from Figure 6.4 with respect to formula* $prod(NF'_{ti})$*. The latter is encoded as follows:*

```
nu X . (([l1]mu Z . (⟨l2⟩X || ⟨τ2⟩Z)) && ([h1]X))
```

The following output is produced by the pbs2bool *tool of the mCRL2 toolset:*

```
Retrieving pbes_equations from equation system...
Computing a BES from the PBES....
Solving a BES with 5 equations.
Solve equations of rank 2.
Solve equations of rank 1.
The solution for the initial variable of the pbes is true.
```

Hence, we may conclude that $prod(T_1,T_2) \models prod(NF'_{ti})$ *and thus* $(T,T) \models NF'_{ti}$*.*

6.5 Model Checking via Games

We next propose two model checking approaches for IHPs via games. The first is based on the combination of IHP checking games and the parity game solver PGSolver [29]. The second is based on the use of several tools and has the advantage that it can be fully automated. Both approaches are based on creating and solving the appropriate parity game, eventually using PGSolver [29].

6.5.1 Model Checking IHP Checking Games

This approach describes how to model check IHP checking games. Start with some IHP checking game $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$.

1. Convert $HG_{\mathsf{V}}((T^1,\ldots,T^m,\ldots,T^k),\Phi)$ into the equivalent min-parity game $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$. This conversion is straightforward, as explained in Section 5.5.
2. Use PGSolver to convert $PG_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$ to the equivalent max-parity game $PG^{max}_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$. Note that PGSolver solves max-parity games, but also offers a converter between the two.
3. Use PGSolver to solve the parity game $PG^{max}_{\mathsf{V}}((T^1,\ldots,T^k),\Phi)$. To know whether $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$ holds or does not hold, it is enough to solve

the game *locally* for the starting node of the game graph. The tool reports which player has a history-free winning strategy, as well as the strategy itself, which is positional (see Section 2.10). If player V has such a strategy, then it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \models_{\mathsf{V}} \Phi$. Dually, if player R has a history-free winning strategy, it has to be that $(T^1,\ldots,T^m,\ldots,T^k) \not\models_{\mathsf{V}} \Phi$. However, the strategy itself is also very important, as it provides intuition why the policy holds or does not hold and it can be used for the purpose of understanding and modifying the system-policy interaction. This will be further investigated in Section 6.6.

Example 6.5.1. *We next illustrate the use of PGSolver for solving IHP checking games. If we start with game $HG_{\mathsf{V}}((T,T),NF'_{ti})$ in Figure 6.1, we can easily convert it into a parity game $PG_{\mathsf{V}}((T,T),NF'_{ti})$. This was demonstrated in Example 6.2.1. The parity game $PG_{\mathsf{V}}((T,T),NF'_{ti})$ (see Figure 6.2) can be specified in PGSolver as follows:*

```
parity 17;
0 2 0 1 "1";
1 2 0 2 "10";
2 1 1 3,4 "19";
3 2 0 3 "28";
4 1 1 5 "82";
5 2 0 6 "13";
6 1 1 7,8 "22";
7 1 1 9 "31";
8 2 0 8 "85";
9 1 1 10 "37";
10 1 1 11 "46";
11 2 0 12,13 "55";
12 1 1 12 "64";
13 2 0 14 "73";
14 1 1 15 "48";
15 2 0 16,17 "57";
16 2 0 1 "66";
17 1 1 17 "75";
```

In this parity game specification, the first line is optional and gives the highest identifier, used for optimization purposes. Each further line specifies a vertex by giving it an identity number, its parity, its owner, the vertices that are successors and finally an optional, symbolic name of the vertex [29].

Running PGSolver to solve the parity game globally produces the following output:

```
 Player 0 wins from nodes:
  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15, 16}
with strategy
  [0->1,1->2,3->3,5->6,8->8,11->13,13->14,15->16,16->1]
Player 1 wins from nodes:
  {12, 17}
with strategy
  [12->12,17->17]
```

Thus PGSolver verifies that the strategy for game $PG_{\mathsf{V}}((T,T),NF'_{ti})$, presented in Example 6.2.2, is correct. Because V has a history-free winning strategy from the start node (node 0), it follows that $(T,T) \models NF'_{ti}$.

This solution actually tells us two facts: there exists a positional winning strategy from the start node for player V and what that strategy is. The strategy itself is important as it provides a witness as to why a hyperproperty holds or does not hold, as well as some intuition.

Alternatively, if we were only interested in the validity of the checked formula, we could use PGSolver as follows: perform local model checking to determine whether player V has a winning strategy from the start node 0. Using any of the local model checking algorithms implemented by PGSolver gives the following output:

```
Parsing ..................... 0.00 sec
Chosen local solver 'modelchecker' .. 0.00 sec
Visited 18 nodes.
Winner of initial node is player 0
```

We can also use PGSolver to produce a graph of the parity game $PG_{\mathsf{V}}((T,T),NF'_{ti})$, together with the solution. This can be seen in Figure 6.5. Each position is either a rhombus or a rectangle. The positions at which player V has to move (i.e. all i s.t. $L(i) = V$) are rhombi and the positions at which player R has a turn are rectangles. The positions from which player V has a winning strategy are shown in green (gray) and the ones from which R can win are in red (black). Inside each rhombus/rectangle, the number before the column is the symbolic name of the position and the number after is the parity. The strategies are not explicitly given in the figure, however it is possible to deduce them. Green positions have a winning strategy for V and the winning strategy is to select a green transition going out of the respective position. Dually, it is possible to read a winning strategy (if it exists) for R by choosing red transitions at rectangles. The winning positions for V are in set W_V, specified as

$$W_V = \{1,10,19,28,82,13,22,31,85,37,46,55,73,48,57,66\},$$

given in green (gray) in Figure 6.5, and the winning positions for R are in set

$$W_R = \{64,75\},$$

given in red (black) in Figure 6.5. As expected, $W_V \cup W_R$ gives the set of all vertices of the game, because the game is positionally determined (see Theorem 2.10.6).

This example demonstrated that we can use PGSolver to solve the parity game corresponding to an IHP checking game. However, it would be better if we could convert the transition system and formula automatically into a parity game, and subsequently solve that game. To that end, we next show how to use a combination of MLSolver and mCRL2.

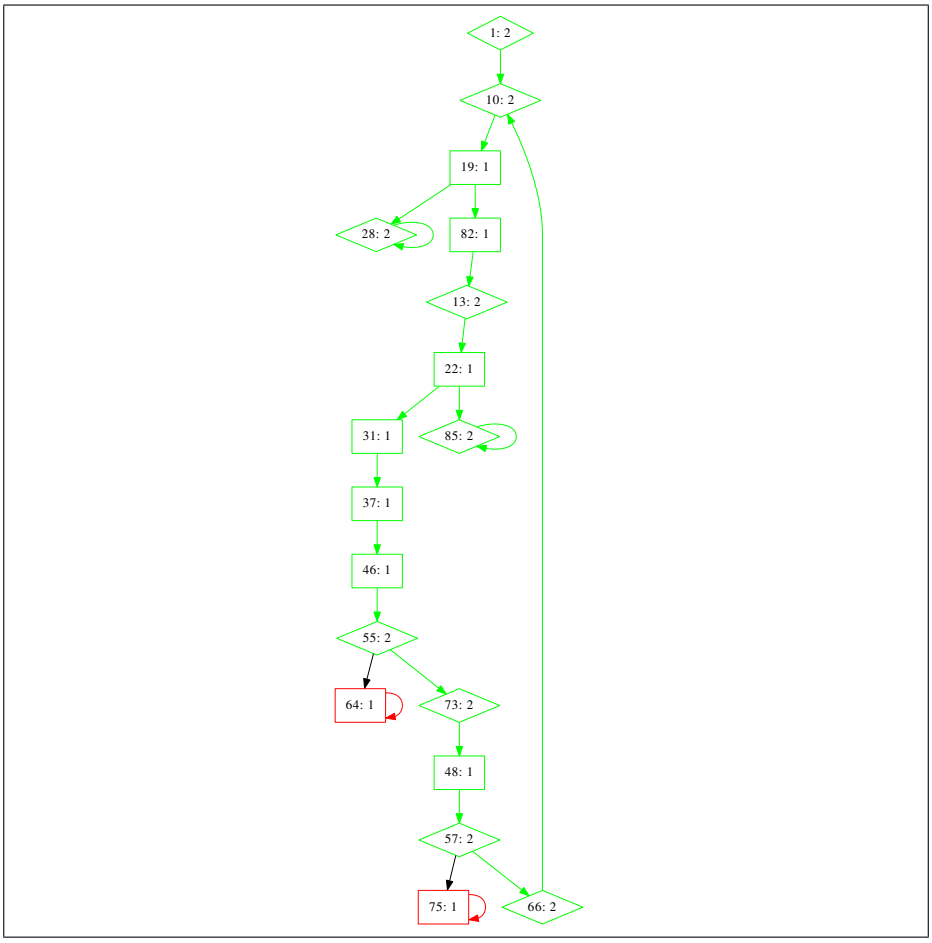


Figure 6.5: Parity game $PG_V((T,T),NF'_{ti})$ with winning positions for V in green (gray) and R in red (black).

6.5.2 Model Checking without Going through IHP Checking Games

An alternative approach that constructs the parity game automatically and does not rely on an IHP checking game is presented next. The needed steps, given systems $(T^1,\ldots,T^m,\ldots,T^k)$ and formula Φ, are:

1. Make the product of the systems denoted $prod(T^1,\ldots,T^m,\ldots,T^k)$, as outlined in Section 6.4. We were able to do this in mCRL2 using the parallel composition

operator (||) and disabling the simultaneous occurrence of multiple actions, as we are not interested in those for the product (see Section 6.5.1).

2. Convert the resulting system $prod(T^1,\ldots,T^m,\ldots,T^k)$ in a format recognizable by MLSolver.
3. Convert formula Φ to work on the product $prod(T^1,\ldots,T^m,\ldots,T^k)$. In essence, each action in the formula is given a subscript linking it with a particular transition system. The result is $prod(\Phi)$ (see Section 6.4).
4. Use MLSolver to create a parity game for system $prod(T^1,\ldots,T^m,\ldots,T^k)$ and formula $prod(\Phi)$.
5. Use PGSolver to solve the parity game resulting from step 4.

In principle, writing a tool for fully automating these steps is relatively easy. For the proof of concept model checking performed in this section, we only wrote a script performing step 2, as converting between formats of huge systems cannot be easily done by hand. The feasibility and to some extent scalability of the proposed approach are demonstrated by the following examples.

Example 6.5.2. *Let view V_0 be given as: $A_v = \{l\}$, $A_n = \{\tau\}$ and $A_c = \{h\}$. Consider system S presented in Figure 6.6.*

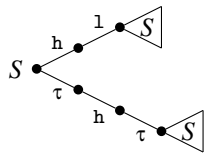


Figure 6.6: System S

We again use mCRL2 to build the product $prod(S_1,S_2)$, specified as follows:

```
act
    h1,h2,l1,l2,τ1,τ2;
proc
    T1 = h1.l1.T1 + τ1.h1.τ1.T1;
    T2 = h2.l2.T2 + τ2.h2.τ2.T2;
    S = T1||T2;
init allow({h1,h2,l1,l2,τ1,τ2},S);
```

The policy we want to check is again $prod(NF'_{ti})$, here in the proper format for MLSolver:

```
nu X . (([l1]mu Z . (<l2>X | <τ2>Z)) & ([h1]X)) .
```

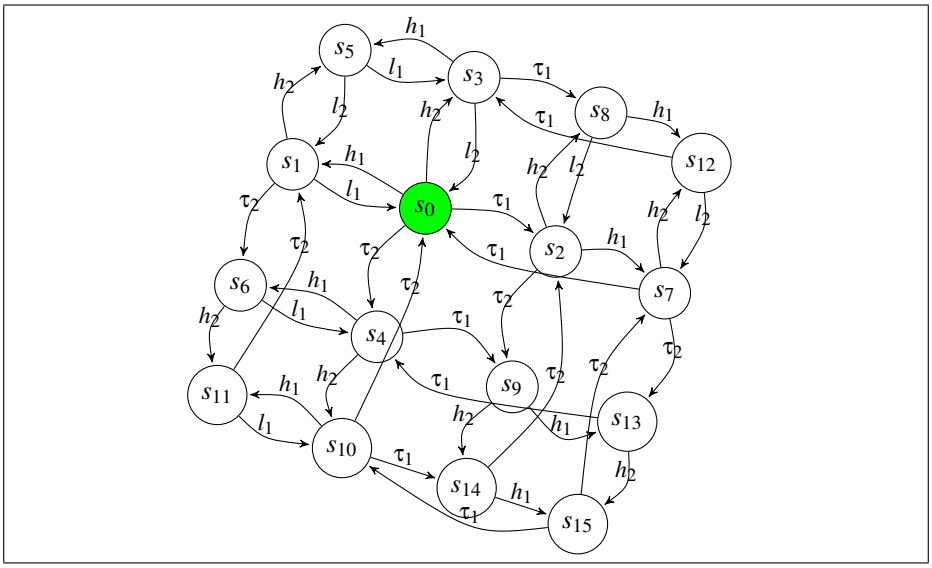


Figure 6.7: The product $prod(S_1, S_2)$

The resulting transition system of $prod(S_1, S_2)$ can be seen in Figure 6.7. The transition system is encoded in MLSolver as follows:

```
lts 16;
start 1;
1 h1:2, tau1:3, h2:4, tau2:5;
2 l1:1, h2:6, tau2:7;
3 h1:8, h2:9, tau2:10;
4 h1:6, tau1:9, l2:1;
5 h1:7, tau1:10, h2:11;
6 l1:4, l2:2;
7 l1:5, h2:12;
8 tau1:1, h2:13, tau2:14;
9 h1:13, l2:3;
10 h1:14, h2:15;
11 h1:12, tau1:15, tau2:1;
12 l1:11, tau2:2;
13 tau1:4, l2:8;
14 tau1:5, h2:16;
15 h1:16, tau2:3;
16 tau1:11, tau2:8;
```

The specification format is simple. The first line says that the transition system has 16 *states, the second line gives the start state* 1*. Each further line specifies a state and lists its successors together with the respective labels.*

The output (checking whether prod$(NF^{ti}_{V_0})$ holds for prod(S_1,S_2)) in MLSolver is:

```
Modelchecking Procedure For Labelled Modal Mu Calculus
Game has 15 states.
Calling solver pgsolver...
Finished solving: 0.00 sec
Transition system is no model of the formula!
```

Hence, we may conclude that $(S,S) \not\models NF'_{ti}$.

Instead of solving the game (MLSolver creates the parity game internally and passes it on to PGSolver by default), it is possible to display the parity game using MLSolver and the strategy using PGSolver. The respective parity game is:

```
parity 15;
start 0;
0 2 0 1;
1 0 1 2,3;
2 0 1 2;
3 0 1 4;
4 2 0 5;
5 0 1 6,7;
6 0 1 8;
7 0 1 7;
8 1 0 9;
9 0 0 10,11;
10 1 0 10;
11 0 0 12;
12 1 0 13;
13 0 0 14,15;
14 1 0 14;
15 1 0 15;
```

The strategy is given as:

```
Player 0 wins from nodes:
  {2, 7}
with strategy
  []
Player 1 wins from nodes:
  {0, 1, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15}
with strategy
  [1->3,3->4,5->6,6->8]
```

As player 1*, i.e. player R, wins from the start node* 0*, it has to be that $(S,S) \not\models NF'_{ti}$.*

Example 6.5.3. *We next consider a somewhat larger example specified in mCRL2 as follows:*

```
act
    h1,h2,l1,l2,τ;
proc
T = (h1.((h2.l1.l2.h1.l2.τ)+(l1.((h1.l1.((l2.l1.T)+(l1.T)))+(h2.l1.T)))))
    +(τ.((l1.τ.((τ.((l1.τ.((l2.τ.l1.T)+(l1.T)))+(τ.l1.T)))+(l2.τ.τ.l2.T)))
    +(τ.l2.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.l2.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.l1.T)))
    +(h2.((l2.h1.l1.h1.h1.h2.l2.h1.l1.h1.h1.h2.h2.l1.T)+(h1.l2.l2.h1.l1.T)))
    +(h1.h1.h2.h1.T)
    +(l2.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.τ.l1.τ.τ.τ.τ.τ.τ.τ.τ.l2.τ.τ.τ.τ.τ.l1.τ.l1.T);
init allow({h1,h2,l1,l2,τ},T);
```

Note that view V_0 is here given as: $A_v = \{l_1, l_2\}$, $A_n = \{\tau\}$ and $A_c = \{h_1, h_2\}$. This example is somewhat artificial as it did not come as a result of analyzing a concrete system. However, it is not difficult to imagine that it can be an abstraction of a possible system. The construction of the system also reveals a pattern that may be useful: we started by creating some traces, which were the "useful" behavior of the system, but would make the system insecure as there were no respective low equivalent modulo τ traces. Then we added behavior to create some uncertainty for the attacker, eventually making the system secure with respect to the policy. In principle, one may use the winning strategy for R to try to incrementally add more traces (uncertainty) to an insecure system so that in the end the policy would be satisfied.

The product $prod(T_1, T_2)$ is specified in mCRL2 as follows:

```
act
    h10,h20,l10,l20,τ0,h11,h21,l11,l21,τ1;
proc
T1 = (h10.((h20.l10.l20.h10.l20.τ0)+(l10.((h10.l10.((l20.l10.T1)+(l10.T1)))+(h20.l10.T1)))))
    +(τ0.((l10.τ0.((τ0.((l10.τ0.((l20.τ0.l10.T1)+(l10.T1)))+(τ0.l10.T1)))+(l20.τ0.τ0.l20.T)))
    +(τ0.l20.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.l20.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.l10.T1)))
    +(h20.((l20.h10.l10.h10.h10.h20.l20.h10.l10.h10.h10.h20.h20.l10.T1)+(h10.l20.l20.h10.l10.T1)))
    +(h10.h10.h20.h10.T1)
    +(l20.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.τ0.l10.τ0.τ0.τ0.τ0.τ0.τ0.τ0.l20.τ0.τ0.τ0.τ0.τ0.l10.τ0.l10.T1);
T2 = (h11.((h21.l11.l21.h11.l21.τ1)+(l11.((h11.l11.((l21.l11.T2)+(l11.T2)))+(h21.l11.T2)))))
    +(τ1.((l11.τ1.((τ1.((l11.τ1.((l21.τ1.l11.T2)+(l11.T2)))+(τ1.l11.T2)))+(l21.τ1.τ1.l21.T2)))
    +(τ1.l21.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.l21.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.l11.T2)))
    +(h21.((l21.h11.l11.h11.h11.h21.l21.h11.l11.h11.h11.h21.h21.l11.T2)+(h11.l21.l21.h11.l11.T2)))
    +(h11.h11.h21.h11.T2)
    +(l21.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.l11.τ1.τ1.τ1.τ1.τ1.τ1.τ1.τ1.l21.τ1.τ1.τ1.τ1.τ1.l11.τ1.l11.T2);
S = (T1||T2);
init allow({h10,h20,l10,l20,τ10,h11,h21,l11,l21,τ11},S);
```

Note that the two copies are T_1 and T_2, the actions have an extra digit signifying which copy they belong to (suffix 0 belongs T_1 and 1 to T_2). The product $prod(T_1, T_2)$ is constructed in mCRL2 and converted to the mCRL2 fsm *format, which is textual. The transition system of $prod(T_1, T_2)$ has the following statistics (using the mCRL2 tool* ltsstat*):*

```
Number of states:18415
Number of state labels:18415
Number of action labels:11
Number of transitions:44572
```

Then we run our script for converting the mCRL2 specification of the transition system prod(T_1,T_2) into the format acceptable by MLSolver. The specification is not presented here due to its size. The formula to be checked against is prod(NF'_{ti}) and is given as follows:

$$nuX.(([l_{10}]muZ.(\langle l_{11}\rangle X|\langle \tau_1\rangle Z))\&([l_{20}]muZ.(\langle l_{21}\rangle X|\langle \tau_1\rangle Z))\&[h_{10}]X\&[h_{20}]X).$$

The result of feeding in the specification and the formula in MLSolver is a parity game having 2463 nodes. The game is omitted here because it is too large. Then we use PGSolver to solve the game: the output produced by the tool solving the parity game locally is:

```
Parsing ..................... 0.01 sec
Chosen local solver 'modelchecker' .. 0.01 sec
Visited 940 nodes.
Winner of initial node is player 0
```

As a result we may conclude that $(T,T) \models NF'_{ti}$ (see Theorem 6.4.2). The winning strategies given by PGSolver when solving the game globally are too large to be displayed here. However, this example shows that it is possible to work with relatively large systems. Empirically exploring the limits on such systems, based on building a model generator given a formula in $\mathcal{L}^k_\mu$ as well as converting RIMP programs to sets of traces and model checking them is left for future work.

6.6 Advantages of Model Checking via Games

We have presented three different approaches to model checking incremental hyperproperties. Two of the approaches are game-based and this section presents some of the advantages of model checking via games in comparison with traditional approaches, such as the one presented in Section 6.4. Although game graphs similar to the ones presented in Figures 5.6 and 5.8 are useful for visualizing the respective games, there is more that can be done for understanding and analyzing IHP checking games. To demonstrate this, we introduce two new views of IHP checking games. These views are important to help us focus on interesting aspects of the game and are based on information that is calculated by the model checking algorithm.

We first present some notation needed for the formalization of the game graphs (of IHP checking games) and the new views. Let $\sigma = \{V,R\}$ denote the set of players. A

game graph can be formalized as the tuple $(\mathbb{V}, \rightarrow, L)$, where $\mathbb{V}$ is the set of positions (vertices), $\rightarrow \subseteq \mathbb{V} \times \mathbb{V}$ is a binary relation on vertices, and the partial function $L : \mathbb{V} \rightarrow \sigma$, when defined, denotes whose turn it is at a position. Note that this definition is slightly different than the one of games from Section 2.10, as the function L is partial. However, IHP checking games can be converted into parity games having labels on each node (and thus making L total), as seen in Section 5.5. Recall that the games we consider are positionally determined. Hence, the strategy for player σ is a partial function $f_\sigma : \mathbb{V}_\sigma \rightarrow \mathbb{V}$, where $v \in \mathbb{V}_\sigma$ iff $L(v) = \sigma$. Let Win_σ be the set of winning positions for player σ, as described in Figure 5.4. Note that technically Win_V and Win_R are not part of the game graph and need to be calculated by the model checking algorithm on the respective game graph. However, we sometimes included them in the visualizations of graphs for illustrative purposes (e.g. in Figures 5.6 and 5.9).

We next present the *extended game graph view*, which enhances the game graph with the winning strategy and at each winning position the complete position itself.

Definition 6.6.1. An *extended game graph* view is a 3-tuple (G, f_σ^G, Win^G), including the graph view G, the winning strategy f_σ^G, where $\sigma \in \{V, R\}$, for one of the players, namely the winner from the start node, and the set of winning positions for both players denoted $Win^G = Win_V^G \cup Win_R^G$.

A simple extended game graph view is presented in Figure 6.6.3. The graph additionally includes the progression of k-tuples of trees along the graph, which is one possible visualization of the tree view presented next.

The second view presented here is called the *tree view* of the game. A tree view is a finite list of the states (k-tuples of trees) visited in a play with stuttering states removed, starting at the root of the game tree and ending at the respective position. Note that this is essentially the projection of the history of a play to the states. Recall that $SubT(\overline{T})$ gives the set of k-tuples of subtrees of some k-tuple of trees $\overline{T}$.

Definition 6.6.2. A *tree view* for an IHP checking game $HG_\forall((T^1, \ldots, T^k), \Phi)$ at position $HG_\forall((T_l^1, \ldots, T_m^k), \Phi_n)$ is a list of states, starting with $(T^1, \ldots, T^k)$ and ending at $(T_l^1, \ldots, T_m^k)$, without repeating stuttering states.

As an example, the tree view at position 12 in Figure 6.6.3 is: $(T,T), (T_h,T), (T_{hl},T)$. The tree view is essentially a view showing the history of a game. The rules for this game come from the policy to be checked; the arena of the game (the k-tuple of trees) together with the current position and the rules determine which next moves are possible. Such games are instances of H'-simulation games (see Section 5.3) and were used to illustrate H'-simulations throughout this thesis (see Examples 3.5,3.6,4.3, etc.). We think that they capture the intuition behind H'-simulations well.

The ability of users to see and cross-reference both views introduced above and to play interactively in IHP checking games in the role of V can be useful for debugging

purposes: the fact that the system does not satisfy some policy (in $\mathcal{L}_\mu^k$) can be seen interactively as the inability of V to win the respective game [83]. An advantage of the views is that they are essentially computed during the model checking process. A tool such as PGSolver can come up with a winning strategy and the winning positions for the respective parity game automatically. Thus, the views can be constructed automatically with relatively minor modifications of existing tools, e.g. PGSolver. Such modifications would be very efficient as they would allow reuse of a large part of the code, data structures and utilities, however they require understanding the code of the respective tool. Alternatively, it is possible to construct a tool based on outputs produced by existing tools, however this would require a more substantial coding effort.

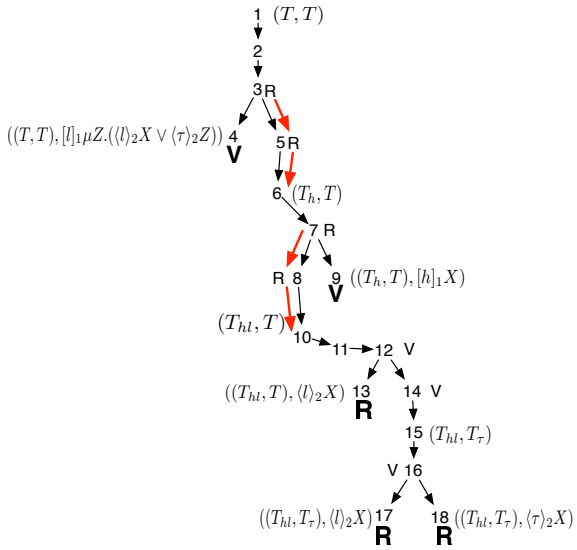
In order to better visualize the strategy-based evidence and show that it is useful to enhance the user's understanding (why a policy does not hold in cases when R has a winning strategy), we propose to combine the extended game graph view with the tree view. This visualization can be done by a specially constructed interactive tool, similar to the one proposed for property checking games [83]. The visualization starts by showing the part of the extended game graph view, for which no player is responsible (such as initial positions) in combination with the respective tree view. At each point in time there are several options. If the current vertex is labeled R, then V has no choice but to observe what the next position is. This position is determined by the winning strategy for R. The play goes into the new position, the extended view is changed appropriately and the tree view is changed when necessary. If the current vertex is labeled V, it is V's turn to choose a move and the tool presents the formulae to choose from. If the current vertex is not labeled, the game progresses automatically. At any time, both views are visualized to the user (for instance, see Figures 6.6.3 and 6.9) and the combined information helps V to make an informed choice. V is also able to backtrack and explore different plays.

Example 6.6.3. *To illustrate these ideas, consider the system T given by the omega-regular expression $(hlh \mid \tau h\tau)^\omega$. We want to check whether $(T,T) \models NF'_{ti}$, where NF'_{ti} is the termination insensitive version of NF' (restricted to some view (A_v, A_n, A_c) with $A_v = \{l\}, A_n = \{\tau\}$ and $A_c = \{h\}$), given as follows:*

$$NF'_{ti} \mathrel{\widehat{=}} \nu X.\, ([l]_1 \mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X.$$

The extended game graph and the respective positions are given in Figure 6.8. It is clear that player R has a winning strategy for the IHP *checking game $HG_\vee((T,T), NF'_{ti})$. The strategy is given as the red (grey) arrows.*

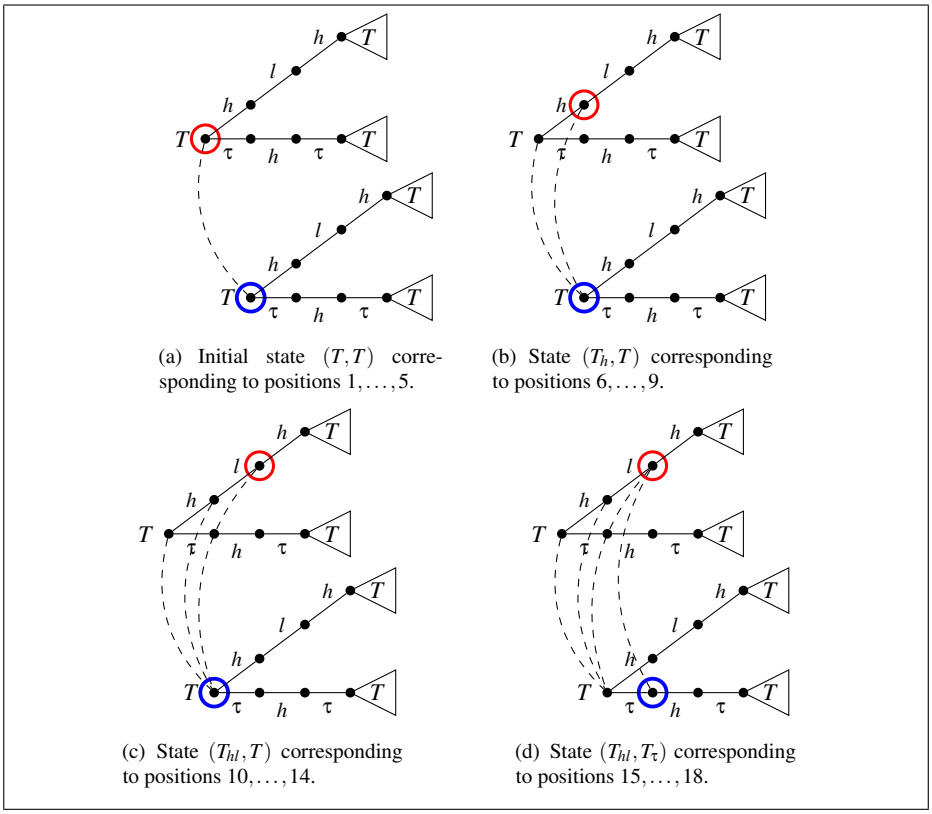
We next illustrate a visualization, building incrementally the views, that may be helpful for the user to understand why the policy NF'_{ti} is violated by system T. Figure 6.9



1 : $((T,T),\nu X.\ ([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
2 : $((T,T),X)$
3 : $((T,T),([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
4 : $((T,T),[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
5 : $((T,T),[h]_1 X)$
6 : $((T_h,T),X)$
7 : $((T_h,T),([l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)) \wedge [h]_1 X)$
8 : $((T_h,T),[l]_1\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
9 : $((T_h,T),[h]_1 X)$
10 : $((T_{hl},T),\mu Z.(\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z))$
11 : $((T_{hl},T),Z)$
12 : $((T_{hl},T),\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
13 : $((T_{hl},T),\langle l\rangle_2 X)$
14 : $((T_{hl},T),\langle\tau\rangle_2 Z)$
15 : $((T_{hl},T_\tau),Z)$
16 : $((T_{hl},T_\tau),\langle l\rangle_2 X \vee \langle\tau\rangle_2 Z)$
17 : $((T_{hl},T_\tau),\langle l\rangle_2 X)$
18 : $((T_{hl},T_\tau),\langle\tau\rangle_2 Z)$

Figure 6.8: The extended game graph view of $HG_{\mathsf{V}}((T,T),NF'_{ti})$

presents the positions (and their respective tree views) corresponding to the interesting plays of the game, namely the plays witnessing the winning strategy for R. The visualization of the tree view can be seen as a game in its own right: on the arena of two copies of T, player R moves in the first tree and has a red (light grey) token, player V moves in the second tree and has a blue (dark grey) token. The rules are



(a) Initial state (T,T) corresponding to positions $1,\ldots,5$.

(b) State (T_h,T) corresponding to positions $6,\ldots,9$.

(c) State (T_{hl},T) corresponding to positions $10,\ldots,14$.

(d) State (T_{hl},T_τ) corresponding to positions $15,\ldots,18$.

Figure 6.9: The tree views of game $HG_{\mathsf{V}}((T,T),NF'_{ti})$ at different positions/states

implicit and depend on NF'_{ti}.

Initially, player V (i.e. the user playing as player V) sees the extended graph until position 3 *and the tree view in Figure 6.9(a). The next two moves are automatic and determined by the winning strategy for R. The play is now at position* 6*, the extended view consists of positions* $1,\ldots,6$ *and the tree view is the one in Figure 6.9(b). At positions* 7 *and* 8 *player R follows her winning strategy. The extended game graph progresses and is built to position* 10*, the respective tree view is the one in Figure 6.9(c).*

Positions 10 *and* 11 *are not labeled so the play goes to position* 12*, which is the first possible choice for player V: she is now asked to choose whether to go to position* $((T_{hl},T),\langle l\rangle_2 X)$ *or* $((T_{hl},T),\langle\tau\rangle_2 X)$*. If V chooses the first option (position* 13*), she is informed that she loses the game. This can be seen in the tree view, as player V has to make an l-move in the second tree, but making such a move is impossible. The user*

can backtrack to position 12 *and choose to go to position* 14 *instead. V continues playing and now the tree view changes to the one in Figure 6.9(d). The play then goes on until position* 16*, where V has to choose between positions* $((T_{hl}, T_\tau), \langle l \rangle_2 X)$ *or* $((T_{hl}, T_\tau), \langle \tau \rangle_2 X)$*. Choosing either position, the user loses. Consulting the tree view in Figure 6.9(d), the user can see that neither the action l nor* τ *can be performed (in the second tree) at position* (T_{hl}, T_τ) *and this is the precise reason for losing. Interestingly, this is also the reason why the policy does not hold on the system. The two views have helped identify and visualize the problem. This shows that the views can be used to provide useful intuition as to what goes wrong in the interaction system-policy.*

The techniques presented here can be used to explore any IHP game in $\mathcal{L}_\mu^k$ on any finite-state system. The user can systematically explore different paths and strategies to play against player *R*. As shown above, this helps with understanding both the policy and system behavior, as well as their interaction and potential problems.

6.7 Relation to H'-simulation Games

Recall that H'-simulation games were introduced in Section 5.3 as games naturally corresponding to H'-simulation relations.The games were introduced to give intuition about and better visualize the H'-simulations. Interestingly, these are the same games that we also used in the visualization of our tree views. However, now the rules of the games have been formalized thanks to the introduction of IHP checking games and the corresponding parity games. This means that we have succeeded in capturing the intuition of H'-simulations, but also have a formal representation of the rules of the game. By introducing the two different views we can see the game itself in terms of changes of positions and cleanly separate it from the rules of the game (given by an $\mathcal{L}_\mu^k$ formula), which helps clarify the system-policy interaction.

6.8 Discussion and Related Work

The main contribution of this chapter is bridging the gap between model checking games and IHP checking games and thus enabling practical reasoning about security-relevant hyperproperties. To achieve this, we proposed two game-based model checking approaches, enabling the reuse of the plenitude of results on model checking parity games, in particular algorithms and tools [83, 88, 85, 78]. In this sense, using the tool PGSolver is particularly beneficial, as it implements most known algorithms for model checking games, both local, on-the-fly and global ones, as well as some heuristics to improve performance. Thus, depending on the particular problem, one may choose an algorithm with good theoretical properties or simply experiment with

a multitude of different algorithms. More importantly, such a tool seems to be an excellent candidate to build upon, as it calculates all the data needed to create both views. As a result, we can easily build an interactive visualization tool, allowing users to play against player R to enhance their understanding of problematic system-policy interaction. We have left building such a tool for future work.

The idea of using strategies for the analysis of why a system does not satisfy a policy is actually not new. Stevens and Stirling [83] present a similar idea of using strategies to construct and prove the correctness of local, on-the-fly model checking algorithms. The proof technique is based on *open games*, which are a generalization of Stirling's model checking games allowing systematic exploration of parts of the game. They also present the idea of visualizing why a property does not hold as a byproduct of the local model checking algorithm finding the strategy. In comparison with our idea of visualization via views, their visualization seems to be less intuitive and it is in fact given by a command line tool. More importantly, our different views present a useful separation of the states, moves and rules of the game. In addition, our views present a visualization of the notion of H'-simulations.

We have not experimented with model checking very large systems with respect to incremental hyperproperties, as, due to our reduction of the problem to solving parity games, doing so would be dependent on the concrete algorithms for solving parity games. Instead, we present a short survey of the time complexity of such algorithms. Currently, the existence of a polynomial time algorithm for solving parity games is a major open problem [41]. The reason is that solving a parity game is equivalent to the problems of model checking the mu-calculus and the complementation of ω-tree automata [24, 80, 70]. Most of the algorithms for solving parity games run in exponential time, for instance this is the case for the recursive algorithm by Zielonka [88] and the strategy improvement one by Vöge and Jurdziński [85].

Surprisingly, the promising and well-behaved in practice (see [29]) policy iteration algorithms, proposed by Vöge and Jurdziński [85] and Schewe [78], can also take exponential time on some parity games [28]. The theoretically fastest algorithms for the problem are randomized algorithms by Kalai [42] as well as by Matoušek, Sharir and Welzl [56]. The fastest deterministic subexponential (in the size of the game) algorithm for the solution of parity games uses only polynomial space and runs in time $2^{O(\sqrt{n \log n})}$, where n is the number of vertices in the game.

Although there is no proof that there are polynomial algorithms for solving parity games, Friedmann and Lange [29] show that parity games can be solved efficiently in practice. One of their results is that the recursive algorithm by Zielonka [88] has the best performance in practice, being able to handle games of size up to 1 million nodes. Although these results look promising, we acknowledge that the topic of practical model checking of incremental hyperproperties needs further exploration.

6.9 Summary

We have demonstrated the potential of practical model checking of incremental hyperproperties using off-the-shelf tools. There are several potential approaches: the traditional approach of reducing the problem to model checking a modal mu-calculus formula on a transformed system as well as game-based approaches. The latter rely on transforming IHP checking games into parity games or directly produce a parity game (by transforming a formula and a system) and then solve it. The main advantage of game-based approaches is the possibility of using a winning strategy as a witness as to why a particular system is or is not secure with respect to some policy. We proposed two views, namely the extended game graph view and tree view, that have the potential to facilitate the illustration of the system-policy interaction. Possible directions for future work include implementing a tool, based on PGSolver, that presents the extended game graph view and tree view, as well as developing a prototype interactive tool based on the views, allowing the user to play against player R as a means to illustrate the interplay between a system and a policy. In addition, we would like to extend the approach to decidable classes of infinite state systems and to consider a game-theoretic semantics for IHP-preserving refinement.

Chapter 7

Conclusions

This dissertation focused on reasoning about hyperproperties, which are important as they provide an abstract formalization of security policies. The main contributions are the introduction, formalization and investigation of the classes of security-relevant holistic and incremental hyperproperties and the development of a framework for reasoning about these, including verification via model checking games. Our incremental hyperproperties are instances of novel, general coinductive predicates, which we discovered and whose importance has been recently reaffirmed [45, 38]. This work can also be seen as giving a blueprint and laying the foundations for a generic verification methodology for holistic hyperproperties.

7.1 Main Contributions

This work started by identifying and formalizing the important class of holistic hyperproperties (in Chapter 3), specified using universal and possibly existential quantification on traces, as well as relations on those traces. We argued that holistic hyperproperties are important in practice (see, for instance, [75, 18, 19]), however, they lack a generic verification methodology as the combination of quantification over traces and the relations on whole infinite traces is difficult to reason about.

To remedy this situation we proposed a framework for reasoning about holistic hyperproperties based on alternative, coinductive, local specifications called incremental hyperproperties. In addition, we presented some techniques to convert holistic specifications into incremental ones. We argued (as demonstrated by Chapters 5 and 6) that incremental specifications enjoy feasible verification methodologies, although

the specifications are less intuitive and elegant. By taking an incremental, coalgebraic perspective on systems and specifications we were able to introduce a framework for reasoning about both holistic and incremental hyperproperties. Corresponding to an incremental hyperproperty is the notion of an H'-simulation relation, a novel type of coinductive predicate, generalizing bisimulation. These relations are convenient for verification; moreover an H'-simulation allows reasoning about the corresponding holistic hyperproperty H.

To better situate our work in the security research field, we explored the connection between incremental hyperproperties and the most closely related verification technique for high level security polices — via unwinding. To achieve this, we proposed a framework for coinductive unwinding of security-relevant hyperproperties based on Mantel's MAKS framework and our work on holistic and incremental hyperproperties. The resultant framework allows building (alternatively implying) BSPs from coinductive unwinding relations and composing security-relevant hyperproperties from BSPs. It turns out that the coinductive unwinding relations in the enhanced framework are instances of H'-simulation relations. Thus, we show that the methodology (via H'-simulations) can be used to reason about a large class of holistic hyperproperties, such as (but not limited to) the class of possibilistic information flow hyperproperties. The latter are instances of liveness hyperproperties, which is noteworthy: we have gone beyond the current state of the art (boasting a methodology for k-safety hyperproperties only [19]) and proposed a novel methodology that also works for a class of liveness hyperproperties. Characterizing the precise class of liveness hyperproperties, for which our techniques work, is left for future work. Finally, the ability to reason about infinite systems that cannot be approximated by finite ones, i.e. about systems in which security-relevant behavior never stops occurring, also makes our framework noteworthy.

The claim that incremental hyperproperties enjoy feasible verification methodologies is crucial for our framework for reasoning about hyperproperties. To demonstrate this, we explored one such methodology based on model-checking games. We showed that the polyadic modal mu-calculus $\mathcal{L}_\mu^k$ (and in particular its fragment $I\mathcal{L}_\mu^k$) is a suitable logic for expressing the known incremental hyperproperties. We introduced H'-simulation and incremental hyperproperty checking games allowing the characterization of the satisfaction relation between a system and an incremental hyperproperty in terms of playing a game. The practical significance of this characterization is in the potential use of off-the-shelf game-based model checking tools for incremental hyperproperties. The main advantage of this approach is the possibility of using a winning strategy as a witness as to why a particular system is or is not secure with respect to some policy. From a theoretical perspective, the proposed approach is also feasible, as checking of incremental hyperproperties expressible in $\mathcal{L}_\mu^k$ is decidable and can be done in polynomial time.

7.2 Future Work

In the future, we plan to add some of the missing pieces to the generic framework presented in this work. One promising direction is characterizing the class of incrementalizable holistic hyperproperties. In addition, it would be interesting to explore the topic of hyperproperty-preserving refinement based on H'-simulation relations and the game theoretic semantics given here. Another interesting direction is to incorporate reasoning about quantitative hyperproperties into the framework. The exploration of methods for dynamic enforcement of hyperproperties and characterizing the class of dynamically enforceable hyperproperties is another exciting direction for future work. Finally, we would like to extend the approach to decidable classes of infinite state systems. We next present some of these ideas in more detail.

7.2.1 Incrementalizable Holistic Hyperproperties

How useful and interesting our proposed framework is depends significantly on how large the class of incremental/incrementalizable security policies is. So far, we have shown that such a class is substantial, by establishing a connection with unwinding conditions, which are instances of H'-simulation relations. It would be interesting from both theoretical and practical perspectives to characterize the class of incrementalizable holistic hyperproperties. That would mean that we have a framework for reasoning about and verifying policies in that substantial class.

7.2.2 Hyperproperty-preserving Refinement

Stepwise refinement [46] is an important technique for software construction. One typically starts with a very general system specification, which is made increasingly more concrete until one gets to a specification so concrete that it can be directly coded in a programming language. The technique has the advantage that bugs can be found early on in the development life-cycle. At each level of refinement, the system becomes more concrete and thus the set of allowed traces becomes smaller [46]. For system specifications as properties the problem is trivial as refinements (as subsets of a trace set) are guaranteed to preserve the property. However, hyperproperties are in general not preserved under refinement [19, 54, 7] (except for the class of subset-closed hyperproperties [19]). There are some results for refinement of concrete information flow hyperproperties: Mantel proposes several refinement operators that preserve the policy and Alur et al. [7] propose the use of simulation relations for secrecy-preserving refinement. Based on these results and on an initial investigation we believe that our H'-simulations can be applied for reasoning about hyperproperty-preserving refinement. Moreover, as we have given

game semantics to hyperproperties, we are interested in exploring the potential of game-based hyperproperty-preserving refinement.

7.2.3 Reasoning about Quantitative Hyperproperties

The hyperproperties we have explored are qualitative in nature: they are specifications that either hold or do not hold, i.e. predicates H_i with signature $H_i : X \to 2$. However, there are a number of hyperproperties that are quantitative specifications, i.e. predicates Q_i with signature $Q_i : X \to \mathbb{R}$. Such examples are quantitative information flow [14], belief-based quantitative information flow [17], probabilistic noninterference [40], etc. It is interesting to explore the applicability of our techniques for such definitions. One immediate way to bootstrap such an exploration is by studying the model checking games for (a fragment of) the quantitative mu-calculus [25] on linear hybrid systems, using metric coinduction [44] as a reasoning technique.

7.2.4 Towards Dynamic Enforcement of Hyperproperties

It is well-known that security policies that are safety properties can be enforced at runtime by a technique called *execution monitoring* [79]. Schneider defined a class of monitors (called *security automata*) to enforce safety properties; each step of the application (on which the policy has to be enforced) is allowed if and only if the respective automaton can also produce the same step. The monitors can only enforce safety properties. Later on, it turned out that certain types of monitors (called *edit automata*) can be used to enforce some nonsafety properties [50]. It would be interesting whether a parallel can be drawn and results transferred to safety and liveness hyperproperties. Some reasons for this speculation are presented next.

It is known that a version of secure information flow (essentially stipulating that the low outputs of a program should not depend on high inputs, which is a 2-safety hyperproperty) can be reduced to a safety property on a product of the system with itself [84]. As a result, it seems natural that dynamic enforcement of at least some safety hyperproperties should be feasible; these are at least the safety hyperproperties, which can be reduced to a safety property on a k-product of the candidate system (i.e. the k-safety hyperproperties). Indeed, Devriese and Piessens propose such an enforcement method [22] (called *secure multi-execution*) for a particular definition of secure information flow which is a 2-safety hyperproperty. Their main idea is that a program, if executed multiple times, once for each security level and giving inputs and outputs special treatment, becomes automatically secure. It would be interesting to further explore the analogy with safety and liveness properties by considering the following questions. Can all safety hyperproperties be dynamically enforced? If not,

can at least the k-safety hyperproperties (which can be reduced to safety properties on a k-product of the system of interest) be enforced at runtime? Can some liveness hyperproperties be enforced at runtime and if yes, precisely which class? These are exciting directions for future work and we believe that our framework has the suitable machinery for exploring them.

Appendix A

Proofs from Chapter 3

A.1 Proofs for PHH^2.

Lemma 3.3.2. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$\varepsilon \in X \to \varepsilon \in Y \wedge\ (\forall a \in A\ \forall w \in A^\infty.\ aw \in X \to$$

$$\exists b \in A\ \exists u \in A^\infty.\ bu \in Y \wedge a\ R\ b \wedge c(aw,bu)).$$

Proof. By mutual implication.
($\Rightarrow$) Straightforward unfolding of the holistic definition PHH^2.
($\Leftarrow$) By folding the resultant formula, we get the original definition of PHH^2. □

Lemma 3.3.3. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$o(X) \to o(Y) \wedge (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \to (\forall w \in A^\infty.\ w \in X_a \to$$

$$\exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge a\ R\ b \wedge \exists u \in A^\infty.\ u \in Y_b \wedge c(w,u))).$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma 3.3.2 and lift the definition from sets of traces to trees. An important fact is that trees have to be well-formed, so as to match the requirement on languages of partial automata to be nonempty, closed and consistent

(see Section 2.6).
($\Leftarrow$) Convert back the definition lifted on trees to sets of traces. Then use the result of Lemma 3.3.2. □

Lemma 3.3.4. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$o(X) \to o(Y)$$

$$\wedge\ (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \to (\exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge a\ R\ b \wedge$$

$$(\forall w \in A^\infty.\ w \in X_a \to \exists u \in A^\infty.\ u \in Y_b \wedge c(w,u)))).$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma 3.3.3. Then swap the quantifiers $\exists b$ and $\forall w$; this can be done as $b = f(a)$ and effectively b may be replaced with $f(a)$ and the existential quantifier on b can be safely removed; the function f is called a Skolem function.
($\Leftarrow$) The backward swap of the quantifiers is not problematic because in general $\exists y \forall x.\phi(x,y) \to \forall x \exists y.\phi(x,y)$. Then use the result of Lemma 3.3.3. □

Lemma 3.3.5. *The predicate* $\mathsf{PHH}^2(X,Y)$ *holds iff*

$$o(X) \to o(Y)$$

$$\wedge\ (\forall a \in A\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \to \exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge a\ R\ b\ \wedge \mathsf{PHH}^2(X_a,Y_b)).$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma 3.3.4. Rearrange the resulting expression and fold the definition of PHH^2.
($\Leftarrow$) Unfold the definition of PHH^2 and rearrange the resultant expression. Then use the result of Lemma 3.3.4. □

Define function $\mathit{fl} : A^\infty \to A^\infty$, applying f in a pointwise manner, coinductively:

$$\frac{}{\mathit{fl}(\varepsilon) = \varepsilon}\ {\scriptstyle coind} \qquad \frac{\mathit{fl}(x) = y \qquad f(a) = b}{\mathit{fl}(a \cdot x) = b \cdot y}\ {\scriptstyle coind}$$

The following lemma uses an equivalent definition of PHH^2 that gets rid of the existential.

Lemma A.1.1. *For all $S,T \in Sys$ and for all $x \in S$, we have that* $\mathsf{PIH}^2(S,T) \to (x \in S \to \mathit{fl}(x) \in T)$.

Proof. By coinduction.
The *CH* is: $\forall S,T \subseteq \mathsf{Sys}\ \forall x \in S.\mathsf{PIH}^2(S,T) \to (x \in S \to \mathit{fl}(x) \in T)$. Destruct x as $a \cdot w$. Unfold the coinductive definition of PIH^2. Because $a \cdot w \in S$, $w \in S_a$. Because of the definition of PIH^2, we know that in S, one can take an a transition and in T, one will be guaranteed to be able to make an $f(a)$ transition and the resultant state would be related by PIH^2; that is $\mathsf{PIH}^2(S_a, T_{f(a)})$. By the *CH*, we have that $\mathit{fl}(w) \in T_{f(a)}$. It follows that $\mathit{fl}(a \cdot w) \in T$, as needed. The case when $\varepsilon \in S$ is trivial. □

Theorem 3.3.6 (Incrementalization of PHH^2)**.** *For all $S,T \subseteq \mathsf{Sys}$, we have that* $\mathsf{PHH}^2(S,T)$ *iff* $\mathsf{PIH}^2(S,T)$.

Proof. By coinduction.
($\Rightarrow$) Coinduction hypothesis *CH*: for all $S,T \subseteq \mathsf{Sys}$, $\mathsf{PHH}^2(S,T) \to \mathsf{PIH}^2(S,T)$.
By Lemma 3.3.5 $\mathsf{PHH}^2(S,T)$ is replaced by its equivalent definition:

$$o(S) \to o(T)$$

$$\wedge\ (\forall a \in A\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \to \exists b \in A\ \exists T_b \subseteq \mathsf{Sys}\ .\ T \xrightarrow{b} T_b \wedge a\ R\ b\ \wedge \mathsf{PHH}^2(S_a,T_b)).$$

Then $\mathsf{PIH}^2(S,T)$ is destructed as:

$$(o(S) \to o(T)$$

$$\wedge (\forall a \in A.\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \to \exists b \in A\ \exists T_b \subseteq \mathsf{Sys}\ .\ T \xrightarrow{b} T_b \wedge a\ R\ b \wedge \mathsf{PIH}^2(S_a,T_b))).$$

Clearly, $\mathsf{PHH}^2(S,T)$ implies $\mathsf{PHH}^2(S_a,T_b)$ whenever $\exists S_a \subseteq \mathsf{Sys}.S \xrightarrow{a} S_a$ and $b = f(a)$. By the coinduction hypothesis *CH* it follows that $\mathsf{PIH}^2(S_a,T_b)$. Thus we may conclude that $\mathsf{PIH}^2(S,T)$. The case when $\varepsilon \in S$ is trivial by applying *CH*.
($\Leftarrow$) Directly follows from Lemma A.1.1, because $\forall x \in S \cdot \mathit{fl}(x) \in T$ is reformulation of $\mathsf{PHH}^2(S,T)$. □

Note that all proofs in Appendix A.1 have been verified in Coq.

A.2 Proofs for SHH^2.

First, unfold the holistic definition of SHH^2.

Lemma A.2.1. *The predicate* $\mathsf{SHH}^2(X,Y)$ *holds iff*

$$\forall a \in A\ \forall w \in A^{\omega}.\ (p(a) \wedge aw \in X \rightarrow$$

$$\exists b \in A\ \exists u \in A^{\infty}.bu \in Y \wedge\ b = f(a) \wedge c_1(aw,bu))$$

$$\wedge \forall a \in A\ \forall w \in A^{\omega}.\ (\neg p(a) \wedge aw \in X \rightarrow \exists y \in A^{\infty}.y \in Y \wedge c_1(aw,bu)).$$

Proof. By mutual implication.
($\Rightarrow$) Straightforward unfolding of the holistic definition SHH^2.
($\Leftarrow$) By folding the resultant formula, we get the original definition of SHH^2. □

Lemma A.2.2. *The predicate* $\mathsf{SHH}^2(X,Y)$ *holds iff*

$$\forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge p(a) \rightarrow \forall w \in A^{\omega}.w \in X_a \rightarrow$$

$$\exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge b = f(a) \rightarrow \exists u \in A^{\omega}.u \in Y_b \wedge c_1(w,u)$$

$$\wedge \forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge \neg p(a) \rightarrow \forall w \in A^{\omega}.w \in X_a \rightarrow$$

$$\exists u \in A^{\omega}.u \in Y \wedge c_1(w,u)$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma A.2.1 and lift the definition from sets of traces to trees.
($\Leftarrow$) Convert back the definition lifted on trees to sets of traces. Then use the result of Lemma A.2.1. □

Lemma A.2.3. *The predicate* $\mathsf{SHH}^2(X,Y)$ *holds iff*

$$\forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge p(a) \rightarrow \exists b \in A\ \exists Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge b = f(a) \rightarrow$$

$$\forall w \in A^{\omega}.w \in X_a \rightarrow \exists u \in A^{\omega}.u \in Y_b \wedge c_1(w,u)$$

$$\wedge \forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge \neg p(a) \rightarrow \forall w \in A^{\omega}.w \in X_a \rightarrow$$

$$\exists u \in A^{\omega}.u \in Y \wedge c_1(w,u)$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma A.2.2. Then swap the quantifiers $\exists b$ and $\forall w$; this can be done as $b = f(a)$ and effectively b may be replaced with $f(a)$ and the existential

quantifier on b can be safely removed. ($\Leftarrow$) The backward swap of the quantifiers is not problematic because in general $\exists y \forall x.\phi(x,y) \to \forall x \exists y.\phi(x,y)$. Then use the result of Lemma A.2.2. □

Lemma A.2.4. *The predicate* $\mathsf{SHH}^2(X,Y)$ *holds iff*

$$\forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\,.\, X \xrightarrow{a} X_a \wedge p(a) \to \exists Y_b \subseteq \mathsf{Sys}.Y \xrightarrow{f(a)} Y_b \wedge \mathsf{SHH}^2(X_a, Y_{f(a)})$$

$$\wedge\, \forall a \in A.\, \forall X_a \subseteq \mathsf{Sys}\,.\, X \xrightarrow{a} X_a \wedge \neg p(a) \to \mathsf{SHH}^2(X_a, Y).$$

Proof. By mutual implication.
($\Rightarrow$) Start with the result of Lemma A.2.3. Rearrange the resulting expression and fold the definition of SHH^2.
($\Leftarrow$) Unfold the definition of SHH^2 and rearrange the resultant expression. Then use the result of Lemma A.2.3. □

Now some helper lemmas to prove the main result: Theorem 3.3.11.
Lemma 3.3.9. *For all* $s,t \in A^\omega$*, we have that* $(p_s(t) \wedge s \sim_\mathrm{p} t) \to c_1(s,t)$.

Proof. By coinduction.
The coinduction hypothesis *CH* is $\forall s,t \in A^\omega.(p_s(t) \wedge s \sim_\mathrm{p} t) \to c_1(s,t)$. First, destruct s and t as $a \cdot w$ and $b \cdot u$. Then use inversion on $p_s(t)$ (results in $p(b)$ and $p_s(u)$) and $s \sim_\mathrm{p} t$ (resulting in the following case analysis, with two viable cases).

- $p(a)$, $p(b)$ and $w \sim_\mathrm{p} u$. By *CH* we have $c_1(w,u)$. Because $b = f(a)$ and by definition of c_1, $p(a), p(b)$ and $c_1(w,u)$, it follows that $c_1(a \cdot w, b \cdot u)$.
- $\neg p(a)$, $p(b)$ and $w \sim_\mathrm{p} b \cdot u$. By *CH* we have $c_1(w, b \cdot u)$. By definition of c_1, $\neg p(a)$ and $c_1(w, b \cdot u)$, we have $c_1(a \cdot w, b \cdot u)$.

□

Lemma 3.3.10. *For all* $s,t \in P_{\Box\Diamond}$*, we have that* $c_1(s,t) \to (p_s(t) \wedge s \sim_\mathrm{p} t)$.

Proof. The proof of $c_1(s,t) \to p_s(t)$ follows by contradiction, assuming $\exists s \exists t \cdot s \in P_{\Box\Diamond} \wedge t \in P_{\Box\Diamond} \wedge c_1(s,t) \to \neg p_s(t)$.
The proof of $c_1(s,t) \to s \sim_\mathrm{p} t$ proceeds by coinduction. The coinduction hypothesis *CH* is $\forall s,t \in P_{\Box\Diamond}.c1(s,t) \to s \sim_\mathrm{p} t$. First, destruct s and t as $a \cdot w$ and $b \cdot u$. Then use inversion on $s \in P_{\Box\Diamond}$ (results in $w \in P_{\Box\Diamond}$) and $c_1(s,t)$ (resulting in the following case analysis, with two viable cases).

- $p(a)$, $p(b)$ and $c_1(w,u)$. By *CH* we have $w \sim_p u$. Because $b = f(a)$ and by definition of the first rule of $\sim_p$, $p(a), p(b)$ and $w \sim_p u$, it follows that $a \cdot w \sim_p b \cdot u$.
- $\neg p(a)$ and $c_1(w, b \cdot u)$. By *CH* we have $w \sim_p b \cdot u$. By definition of the second rule of $\sim_p$, $\neg p(a)$ and $c_1(w, b \cdot u)$, we have $a \cdot w \sim_p b \cdot u$.

□

Let $filter : P_{\Box\Diamond} \to P_{\Box\Diamond}$ be the usual function filtering $\neg p$ events. Such a function is guaranteed to be productive as long as it is applied to streams in $P_{\Box\Diamond}$. Define function $fl : A^\omega \to A^\omega$, applying f in a pointwise manner, coinductively:

$$\frac{fl(x) = y \qquad f(a) = b}{fl(a \cdot x) = b \cdot y}\ coind$$

Lemma A.2.5. *For all $x \in P_{\Box\Diamond}$, the following predicate holds $c_1(x, fl(filter\ x))$.*

Proof. By coinduction.
The coinduction hypothesis *CH* is $\forall x \in P_{\Box\Diamond}.c_1(x, fl(filter\ x))$. Destruct x as $a \cdot w$ and perform case analysis on a:

- $p(a)$. By the *CH* $c_1(w, fl(filter\ w))$. By the definition of c_1, we have $c_1(a \cdot w, f(a) \cdot fl(filter\ w))$.
- $\neg p(a)$. Because $a \cdot w \in P_{\Box\Diamond}$, it is guaranteed that there will eventually be an element $b \in A$ such that $p(b)$. Without loss of generality assume this b to be the first such element in w. By reasoning as in the previous case and applying *CH* and the first rule of the definition of c_1, it follows that $c_1(x, fl(filter\ x))$.

□

Lemma A.2.6. *For all $x \in P_{\Box\Diamond}$, $y, y_1 \in A^\omega$, $c_1(x,y)$ and $c_1(x,y_1)$ imply that $y \sim y_1$.*

Proof. By coinduction.
The coinduction hypothesis *CH* is $\forall x \in P_{\Box\Diamond} \forall y \in A^\omega \forall y_1 \in A^\omega.(c_1(x,y) \wedge c_1(x,y_1)) \to y \sim y_1$ Start by using inversion on $c_1(x,y)$ and $c_1(x,y_1)$. Case analysis on the fist element of x:
Case 1**:** $p(x(0))$
By the inversion of $c_1(x,y)$, we have that $p(y(0))$ and $f(x(0)) = y(0)$ and $c_1(x',y')$. Also by the inversion of $c_1(x,y_1)$, we have that $p(y_1(0))$ and $f(x(0)) = y_1(0)$ and $c_1(x',y_1')$. By *CH* we have $y' \sim y_1'$. Also $y(0) = y_1(0)$, as needed. Thus $y \sim y_1$.

Case 2: $\neg p(x(0))$
Since $x \in P_{\Box\Diamond}$ there exists a first k such that $p(x(k))$. By the inversion of $c_1(x,y)$, it follows that $p(y(0))$ and $y(0) = f(x(k))$. Similarly $p(y_1(0))$ and $y_1(0) = f(x(k))$. It follows that $y(0) = y_1(0)$ and by the coinduction hypothesis $y' \sim y'_1$, as needed.
□

Lemma A.2.7. *For all $x,y,y_1 \in P_{\Box\Diamond}$, $c_1(x,y)$ and $c_1(x,y_1)$ imply that $y = y_1$.*

Proof. Trivial as bisimilarity implies equality in the final coalgebra of streams. Note that in all proofs about elements in $P_{\Box\Diamond}$, productivity of the definitions is guaranteed by the fact they are in $P_{\Box\Diamond}$. This fact could also be seen as a fairness condition (such that one side of the definition does not go infinitely far ahead of the other). □

Lemma A.2.8. *For all $X,Y \in P_{\Box\Diamond}, x \in P_{\Box\Diamond}$, $\mathsf{SIH}^2(X,Y)$ and $x \in X$ imply that $\mathit{fl}(\mathit{filter}(x)) \in Y$.*

Proof. By contradiction.
Let X_γ denote the derivative of some system X w.r.t. a string γ. Assume there exist systems S,T, stream $s \in P_{\Box\Diamond}$ with $\mathsf{SIH}^2(S,T)$, $s \in S$ and $\mathit{fl}(\mathit{filter}(s)) \notin T$. Let $t = \mathit{fl}(\mathit{filter}(s))$. Note that there exists some 2-tuple (S_{s_e}, T_{t_e}) such that $s_e \cdot s' = s$ and $t_e \cdot t' = t$, $t_e = \mathit{fl}(\mathit{filter}(s_e))$, $s'(0) = a$, $p(a)$ hold, but $\neg \mathit{test}_{f(a)} T_{t_e}$ (because of the restriction $P_{\Box\Diamond}$). This directly contradicts the assumption that $\mathsf{SIH}^2(X,Y)$. □

The following lemma uses an equivalent definition of SHH^2 that gets rid of the existential.

Lemma A.2.9. *For all $S,T \subseteq P_{\Box\Diamond}$ and for all $x \in S$, we have that $\mathsf{SIH}^2(S,T) \to (x \in S \to (\mathit{fl}(\mathit{filter}\ x) \in T))$.*

Proof. By coinduction.
The *CH* is: $\forall S,T \subseteq P_{\Box\Diamond}\ \forall x \in S. \mathsf{SIH}^2(S,T) \to (x \in S \to \mathit{fl}(\mathit{filter} x) \in T)$. Destruct x as $a \cdot w$. Unfold the coinductive definition of SIH^2. Because $a \cdot w \in S$, $w \in S_a$. Because of the definition of SIH^2, we know that if in S, one can take an a transition s.t. $p(a)$ then in T, one will be guaranteed to be able to make an $f(a)$ transition and the resultant state would be related by SIH^2. That is $\mathsf{SIH}^2(S_a, T_{f(a)})$. Alternatively, if $\mathit{test}_a(S)$ and $\neg p(a)$, then $\mathsf{SIH}^2(S_a,T)$. By the *CH* and since $a \cdot w \in P_{\Box\Diamond}$, we have that $\mathit{fl}(w) \in T$. It follows that $\mathit{fl}(\mathit{filter}(a \cdot w)) \in T$, as needed. □

Theorem 3.3.11 (Incrementalization of SHH^2)**.** *For all $S,T \in P_{\Box\Diamond}$, we have that* $\mathsf{SHH}^2(S,T)$ iff $\mathsf{SIH}^2(S,T)$.

Proof. By coinduction.
($\Rightarrow$) Coinduction hypothesis *CH*: for all $S,T \in P_{\Box\Diamond}$, $\mathsf{SHH}^2(S,T) \to \mathsf{SIH}^2(S,T)$.
By Lemma A.2.4 $\mathsf{SHH}^2(S,T)$ is replaced by its equivalent definition:

$$\forall a \in A.\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \wedge p(a) \to \exists T_b \subseteq \mathsf{Sys}.T \xrightarrow{f(a)} T_b \wedge \mathsf{SHH}^2(S_a, T_{f(a)})$$

$$\wedge\, \forall a \in A.\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \wedge \neg p(a) \to \mathsf{SHH}^2(S_a, T).$$

Then $\mathsf{SIH}^2(X,Y)$ is destructed as:

$$(\forall a \in A\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \wedge p(a) \to \exists T_b \subseteq \mathsf{Sys}.T \xrightarrow{f(a)} T_b \wedge \mathsf{SIH}^2(S_a, T_{f(a)})$$

$$\wedge\, \forall a \in A.\ \forall S_a \subseteq \mathsf{Sys}\ .\ S \xrightarrow{a} S_a \wedge \neg p(a) \to \mathsf{SIH}^2(S_a, T))$$

Case 1: Whenever $p(a)$ and $test_a(S)$, $\mathsf{SHH}^2(S,T)$ implies $\mathsf{SHH}^2(S_a, T_{f(a)})$. By *CH* it follows that $\mathsf{SIH}^2(S_a, T_{f(a)})$; therefore, it must be that $\mathsf{SIH}^2(S,T)$ (same definition, taking the derivative is deterministic).
Case 2: Whenever $\neg p(a)$ and $test_a(S)$, $\mathsf{SHH}^2(S,T)$ implies $\mathsf{SHH}^2(S_a,T)$. By *CH* it follows that $\mathsf{SIH}^2(S_a,T)$; therefore, it must be that $\mathsf{SIH}^2(S,T)$.
($\Leftarrow$) Directly from Lemma A.2.9. □

Note that most proofs in Appendix A.2 (except some proofs related to *filter* and the productivity of the definitions, notoriously difficult in Coq) have been verified in Coq.

A.3 Proofs for OHH.

The proofs are not too different from previous ones and some details are skipped. First, unfold the holistic definition of OHH.

Lemma A.3.1. *The predicate* $\mathsf{OHH}^2(X,Y)$ *holds iff*

$$\forall a \in A\ \forall w \in A^\infty.\ aw \in X \to$$

$$\forall b \in A\ \forall u \in A^\infty.bu \in Y \to a \cdot w \sim_{\mathrm{pt_i}} b \cdot u \to a \cdot w \sim_{\mathrm{pt}} b \cdot u$$

Proof. By mutual implication.
Straightforward. □

Next, we proceed simplifying the formula:

$$\forall a \in A\ \forall w \in A^{\infty}.\ aw \in X \rightarrow$$

$$\forall b \in A\ \forall u \in A^{\infty}.bu \in Y \rightarrow a \cdot w \sim_{\mathrm{pt_i}} b \cdot u \rightarrow a \cdot w \sim_{\mathrm{pt}} b \cdot u$$

with case analysis on a and b give the following interesting cases (remaining cases are impossible):
Case 1: $a \in A_i \wedge p(a)$ and $b \in A_i \wedge p(b)$
Under this restriction, we have the following

$$\forall a \in A_i\ \forall w \in A^{\infty}.p(a) \wedge aw \in X \rightarrow$$

$$\forall b \in A_i\ \forall u \in A^{\infty}.p(b) \wedge bu \in Y \rightarrow a \cdot w \sim_{\mathrm{pt_i}} b \cdot u \rightarrow a \cdot w \sim_{\mathrm{pt}} b \cdot u$$

$$\Updownarrow$$

$$\forall a \in A_i.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge p(a) \rightarrow \forall b \in A_i\ \forall Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge p(b) \rightarrow$$

$$\forall w \in X_a \forall u \in Y_b.(b = f(a) \wedge w \sim_{\mathrm{pt_i}} u \rightarrow w \sim_{\mathrm{pt}} u)$$

$$\Updownarrow$$

$$\forall a \in A_i \forall b \in A_i.\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge \wedge p(b) \rightarrow$$

$$\forall w \in X_a \forall u \in Y_b.(b = f(a) \wedge w \sim_{\mathrm{pt_i}} u \rightarrow w \sim_{\mathrm{pt}} u)$$

$$\Updownarrow$$

$$\forall a \in A_i\ \forall b \in A_i\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(f(a)) \rightarrow$$

$$\mathsf{OHH}^2(X_a, Y_b)$$

Case 2: $a \in A_o \wedge p(a)$ and $b \in A_o \wedge p(b)$
Under this restriction, we have the following

$$\forall a \in A_o\ \forall w \in A^\infty.p(a) \wedge aw \in X \rightarrow$$

$$\forall b \in A_o\ \forall u \in A^\infty.p(b) \wedge bu \in Y \rightarrow a \cdot w \sim_{\mathrm{pt_i}} b \cdot u \rightarrow a \cdot w \sim_{\mathrm{pt}} b \cdot u$$

$$\Updownarrow$$

$$\forall a \in A_o.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge p(a) \rightarrow \forall b \in A_o\ \forall Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge p(b) \rightarrow$$

$$\forall w \in X_a \forall u \in Y_b.(w \sim_{\mathrm{pt_i}} u \rightarrow a \cdot w \sim_{\mathrm{pt}} b \cdot u)$$

$$\Updownarrow$$

$$\forall a \in A_o \forall b \in A_o.\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(b) \rightarrow$$

$$\forall w \in X_a \forall u \in Y_b.(w \sim_{\mathrm{pt_i}} u \rightarrow (b = f(a) \wedge w \sim_{\mathrm{pt}} u))$$

$$\Updownarrow$$

$$\forall a \in A_o\ \forall b \in A_o.\ \ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(b) \rightarrow$$

$$b = f(a) \wedge \mathsf{OHH}^2(X_a, Y_b).$$

Case 3: $a \in A \wedge \neg p(a)$ and $b \in A \wedge \neg p(b)$
Under this restriction, we have the following

$$\forall a \in A\ \forall w \in A^{\infty}.\neg p(a) \wedge aw \in X \rightarrow$$

$$\forall b \in A\ \forall u \in A^{\infty}.\neg p(b) \wedge bu \in Y \rightarrow a \cdot w \sim_{\mathrm{pt_i}} b \cdot u \rightarrow a \cdot w \sim_{\mathrm{pt}} b \cdot u$$

$$\Updownarrow$$

$$\forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge \neg p(a) \rightarrow \forall b \in A\ \forall Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge \neg p(b) \rightarrow$$

$$\forall w \in X_a \forall u \in Y_b.(w \sim_{\mathrm{pt_i}} u \rightarrow w \sim_{\mathrm{pt}} u)$$

$$\Updownarrow$$

$$\forall a \in A.\ \forall X_a \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge \neg p(a) \rightarrow \forall b \in A\ \forall Y_b \subseteq \mathsf{Sys}\ .\ Y \xrightarrow{b} Y_b \wedge \neg p(b) \rightarrow$$

$$\mathsf{OHH}^2(X_a, Y_b)$$

$$\forall a \in A\ \forall b \in A\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge \neg p(a) \wedge \neg p(b) \rightarrow$$

$$\mathsf{OHH}^2(X_a, Y_b)$$

Case 4: $x = \varepsilon$ implies that $\varepsilon \sim_{\mathrm{pt_i}} y \rightarrow \varepsilon \sim_{\mathrm{pt}} y$; thus $y = \varepsilon$ by definition. $y = \varepsilon$ implies that $x \sim_{\mathrm{pt_i}} \varepsilon \rightarrow x \sim_{\mathrm{pt}} \varepsilon$; thus $x = \varepsilon$ by definition.

Lemma A.3.2. *The predicate* $\mathsf{OHH}^2(X,Y)$ *holds iff*

$$(o(X) \leftrightarrow o(Y))$$

$$\wedge\, \forall a \in A_i\ \forall b \in A_i\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(f(a)) \rightarrow \mathsf{OHH}^2(X_a, Y_b)$$

$$\wedge\, \forall a \in A_o\ \forall b \in A_o.\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge p(a) \wedge p(b) \rightarrow$$

$$b = f(a) \wedge \mathsf{OHH}^2(X_a, Y_b)$$

$$\wedge\, \forall a \in A\ \forall b \in A\ \forall X_a \subseteq \mathsf{Sys}\ \forall Y_b \subseteq \mathsf{Sys}\ .\ X \xrightarrow{a} X_a \wedge Y \xrightarrow{b} Y_b \wedge \neg p(a) \wedge \neg p(b) \rightarrow \mathsf{OHH}^2(X_a, Y_b)).$$

Proof. By mutual implication, using the 4 cases presented above.

□

Theorem 3.3.14 (Incrementalization of OHH^2**).** *For all* $S, T \subseteq \mathsf{Sys}$*, we have that* $\mathsf{OHH}^2(S,T)$ *iff* $\mathsf{OIH}^2(S,T)$.

Proof. By coinduction.
($\Rightarrow$) Coinduction hypothesis *CH*: for all $S,T \subseteq \mathsf{Sys}$, $\mathsf{OHH}^2(S,T) \rightarrow \mathsf{OIH}^2(S,T)$.
By Lemma A.3.2 $\mathsf{OHH}^2(S,T)$ is replaced by its equivalent definition:

$$(o(S) \leftrightarrow o(T)$$

$$\wedge\, \forall a \in A_i\, \forall b \in A_i\, \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge p(a) \wedge p(f(a)) \rightarrow \mathsf{OHH}^2(S_a,T_b)$$

$$\wedge\, \forall a \in A_o\, \forall b \in A_o.\ \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge p(a) \wedge p(b) \rightarrow$$

$$b = f(a) \wedge \mathsf{OHH}^2(S_a,T_b)$$

$$\wedge\, \forall a \in A\, \forall b \in A\, \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge \neg p(a) \wedge \neg p(b) \rightarrow \mathsf{OHH}^2(S_a,T_b)).$$

Then $\mathsf{OIH}^2(S,T)$ is destructed as:

$$(o(S) \leftrightarrow o(T)$$

$$\wedge\, \forall a \in A_i\, \forall b \in A_i\, \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge p(a) \wedge p(f(a)) \rightarrow \mathsf{OIH}^2(S_a,T_b)$$

$$\wedge\, \forall a \in A_o\, \forall b \in A_o.\ \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge p(a) \wedge p(b) \rightarrow$$

$$b = f(a) \wedge \mathsf{OIH}^2(S_a,T_b)$$

$$\wedge\, \forall a \in A\, \forall b \in A\, \forall S_a \subseteq \mathsf{Sys}\, \forall T_b \subseteq \mathsf{Sys}\, .\, S \xrightarrow{a} S_a \wedge T \xrightarrow{b} T_b \wedge \neg p(a) \wedge \neg p(b) \rightarrow \mathsf{OIH}^2(S_a,T_b)).$$

Clearly, $\mathsf{OHH}^2(S,T)$ implies $\mathsf{OHH}^2(S_a,T_{f(a)})$ whenever $a \in A_i \wedge \mathit{test}_a(X) \wedge p(a) \wedge \mathit{test}_{f(a)}(T)$. By the coinduction hypothesis *CH* it follows that $\mathsf{OIH}^2(S_a,T_b)$. Because the constructors are identical, it follows that $\mathsf{OHH}^2(S,T) \rightarrow \mathsf{OIH}^2(S,T)$. Similarly for the case when $a \in A_o \wedge b \in A_o \wedge \mathit{test}_a(X) \wedge p(a) \wedge \mathit{test}_b(Y) \wedge p(b)$ and the case $a \in A \wedge b \in A \wedge \mathit{test}_a(S) \wedge \neg p(a) \wedge \mathit{test}_b(Y) \wedge \neg p(b)$.
($\Leftarrow$) By coinduction.
The coinduction hypothesis is *CH* : for all$S,T \subseteq \mathsf{Sys}$ we have

$$\mathsf{OIH}^2(S,T) \rightarrow \forall x \in S\, \forall y \in T.x \sim_{\mathrm{pt_i}} y \rightarrow x \sim_{\mathrm{pt}} y$$

Proof by inversion on the assumption $x \sim_{\mathrm{pt_i}} y$, where $x = a \cdot w$ and $y = b \cdot u$:
Case 1: $a \in A_i \wedge p(a) \wedge b \in A_i \wedge p(b)$. By our assumption ($x \sim_{\mathrm{pt_i}} y$) we know that $b = f(a)$ and by $\mathsf{OIH}^2(S,T)$ we know $\mathsf{OIH}^2(S_a,T_{f(a)})$. By *CH* it follows that $w \sim_{\mathrm{pt}} u$ and we can conclude by the definition of $\sim_{\mathrm{pt}}$ (second rule) that $aw \sim_{\mathrm{pt}} bu$.
Case 2: $a \in A_o \wedge p(a) \wedge b \in A_o \wedge p(b)$. By our assumption ($\mathsf{OIH}^2(S,T)$) we know that $b = f(a)$ and $\mathsf{OIH}^2(S_a,T_{f(a)})$. By *CH* it follows that $w \sim_{\mathrm{pt}} u$ and we can conclude by the definition of $\sim_{\mathrm{pt}}$ (second rule) that $aw \sim_{\mathrm{pt}} bu$.

Case 3: $\neg p(a) \wedge \neg p(b)$. By assumptions we know that $\mathsf{OIH}^2(S_a, T_b)$ and $w \sim_{\mathrm{pt_i}} u$. By *CH* it follows that $w \sim_{\mathrm{pt}} u$. By the definition of $\sim_{\mathrm{pt}}$ (third rule) we have $aw \sim_{\mathrm{pt}} bu$.
Case 4: $x = \varepsilon$ or $y = \varepsilon$: as before and by the *CH* and third rule $\varepsilon \sim_{\mathrm{pt}} \varepsilon$. □

Theorem 3.4.3. *The predicate $H'(S_1, \ldots, S_k)$ on G-systems $\langle S_i, \alpha_i, x_i \rangle$ for $i \in 1..k$ holds iff there exists some H-simulation Q s.t. the k-tuple of the start states $\langle x_1, \ldots, x_k \rangle \in Q$.*

Proof. Straightforward using $Q \subseteq \Psi_{H'}(Q)$ and $\mathbf{gfp}(\Psi_{H'}) = H'$ and the Knaster-Tarski coinduction principle (see Equation 3.1). □

Appendix B

Proofs from Chapter 4

Theorem 4.3.3. *Let $V = (A_v, A_n, A_c)$ be a view and T be a set of traces. Then the following implications hold: $BSD_V(T) \to D_V(T)$ and $D_V(T) \to R_V(T)$.*

Proof. $BSD_V(T) \to D_V(T)$: from the definition and selecting $\beta' = \beta$.
$D_V(T) \to R_V(T)$: proof techniques similar to the one used in Appendix A and our recent work [63]. First, define ${D_V}^2(X,Y)$ and ${R_V}^2(X,Y)$ to be the same as D_V and R_V but taking two parameters. In other words

$$D_V(X,Y) \mathrel{\widehat{=}} \forall \alpha \in A^\infty \, \forall \beta \in A^* \, \forall c \in A_c \, . \, [\beta \cdot c \cdot \alpha \in X \to \exists \alpha' \in A^\infty, \beta' \in A^* . \, (\beta' \cdot \alpha' \in Y \wedge \beta' \approx_{A_v \cup A_c} \beta \wedge \alpha' \approx_{A_v \cup A_c} \alpha)]$$

$${R_V}^2(X,Y) \mathrel{\widehat{=}} \forall \sigma \in X \, \exists \gamma \in Y \, . \, (no_{A_c}(\gamma) \wedge \sigma \approx_{A_v} \gamma).$$

For all $T \in \mathsf{Sys}$ we have ${D_V}^2(T,T)$ iff $D_V(T)$, similarly we have ${R_V}^2(T,T)$ iff $R_V(T)$. Proceed by coinduction with hypothesis: $CH^2 : \forall S, T \in \mathsf{Sys}. D_V(S,T) \to R_V(S,T)$. Assume $D_V(T,T)$ holds and let t_0 be an arbitrary trace s.t. $t_0 \in T$. Now destruct t_0. If $t_0 = \varepsilon$ the needed trace is ε. Otherwise, t_0 is destructed as $\beta \cdot c \cdot \alpha$. Without loss of generality assume that $no_{A_c}(\beta)$. Because $D_V(T,T)$ holds it has to be that there is a trace $t_1 \in T$ s.t. $t_1 = \beta' \cdot \alpha'$, $\beta' \in A^*, \alpha' \in A^\infty$, $\beta' \approx_{A_v} \beta$, $no_{A_c}(\beta')$ and $\alpha \approx_{A_v \cup A_c} \alpha'$. Because $\beta' \approx_{A_v} \beta \cdot c$, we are ready to apply CH with $(T_{\beta \cdot c}, T_{\beta'})$. It follows that there is trace $\alpha'' \in A^\infty$ s.t. $\alpha'' \in T_{\beta'}$, $no_{A_c}(\alpha'')$ and $\alpha \approx_{A_v} \alpha''$. Hence the required trace is $\beta' \alpha''$. □

Lemma 4.5.1. *Let $T,S \in \mathsf{Sys}$ be arbitrary systems and $R \subseteq T \times S$ be a relation. If $osc_V(T,S,R)$ holds then we have*

$$\forall \alpha_1 \in (A \setminus A_c)^*.(test^*{}_{\alpha_1}(T) \rightarrow$$

$$\exists \alpha_2 \in (A \setminus A_c)^*.\ (test_{\alpha_2}(S) \wedge \alpha_1 \approx_{A_v} \alpha_2 \wedge (T_{\alpha_1}, S_{\alpha_2}) \in R)).$$

Proof. Assume that $osc_V(T,S,R)$ holds for some $T,S \in \mathsf{Sys}$. Let α_1 be an arbitrary trace such that $\alpha_1 \in (A \setminus A_c)^*$ and $test^*{}_{\alpha_1}(T)$. Proceed by induction on the length of α_1.
Base case: for $\alpha_1 = \varepsilon$ take $\alpha_2 = \varepsilon$ (and by definition of osc_V).
Induction case: assume that α_1 has length $k+1$, i.e. it can be destructed as $\alpha_1 = a \cdot w$ for some $a \in A \setminus A_c$ and $w \in A^*$. It has to be that $test_a(T)$. Hence there exists some word $\sigma \in (A \setminus A_c)^*$ such that $test_\sigma(S)$ and $a \approx_{A_v} \sigma$ and $(T_a, S_\sigma) \in R$ (by definition of osc_V). By the inductive hypothesis there is $u \in (A \setminus A_c)^*$ such that $test_u(S_\sigma)$ and $w \approx_{A_v} u$. Thus, for $\alpha_1 = a \cdot w$ there is a trace $\alpha_2 = \sigma \cdot u$ such that $\alpha_1 \approx_{A_v} \alpha_2$ and $(T_{\alpha_1}, S_{\alpha_2}) \in R$. □

Lemma 4.5.2. *For all $T,S \in \mathsf{Sys}$ if there exists $R \subseteq T \times S$ s.t. $osc_V(T,S,R)$ holds, then the following is valid:*

$$\forall \alpha_1 \in (A \setminus A_c)^\infty.(\alpha_1 \in T \rightarrow$$

$$\exists \alpha_2 \in (A \setminus A_c)^\infty.(\alpha_2 \in S \wedge \alpha_1 \approx_{A_v} \alpha_2))$$

Proof. By coinduction. The *CH* is: $osc_V(T,S,R) \rightarrow \forall \alpha_1 \in (A \setminus A_c)^\infty.(\alpha_1 \in T \rightarrow \exists \alpha_2 \in (A \setminus A_c)^\infty.\alpha_2 \in S \wedge \alpha_1 \approx_{A_v} \alpha_2)$. Let α_1 be an arbitrary trace. The case when $\alpha_1 = \varepsilon$ is straightforward. Otherwise, destruct α_1 as $v \cdot w$, where v is finite. By Lemma 4.5.1 we have that there is $u \in (A \setminus A_c)^*$ s.t. $test^*{}_u(S)$, $v \approx_{A_v} u$ and $(T_v, S_u) \in R$. Hence, we can complete the proof by coinduction. □

Theorem 4.5.3. *Let T be an arbitrary system and $R \subseteq T \times T$ be a relation on T. The following implications are valid:* (implications given in the proof)

Proof. 1. $lrf_V(T,T,R) \wedge osc_V(T,T,R) \rightarrow BSD_V(T)$:
By coinduction with *CH*

$$(T,S) \in R \rightarrow$$

$$\forall \alpha \in A^\infty\ \forall \beta \in A^*\ \forall c \in A_c\ .\ [(\beta \cdot c \cdot \alpha \in T) \rightarrow \exists \alpha' \in A^\infty.\ (\beta \cdot \alpha' \in S \wedge \alpha' \approx_{A_v \cup A_c} \alpha)].$$

Let t_0 be an arbitrary trace in T. Destruct t_0 as $\beta \cdot c \cdot \alpha$ where $\alpha \in A^\infty$, $\beta \in A^*$, $c \in A_c$ and w.l.g. assume that $no_{A_c}(\beta)$. Hence there are subtrees T_β and $T_{\beta \cdot c}$ such that $\alpha \in T_{\beta \cdot c}$.

Because $osc_V(T,T,R)$ and $no_{A_c}(\beta)$ we have $(T_\beta, T_\beta) \in R$. Because $lrf_V(T_\beta, T_\beta, R)$, and $test_c(T_\beta)$ it follows that $lrf_V(T_{\beta\cdot c}, T_\beta, R)$ (by definition of lrf_V). From $(T_\beta, T_{\beta\cdot c}) \in R$ and applying the *CH* we get: $\forall\alpha'' \in A^\infty \forall\beta'' \in A^* \forall c'' \in A^c$. $(\beta''\cdot c''\cdot\alpha'' \in T_{\beta\cdot c})$ implies there exists $\alpha''' \in A^\infty.\beta''\alpha''' \in T_\beta$ and $\alpha''' \approx_{A_v\cup A_c} \alpha''$, as needed. In case there are no more A_c events after c, we can still apply Lemma 4.5.2 to get the needed result. In case $t_0 = \varepsilon$, the statement is vacuously true.

2.$lrf_V(T,T,R) \wedge osc_V(T,T,R) \to D_V(T)$: applying Theorem 4.3.3(1).
3.$lrf_V(T,T,R) \wedge osc_V(T,T,R) \to R_V(T)$: applying Theorem 4.3.3(2).
4.$lrbe^\rho_V(T,T,R) \wedge osc_V(T,T,R) \to BSIA_V{}^\rho(T)$: Let $\alpha \in A^\infty, \beta \in A^*, c \in A_c$ be arbitrary such that $\beta\cdot\alpha \in T$ and w.l.g. assume that $no_{A_c}(\beta)$ holds. It follows that $test^*{}_\beta(T)$ holds and $\alpha \in T_\beta$. Clearly $(T_\beta, T_\beta) \in R$ by the definition of osc_V.
Case 1: Assume $En^\rho_V(T, T_\beta, c)$ holds. Then it has to be that $test_c(T_\beta)$ and $(T_\beta, T_{\beta\cdot c}) \in R$. Then the proof proceeds by coinduction with *CH* :

$$(T,S) \in R \to \forall\alpha \in A^\infty\ \forall\beta \in A^* \forall c \in A_c\ .\ [(\beta\cdot\alpha \in T \wedge Adm^\rho_V(T,\beta,c)) \to$$

$$\exists\alpha' \in A^\infty.(\beta\cdot c\cdot\alpha' \in S \wedge \alpha' \approx_{A_v\cup A_c} \alpha)]$$

Since $(T_\beta, T_{\beta\cdot c}) \in R$, the result follows by coinduction.
Case 2: Assume that $En^\rho_V(T, T_\beta, c)$ does not hold. Then by Theorem B.0.3 it follows that $Adm^\rho_V(T,\beta,c)$ does not hold. Thus $BSIA^\rho_V(T)$ holds trivially.
5.$lrb_V(T,R) \wedge osc_V(T,R) \to BSI_V(T)$: Let $\alpha \in A^\infty$, $\beta \in A^*$, $c \in A_c$ be arbitrary and assume the following hold: $\beta\cdot\alpha \in T$ and w.l.g. $no_{A_c}(\beta)$. Hence there is subtree T_β s.t. $\alpha \in T_\beta$. Clearly $(T_\beta, T_\beta) \in R$ by the definition of osc_V. Because $lrb_V(T_\beta, T_\beta, R)$ it has to be that $test_c(T_\beta)$ and $(T_{\beta\cdot c}, T_\beta) \in R$ hold. A straightforward proof by coinduction (using the fact that $(T_\beta, T_{\beta\cdot c}) \in R$ holds) is in order. □

Lemma 4.5.8. *For all $T \in$ Sys, the following hold:*

1. $SD_V(T) \to BSD_V(T)$ 2. $SIA^\rho_V(T) \to BSIA^\rho_V(T)$
3. $BSD_V(T) \to R_V(T)$ 4. $BSIA^\rho_V(T) \to IA^\rho_V(T)$.

Proof. 1. From the definitions (choose α' to be α). 2. From the definitions (choose α' to be α). 3. Directly from Theorem 4.3.3. 4. From the definitions (choose β' to be β). □

Theorem 4.5.4. *Consider a view (A_v, A_n, A_c) with $A_n = \emptyset$. The following are valid:*

1. *$BSD(T)$ implies there exists a relation $R \subseteq T \times T$ s.t. $lrf_V(T,T,R)$ and $osc_V(T,T,R)$ hold.*
2. *$BSIA^\rho_V(T)$ implies there exists a relation $R \subseteq T \times T$ s.t. $lrbe^\rho_V(T,T,R)$ and $osc_V(T,T,R)$ hold.*

Proof. Similar to Mantel's proof (Theorem 5.4.9 [55]), main difference is the use of coinduction as a proof technique and the different definitions. □

Lemma B.0.3. *Let T be a tree. The following are valid:* 1. *For all subtrees s of T, $En^{\rho}_{V}(T,s,a)$ implies there exists a finite trace $\beta \in A^*$ s.t. $test^*{}_{\beta}(T)$ and $Adm^{\rho}_{V}(T,\beta,a)$ holds.* 2. *For all finite traces $\beta \in A^*$ if $\beta \in T$ and $Adm^{\rho}_{V}(T,\beta,a)$ holds then there is a subtree s of T such that $s = T_{\beta}$ and $En^{\rho}_{V}(T,s,a)$.*

Proof. First proposition: assume $En^{\rho}_{V}(T,s,a)$. By definition there are $\beta,\gamma \in A^*$ such that $test^*{}_{\beta}(T)$, $s = T_{\beta}$, $\gamma \approx_{\rho(V)} \beta$, $test^*{}_{\gamma \cdot a}(T)$. By definition of $test^*$ we have that there is $\gamma \in T$, $\gamma \cdot a \in T$ and $\beta \in T$. Finally $\gamma \approx_{\rho(V)} \beta$ holds and thus $Adm^{\rho}_{V}(T,\beta,a)$. Second proposition: assume $\beta \in T$ and $Adm^{\rho}_{V}(T,\beta,a)$. Because $\beta \in T$ it follows that $test^*{}_{\beta}(T)$. Since $Adm^{\rho}_{V}(T,\beta,a)$, we have a trace $\gamma \in A^*$ s.t. $\gamma \cdot a \in T$ and $\gamma \approx_{\rho(V)} \beta$. Because $\gamma \cdot a \in T$ it follows that $test^*{}_{\gamma \cdot a}(T)$. □

Lemma B.0.4. *For all $T \in$ Sys we have:*

$1.BSD_{\mathcal{HI}}(T) \wedge BSI_{\mathcal{HI}}(T) \rightarrow GNI(T)$ $\quad$ $2.GNI(T) \rightarrow BSD_{\mathcal{HI}}(T)$

$3.GNI(T) \rightarrow BSI_{\mathcal{HI}}(T)$

Proof. $BSD_{\mathcal{HI}}(T) \wedge BSI_{\mathcal{HI}}(T) \rightarrow GNI(T)$: Similar to Mantel's proof [55] (Lemma 4.4.2), but we use coinductive arguments, dictated by the nature of our definitions. Let $x_1 \in A^*, x_2, x_3 \in A^{\infty}$ be arbitrary such that $x_1 \cdot x_2 \in T$, $x_3 \approx_{A \setminus HI} x_2$ and w.l.g. assume that $no_{HI}(x_1)$ holds. By a coinductive argument and the definition of $BSD_{\mathcal{HI}}(T)$, there is a trace $y \in A^{\infty}$ such that $x_1 \cdot y \in T$ and $y \approx_L x_2$. Because $x_3 \approx_{A \setminus HI} x_2$ and $y \approx_L x_2$ it follows that $y \approx_L x_3$. Now, applying the definition of $BSI_{\mathcal{HI}}(T)$ (plus a coinductive argument), the high inputs in x_3 can be inserted in y in a stepwise manner (left to right). The resultant trace is called y_1 and has the properties: $y_1 \approx_L x_3$ and $y_1 \approx_{HI} x_3$. Thus we have shown that there is a trace $x_4 \in A^{\infty}$ with $x_1 \cdot x_4 \in T$ and $x_4 \approx_{L \cup HI} x_3$. This trace is precisely y_1.

$GNI(T) \rightarrow BSD_{\mathcal{HI}}(T)$: The proof is similar to Mantel's proof [55] (Lemma 4.4.2), but again with coinductive arguments. Assume $GNI(T)$ holds. Let $\alpha \in A^{\infty}$, $\beta \in A^*$, $c \in HI$ be arbitrary, such that $\beta \cdot c \cdot \alpha \in T$ and w.l.g. $no_{HI}(\beta)$ holds. Because the assumption is $GNI(T)$ we can apply the definition of GNI with $x_1 = \beta$, $x_2 = c \cdot \alpha$ and $x_3 = \alpha$ and we get: there exists some trace $x_4 \in A^{\infty}$ such that $x_1 \cdot x_4 \in T$ and $x_4 \approx_{L \cup HI} x_3$. We argue that x_4 is the needed trace α' in the definition of $BSD_{\mathcal{HI}}$: firstly, $\beta \cdot \alpha' \in T$ and secondly, $\alpha' \approx_{L \cup HI} \alpha$.

$GNI(T) \rightarrow BSI_{\mathcal{HI}}(T)$: Assume that $GNI(T)$ holds. Let $\alpha \in A^{\infty}$, $\beta \in A^*$, $c \in HI$ be arbitrary such that $\beta \cdot \alpha \in T$ and w.l.g. assume that $no_{HI}(\beta)$ holds. Because $GNI(T)$ holds, we can apply its definition with $x_1 = \beta$, $x_2 = \alpha$ and $x_3 = c \cdot \alpha$. It follows that:

there exists some trace $x_4 \in A^\infty$ such that $x_1 \cdot x_4 \in T$ and $x_4 \approx_{L\cup HI} x_3$. We argue that x_4 is the needed trace $c \cdot \alpha'$ in the definition of $BSI_{\mathcal{H}I}$. First, clearly $\beta \cdot c \cdot \alpha' \in T$ (by the definition of *GNI*). Second, $c \cdot \alpha' \approx_{L\cup HI} c \cdot \alpha$ (by the definition of *GNI*), thus it has to be that $\alpha' \approx_{L\cup HI} \alpha$. □

Theorem 4.5.5. *For all $T \in$ Sys we have $BSD_{\mathcal{H}I}(T) \wedge BSI_{\mathcal{H}I}(T)$ iff $GNI(T)$.*

Proof. Follows directly from Lemma B.0.4. □

Theorem 4.5.6. *For all $T \in$ Sys we have $R_{\mathcal{H}}(T)$ iff $NF(T)$.*

Proof. By mutual implication.
($\Rightarrow$) Let t_0 be an arbitrary trace in T. Since $R_{\mathcal{H}}(T)$ there exists a trace $\gamma \in T$ such that $no_H(\gamma)$ and $t_0 \approx_L \gamma$. Use Lemma B.0.5 to get $\gamma \approx ev_L(t_0)$, hence $\gamma = ev_L(t_0)$. Thus $NF(T)$ holds.
($\Leftarrow$) Assume $NF(T)$. Let t_0 be an arbitrary trace in T. We know that $ev_L(t_0) \in T$ by the assumption. Clearly $no_H(ev_L(t_0))$ (straightforward proof by coinduction). Also $t_0 \approx_L ev_L(t_0)$ by Lemma B.0.6. □

Theorem 4.5.7. *For all $T \in$ Sys we have $R_{\mathcal{H}I}(T)$ iff $GNF(T)$.*

Proof. Follows immediately from the definitions, using view $\mathcal{H}I$. □

Lemma 4.5.9. *For all $T \in$ Sys, the following holds: $PSP(T) \rightarrow (SD_{\mathcal{H}}(T) \wedge SIA^{\rho_A}_{\mathcal{H}}(T))$.*

Proof. First, we prove that $PSP(T) \rightarrow SD_{\mathcal{H}}(T)$. Assume $PSP(T)$. Start with an arbitrary trace $t = \beta \cdot c \cdot \alpha$ and w.l.g. assume that $no_H(\beta)$ holds. Consider trace $\beta \cdot ev_L(\alpha)$. From the first conjunct of the *PSP* definition it follows that $\beta \cdot ev_L(\alpha) \in T$. Now, by the second conjunct of the definition of *PSP* and by a coinductive argument, we can keep adding the high events from α (from left to right) until we have the trace $\beta \cdot \alpha \in T$.

Second, show $PSP(T) \rightarrow SIA^{\rho_A}_{\mathcal{H}}(T)$: Assume $PSP(T)$. Start with an arbitrary trace $t = \beta \cdot \alpha, Adm^{\rho_A}_V(T, \beta, c)$ and w.l.g. assume that $no_H(\beta)$ holds. Clearly $\beta \cdot ev_L(\alpha) \in T$ by the first conjunction of *PSP*. By the second conjunction of *PSP* we have $\beta \cdot c \cdot ev_L(\alpha) \in T$. Finally by the second conjunction of *PSP* and a coinductive argument we have that $\beta \cdot c \cdot \alpha \in T$. □

Lemma 4.5.10. *For all $T \in$ Sys, the following holds: $(R_{\mathcal{H}}(T) \wedge IA^{\rho_A}_{\mathcal{H}}(T)) \rightarrow PSP(T)$.*

Proof. First, note that $R_{\mathcal{H}}(T)$ implies the first conjunct in the definition of *PSP*, namely $\forall t \in T.ev_L(t) \in T$ (straightforward, using Lemma B.0.5). Second, by the assumption $IA^{\rho_A}_{\mathcal{H}}(T)$ and substituting α for α' and β for β' we get that

$$\forall \alpha \in A^{\infty} \forall \beta \in A^{*} \forall h \in H.(\beta \cdot \alpha \in T \wedge test_h(T_\beta)) \rightarrow \beta \cdot h \cdot \alpha \in T$$

Also note that $Adm^{\rho_A}_{\mathcal{H}}(T,\beta,h)$ is equivalent to $test_h(T_\beta)$. Here, we have in fact proven a statement which is stronger. It implies that also it works when $no_H(\alpha)$ holds. □

Theorem 4.5.11. *For all* $T \in \mathsf{Sys}$ *we have* $BSD_{\mathcal{H}}(T) \wedge BSIA^{\rho_A}_{\mathcal{H}}(T)$ *iff* $PSP(T)$.

Proof. By mutual implication.
($\Rightarrow$) From the assumption and Lemma 4.5.8 (3 and 4) we get $BSD_{\mathcal{H}}(T) \wedge BSIA^{\rho_C}_{\mathcal{H}}(T) \rightarrow \mathcal{R}_{\mathcal{H}}(T) \wedge IA^{\rho_A}_{\mathcal{H}}(T)$. Then from Lemma 4.5.10 we get $PSP(T)$, as needed.
($\Leftarrow$) From the assumption and Lemma 4.5.9 we get $PSP(T) \rightarrow SD_{\mathcal{H}}(T) \wedge SIA^{\rho_A}_{\mathcal{H}}(T)$. Then from Lemma 4.5.8 (1 and 2) we get $BSD_{\mathcal{H}}(T) \wedge BSIA^{\rho_C}_{\mathcal{H}}(T)$, as needed. □

Lemma B.0.5. *For all* $t, \gamma \in A^{\infty}$ *we have:* $(no_H(\gamma) \wedge t \approx_L \gamma) \rightarrow \gamma \approx ev_L(t)$.

Proof. By coinduction with hypothesis $CH : \forall t, \gamma \in A^{\infty}.(no_H(\gamma) \wedge t \approx_L \gamma) \rightarrow \gamma \approx ev_L(t)$. Assume $no_H(\gamma)$ and $t \approx_L \gamma$. Next destruct γ and t. If $t = \varepsilon$ it has to be that $\gamma = \varepsilon$ and vice versa. Otherwise, destruct γ as $a \cdot w$ and t as $y \cdot z$: we know that $a \in L$, $w \in A^{\infty}$, $y \in A^{*}$ and $z \in A^{\infty}$. Also $y \rightsquigarrow_L a$ such that $a \in L$ (because $t \approx_L \gamma$). We also have that $no_H(w)$, $w \approx_L z$. By *CH* we get that $aw \approx yz$ □

Lemma B.0.6. *For all* $t \in A^{\infty}$ *we have:* $t \approx_L ev_L(t)$.

Proof. By coinduction. $CH : \forall t \in A^{\infty}.t \approx_L ev_L(t)$. Let t_0 be an arbitrary trace. Destruct t_0 as $v \cdot w$, where $v \in A^{*}$, $w \in A^{\infty}$ such that $v \rightsquigarrow_L a \cdot w$ (by the definition of $\approx_L$ such v exists). Apply the definition of ev_L we get $a \cdot w \approx_L a \cdot ev_L(w)$. Use inversion to get $w \approx_L ev_L(w)$. By the *CH* we get that $t_0 \approx_L ev_L(t_0)$. □

Appendix C

Proofs from Chapter 5

We start by introducing some notation for the following proofs. Let P denote the set of positions, at which R has to move next. Let $W \subseteq P$ be the subset of positions that are R-wins. Finally, let *Force* be the subset of positions from which R can win by eventually entering W. These sets will be defined differently depending on the concrete H'-simulation game.

Proposition 5.3.5. *For any* SIH^2*-simulation game* $G_{\mathsf{SIH}^2}(S,T)$*, where* $S,T \subseteq P_{\Box\Diamond}$*, either player R or player V has a history-free winning strategy.*

Proof. The proof of this proposition has a similar idea and structure to the proof of a similar theorem for equivalence games [82].

Let *filter* : $P_{\Box\Diamond} \to P_{\Box\Diamond}$ be the usual function filtering $\neg p$ events. Such a function is guaranteed to be productive as long as it is called on streams in $P_{\Box\Diamond}$. The set of possible R-positions is

$$P = \{(S',T') \text{ such that } \exists w \in A^* \cdot S \xrightarrow{w} S' \text{ and } T \xrightarrow{\mathit{filter}(w)} T'\}.$$

In other words, P contains the positions in which R moves next and $W \subseteq P$ are the R-win positions. Formally, W can be given as follows: $\forall (S,T) \in W \;.\; \exists S' \;.\; S \xrightarrow{a} S' \wedge T \not\xrightarrow{a}$.

The set *Force* is defined iteratively using ordinals, where λ is some limit ordinal.

$$Force^1 = W$$

$$Force^{\alpha+1} = Force^\alpha \cup \{(S,T) \in P \mid \exists S'.\, S \xrightarrow{a} S' \text{ and } p(a) \text{ and}$$

$$\exists T'.\, T \xrightarrow{f(a)} T' \text{ then } (S',T') \in Force^\alpha \text{ or}$$

$$\exists T'.\, T \xrightarrow{a} T' \text{ and } \neg p(a) \text{ then } (S',T) \in Force^\alpha\}$$

$$Force^\lambda = \bigcup\{Force^\alpha.\alpha < \lambda\}$$

Define *Force* as follows:

$$Force = \bigcup\{Force^\alpha \mid \alpha > 0\}.$$

Let the *rank* of any $(S,T) \in Force$ be the least ordinal α such that $(S,T) \in Force^\alpha$. For each $(S,T) \in Force$, player R has a history-free winning strategy for game $G_{\mathsf{SIH}^2}(S,T)$. This strategy is based on rules of the form: "at any (S',T') choose transition t such that, no matter whether V has to make a move or not, the resulting pair of trees has lower rank". Such a choice for R exists by the definition of *Force*.

Whenever $(S,T) \notin Force$, player V has a history-free winning strategy, namely to simply (play by the rules of the game and) avoid getting into *Force*. The initial pair of trees (S,T) are either in set *Force* or in $P \setminus Force$. Thus one of the players has a history-free winning strategy for $G_{\mathsf{SIH}^2}(S,T)$. □

Theorem 5.3.6 (Correctness of SIH^2-simulation games)**.** *The coinductive predicate* $\mathsf{SIH}^2(S,T)$*, where* $S,T \subseteq P_{\Box\Diamond}$*, holds iff* V *has a history-free winning strategy for* $G_{\mathsf{SIH}^2}(S,T)$.

Proof. ($\Rightarrow$) Assume $\mathsf{SIH}^2(S,T)$. It follows that there is a SIH^2-simulation relation Q (see [63]) such that $(S,T) \in Q$. We can construct a history-free winning strategy for V (using Q) in the following manner: when a move $a \in A$ such that $p(a)$ is made by R at position (S',T'), player V responds with such a move (again a), so that the resulting pair $(S'',T'') \in Q$; otherwise V makes a null move (i.e. no move). More concretely: $(S',T') \in Q$ means that if R moves $S' \xrightarrow{a} S''$ and $p(a)$ then V should choose $T' \xrightarrow{a} T''$ such that and $(S'',T'') \in Q$: such T'' is guaranteed to exist by the definition of Q; if R moves $S' \xrightarrow{a} S''$ and $\neg p(a)$ then V makes a null move and $(S'',T') \in Q$.

($\Leftarrow$) Assume that V has a history-free winning strategy for $G_{\mathsf{SIH}^2}(S,T)$. We show that $\mathsf{SIH}^2(S,T)$ by proving that relation H, given as follows

$$H = \{(S',T') \mid V \text{ has a history-free winning strategy for } G_{\mathsf{SIH}^2}(S',T')\}$$

and including (S,T), is a SIH^2-simulation. This proceeds by coinduction, with hypothesis $(S',T') \in H$. By the definition of H, V has a history-free winning strategy for (S',T'). Suppose $S' \xrightarrow{a} S''$. This is a possible move by R and it follows that either $p(a)$ and V can reply with $T' \xrightarrow{a} T''$ such that $(S'',T'') \in H$ or else $p(a)$ and $(S'',T') \in H$. Other options are not possible, because they would contradict the existence of a history-free winning strategy for V at (S',T'). Apply the coinductive hypothesis and we are done. □

Next, we will present the proof of correctness of IHP checking games. But first, we introduce some propositions (from [82], but generalized to $\mathcal{L}_\mu^k$).

Proposition C.0.7. *The following statements are valid:*

1. *X subsumes X.*
2. *X subsumes Y and Y subsumes Z imply that X subsumes Z.*
3. *X subsumes Y and $X \neq Y$ imply that not Y subsumes X.*

Proof. By straightforward reasoning about the respective sets of formulas. □

For the following proofs, let E_i range over pairs of k-tuples of trees and formulas. Let $E_0 = ((T_0^1,\ldots,T_0^m,\ldots,T_0^k),\Phi_0)$, where $T_0^1 = T^1, T_0^m = T^m, T_0^k = T^k$ and $\Phi_0 = \Phi$. It is known that the following proposition holds:

Proposition C.0.8. *If $(E_0,\Phi_0)\ldots(E_n,\Phi_n)\ldots$ is an infinite length play of the game $HG_\vee(E_0,\Phi_0)$, then there is a unique variable X that*

1. *occurs infinitely often, i.e. for infinitely many j, $X = \Phi_j$ and*
2. *for all Y occurring infinitely often, X subsumes Y.*

Proof. Essentially the same as Stirling's [82]. Proof idea: let $\sigma_1 X_1.\Phi_1,\ldots,\sigma_n X_n.\Phi_n$ be the fixed point subformulae in Φ_0(and hence Φ). The idea is that since every subformula is finite, an infinite play may only occur because some variable from the set $\{X_1,\ldots,X_n\}$ occurs infinitely often. The variable is unique by the principle of transitivity of subsumption (see Proposition C.0.7, rule 2). □

Proposition C.0.9. *The following implications are valid:*

1. *If $E \models_\vee \mu Z.\Psi$, then there exists a least ordinal α s.t. $E \models_\vee \mu Z^\alpha.\Psi$ and for all $\beta < \alpha$, we have $E \not\models_\vee \mu Z^\beta.\Psi$.*

2. *Dually, if* $E \not\models_{\mathsf{V}} \nu Z.\Psi$*, then there exists a least ordinal* α *s.t.* $E \not\models_{\mathsf{V}} \nu Z^{\alpha}.\Psi$ *and for all* $\beta < \alpha$*, we have* $E \models_{\mathsf{V}} \nu Z^{\beta}.\Psi$.

Proof. 1. Assume that $E \models_{\mathsf{V}} \mu Z.\Psi$. For any E, we have that $E \not\models_{\mathsf{V}} \mu Z^{0}.\Psi$. If $E \models_{\mathsf{V}} \mu Z^{\alpha}.\Psi$, then for all $\beta > \alpha$ we have that $E \models_{\mathsf{V}} \mu Z^{\beta}.\Psi$ holds (by monotonicity). Hence, there exists a least ordinal α such that $E \models_{\mathsf{V}} \mu Z^{\alpha}.\Psi$.

2. Assume that $E \not\models_{\mathsf{V}} \nu Z.\Psi$. For any E, we have that $E \models_{\mathsf{V}} \nu Z^{0}.\Psi$. If $E \not\models_{\mathsf{V}} \nu Z^{\alpha}.\Psi$, then for all $\beta > \alpha$ we have that $E \not\models_{\mathsf{V}} \nu Z^{\beta}.\Psi$ holds (by monotonicity). Hence, there exists a least ordinal α such that $E \not\models_{\mathsf{V}} \nu Z^{\alpha}.\Psi$.

□

Proposition C.0.10. *If* $F \models_{\mathsf{V}_n} \Psi$ *then there exists a smallest signature* s *s.t.* $F \models_{\mathsf{V}_n^s} \Psi$.

Proof. Proceed by induction on the structure of Ψ. If the formula is $\Psi = tt$, then the proposition is vacuously true (the signature is a sequence of ordinals of length 0). If $\Psi = ff$, then the signature is 0, because $ff = \mu Z^{0}.\Phi$ for all Φ. The other cases:

- $\Psi = \Phi_1 \wedge \Phi_2$. By the inductive hypothesis, the proposition holds for Φ_1 and Φ_2: let s_1 be the smallest signature of Φ_1 (hence $F \models_{\mathsf{V}_n^{s_1}} \Phi_1$) and s_2 the smallest signature of Φ_2 (hence $F \models_{\mathsf{V}_n^{s_2}} \Phi_2$). The smallest signature for Ψ is thus the (lexicographically) smaller of $s_1 \cdot s_2$ and $s_2 \cdot s_1$, where notation $s_1 \cdot s_2$ denotes the concatenation of s_1 and s_2. Let us denote the concatenated signature s'. Hence, $F \models_{\mathsf{V}_n^{s'}} \Phi_1 \wedge \Phi_2$.

- $\Psi = \langle K \rangle_l \Phi$, where $K \subseteq A$. Let s_1 denote the smallest signature of Φ. The smallest signature of Ψ is s_1, as there are no more fixed points in Ψ than in Φ. By the inductive hypothesis, $E \models_{\mathsf{V}_n^{s_1}} \Phi$ (also by IH $F \models_{\mathsf{V}_n} \Psi$) and hence $F \models_{\mathsf{V}_n^{s_1}} \Psi$ (because signature does not change by $\langle a \rangle$).

- $\Psi = \Phi_1 \vee \Phi_2$. Let s_1 be the smallest signature of Φ_1 and s_2 the smallest signature of Φ_2. If both Φ_1 and Φ_2 hold, then the smallest signature of Ψ is the (lexicographically) smaller of s_1 and s_2 (let us call it s_i). Otherwise, if only one of Φ_i ($i \in \{1,2\}$) holds, then the signature is s_i. By the IH, $F \models_{\mathsf{V}}^{s_i} \Phi_i$ and this implies $F \models_{\mathsf{V}}^{s_i} \Psi$.

- $\Psi = [K]_l \Phi$, where $K \subseteq A$. Let s_1 denote the smallest signature of Φ. The smallest signature of Ψ is s_1, as there are no more fixed points in Ψ than in Φ. By IH $E \models_{\mathsf{V}_n^{s_1}} \Phi$. Because $F \models_{\mathsf{V}_n} \Psi$, we have that $F \models_{\mathsf{V}_n^{s_1}} \Psi$.

- $\Psi = \nu Z.\Phi$. Let s_1 denote the smallest signature of Φ. The smallest signature of Ψ is s_1, as there are no more least fixed points in Ψ than in Φ. By IH $E \models_{\mathsf{V}_n^{s_1}} \Phi$. Because $F \models_{\mathsf{V}_n} \Psi$ and $F = E$, we have that $F \models_{\mathsf{V}_n^{1}} \Psi$.

- $\Psi = \mu Z.\Phi$. Let s_1 denote the smallest signature of Φ. By Proposition C.0.9, we know that there is some least ordinal α, s.t. $\mu Z^{\alpha}.\Phi$ is true. Hence, the least signature of Ψ is $\alpha \cdot s_1$. By IH $E \models_{\mathsf{V}_n^{s_1}} \Phi$. Because $F \models_{\mathsf{V}_n} \Psi$ and $F = E$, we have that $F \models_{\mathsf{V}_n^{\alpha \cdot s_1}} \Psi$.

□

Proposition C.0.11. *If $F \not\models_{\mathsf{V}_n} \Psi$ then there exists a smallest signature s s.t. $F \not\models_{\mathsf{V}_n^s} \Psi$.*

Proof. Similar to the proof of Proposition C.0.10, using the duality of least and greatest fixed point. □

Theorem 5.4.4. The following equivalences are valid:

1. $(T^1, \ldots, T^m, \ldots, T^k) \models_{\mathsf{V}} \Phi$ *iff player V has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1, \ldots, T^m, \ldots, T^k), \Phi)$.
2. $(T^1, \ldots, T^m, \ldots, T^k) \not\models_{\mathsf{V}} \Phi$ *iff player R has a history-free winning strategy for* $HG_{\mathsf{V}}((T^1, \ldots, T^m, \ldots, T^k), \Phi)$.

Proof. Statement 1.
The proof goes by lifting Stirling's proof for property checking games in the modal mu-calculus to IHP checking games in $\mathcal{L}_{\mu}^{k}$ [82] (Section 6.3).
Proof strategy: show that whenever V has a turn, she can always choose a *true* next position.

($\Rightarrow$) Assume that $(T^1, \ldots, T^m, \ldots, T^k) \models \Phi$. This is equivalent to having the formula $(T_0^1, \ldots, T_0^m, \ldots, T_0^k) \models \Phi_0$ hold. Assume that $\sigma_1 Z_1.\Psi_1, \ldots, \sigma_n Z_n.\Psi_n$ are the fixed point subformulae of Φ_0 in decreasing order of size. Let $\mathcal{T}$ be the set of k-tuples of trees reachable from $(T_0^1, \ldots, T_0^m, \ldots, T_0^k)$. Then define valuations $\mathsf{V}_0, \ldots, \mathsf{V}_n$ as follows:

$$\begin{aligned} \mathsf{V}_0 &= \mathsf{V} \\ \mathsf{V}_{i+1} &= \mathsf{V}_i[\mathtt{T}_{i+1}/Z_{i+1}], \end{aligned}$$

where $\mathtt{T}_{i+1} = \{T \in \mathcal{T} : T \models_{\mathsf{V}_i} \sigma_{i+1} Z_{i+1}.\Psi_{i+1}\}$. Intuitively, valuation V_n captures the meaning of all bound variables. In other words, some position $((T_i^1, \ldots, T_j^m, \ldots, T_l^k), \Psi)$ is true whenever the formula $(T_i^1, \ldots, T_j^m, \ldots, T_l^k) \models_{\mathsf{V}_n} \Psi$ holds. Hence we have that $(T_0^1, \ldots, T_0^m, \ldots, T_0^k) \models_{\mathsf{V}_n} \Phi_0$, because formula $(T_0^1, \ldots, T_0^m, \ldots, T_0^k) \models \Phi_0$ holds by the assumption.

Next, a refined valuation, using the smallest least fixed point approximants that make a certain game configuration true, is defined. Let $\mu Y_1.\Psi'_1,\ldots,\mu Y_q.\Psi'_q$ be all least fixed point formulae in Φ_0 in decreasing order of size. Define a signature as a sequence of ordinals $\alpha_1,\ldots,\alpha_q$ and assume lexicographic ordering on signatures. Given some signature $s=\alpha_1,\ldots,\alpha_q$, its associated valuation V^s_n is defined as:

$$\begin{aligned}\mathsf{V}^s_0 &= \mathsf{V}\\ \mathsf{V}^s_{i+1} &= \mathsf{V}^s_i[\mathtt{T}_{i+1}/Z_{i+1}].\end{aligned}$$

The only difference is that now $\mathtt{T}_{i+1}$ depends on the fixed point:

1. When $\sigma_{i+1}=\nu$ then $\mathtt{T}_{i+1}=\{T\in\mathcal{T}:T\models_{\mathsf{V}^s_i}\sigma_{i+1}Z_{i+1}.\Psi_{i+1}\}$.
2. When $\sigma_{i+1}=\mu Y_j$ then $\mathtt{T}_{i+1}=\{T\in\mathcal{T}:T\models_{\mathsf{V}^s_i}\sigma Y_j^{\alpha_j}.\Psi'_j\}$.

The existence of a smallest signature s such that $(T^1_i,\ldots,T^m_j,\ldots,T^k_l)\models_{\mathsf{V}^s_n}\Psi$ if $(T^1_i,\ldots,T^m_j,\ldots,T^k_l)\models_{\mathsf{V}_n}\Psi$ follows from Proposition C.0.10. For a true configuration $((T^1_i,\ldots,T^m_j,\ldots,T^k_l),\Psi)$, define a *signature* to be the least s such that $(T^1_i,\ldots,T^m_j,\ldots,T^k_l)\models_{\mathsf{V}^s_n}\Psi$.

Now, we are ready to define a history-free winning strategy for player V. This is done by case analysis of true player V positions. If the position is $(F,\Psi_1\vee\Psi_2)$, then it has to be that $F\models_{\mathsf{V}^s_n}\Psi_1\vee\Psi_2$. It has to be that one of the conjuncts (Ψ_i) holds. Add the rule for V: "at position $(F,\Psi_1\vee\Psi_2)$ choose (F,Ψ_i)". For the case when the position is $(F,\langle K\rangle_l\Psi)$ and K is a set of events ($K\subseteq A$), we proceed in a similar way. Let s be the position's signature and thus $F\models_{\mathsf{V}^s_n}\langle K\rangle_l\Psi$. Thus, there is a transition $F\xrightarrow{a}_l F'$ (where $a\in K$) such that $F'\models_{\mathsf{V}^s_n}\Psi$. Hence, the rule for player V is "at position $(F,\langle K\rangle_l\Psi)$ choose (F',Ψ)". These rules result in a history-free strategy. It is only left to show that this strategy is winning.

Assume the opposite, i.e. that R can win. It is clear (by the assumption that formula $(T^1,\ldots,T^m,\ldots,T^k)\models\Phi$ holds) that the start position is true. We will show that if player V uses the strategy defined above, a false position can never be reached. Note that a signature will be identified with each position. The initial position is (E_0,Φ_0) and $E_0\models^{s_0}_{\mathsf{V}_n}\Phi_0$. Assume that (E_m,Φ_m) is the current position in the play and $E_m\models^{s_m}_{\mathsf{V}_n}\Phi_m$. If the position is final, then by the rules V is the winner. Otherwise, the play is not complete and we have to explore how the game is extended into (E_{m+1},Φ_{m+1}) s.t. $E_{m+1}\models^{s_m+1}_{\mathsf{V}_n}\Phi_{m+1}$ (this is the "step" in the proof):

- If $\Phi_m = \Psi_1 \wedge \Psi_2$, then R choses Ψ_i and the next position (E_{m+1}, Φ_{m+1}) has to be true, s.t. $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$, and $E_{m+1} \models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. Note that $s_{m+1} \leq s_m$ (informally, s_{m+1} cannot be greater than s_m for sure).
- If $\Phi_m = [K]_l\Psi$, then R choses an $a \in K$ such that $E \xrightarrow{a}_l E'$ and the next position (E_{m+1}, Ψ) has to be true, where $E_{m+1} = E'$ and $\Phi_{m+1} = \Psi$, and $E_{m+1} \models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. Again note that again $s_{m+1} \leq s_m$.
- If $\Phi_m = \Psi_1 \vee \Psi_2$, then V uses the strategy to chose the next position. This preserves the truth as s_m is the signature used to define the strategy. Note that $s_{m+1} \leq s_m$.
- If $\Phi_m = \langle K \rangle_l \Psi$, then V uses the strategy to chose $a \in K$ such that $E \xrightarrow{a}_l E'$ and the next position (E_{m+1}, Ψ) has to be true, where $E_{m+1} = E'$ and $\Phi_{m+1} = \Psi$, and $E_{m+1} \models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. This preserves the truth as s_m is the signature used to define the strategy. Note that again $s_{m+1} \leq s_m$.
- If $\Phi_m = \sigma_i Z_i.\Psi_i$, then the next configuration is the true position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = Z_i$.
- If $\Phi_m = Z_i$ and $\sigma_i = \nu$, then the next configuration is the true position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$ and $s_{m+1} = s_m$ (see proof of Proposition C.0.10, not related to the greatest fixed point).
- If $\Phi_m = Z_i$ and $\sigma_i = \mu$, then the next configuration is the true position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$ and $s_{m+1} < s_m$, since the fixed point has been in effect unfolded.

So far, we have seen that V wins finite plays. Now, consider an infinite play. Proposition C.0.8 guarantees the existence of a unique Z_i, which occurs infinitely often and subsumes any other Z_j occurring infinitely often. Consider the infinite play from position (E_k, Φ_k) on, i.e. play $(E_k, \Phi_k)\ldots$, such that any occurrence of Z_j is subsumed by Z_i. Let $k_1, k_2, \ldots$ mark the positions of Z_i in $(E_k, \Phi_k)\ldots$. It is impossible for Z_i to correspond to a least fixed point formula. Otherwise, there would be a strictly decreasing sequence of signature sets $s_{k1} > s_{k2} > \ldots$. But that is impossible and hence Z_i corresponds to a greatest fixed point formula. Hence V wins infinite plays and the strategy is indeed *winning*.

($\Leftarrow$) By induction on the structure of the formula Φ. Assume that V has a history-free winning strategy for $HG_{\mathsf{V}}(E, \Phi)$.

Base cases: If $\Phi_n = tt$ we are done. $\Phi_n = ff$ is impossible, contradicts the existence of a winning strategy for V. If $\Phi = Z$ and $E \in V(Z)$ we are done. $\Phi = Z$ and $E \notin V(Z)$ is impossible (contradiction). Inductive case:

- $\Phi = \Psi_1 \wedge \Psi_2$. From the assumption it follows that V has a history-free winning strategy for $HG_{\vee}(E, \Psi_1)$ and $HG_{\vee}(E, \Psi_2)$. By the inductive hypothesis $E \models \Psi_1$ and $E \models \Psi_2$. Hence $E \models \Phi$.

- If $\Phi = [K]_l\Psi$. By the assumption (that V has a history-free winning strategy for $HG_{\vee}(E, \Phi)$), it follows that for all $a \in K$ and respectively E' s.t. $E \xrightarrow{a}_l E'$ that R potentially chooses, V has a winning strategy for $HG_{\vee}(E', \Psi)$. Applying the inductive hypothesis, we have that $E' \models \Psi$. This means that for all $a \in K$ s.t. $E \xrightarrow{a}_l E'$, $E' \models \Psi$. Hence $E \models [K]_l\Psi$.

- $\Phi = \Psi_1 \vee \Psi_2$. By the assumption, it follows that V has a winning strategy for $HG_{\vee}(E, \Psi_i)$, where $i \in \{1, 2\}$. By the inductive hypothesis $E \models \Psi_i$. Hence $E \models \Phi$.

- If $\Phi = \langle K \rangle_l \Psi$. By the assumption, V has a winning strategy for $HG_{\vee}(E, \Phi)$. Hence, there is some $a \in K$ (according to the winning strategy) and E' s.t. $E \xrightarrow{a}_l E'$ and V has a winning strategy for $HG_{\vee}(E', \Psi)$. Because V has a winning strategy for $HG_{\vee}(E', \Psi)$ and by the induction hypothesis, it follows that $E' \models \Psi$. Hence, $E \models \langle K \rangle_l \Psi$.

- If $\Phi = \sigma Z.\Psi_i$. By the assumption, V has a winning strategy for $HG_{\vee}(E, \Phi)$. Hence, V has a winning strategy for $HG_{\vee}(E, Z)$. By the inductive hypothesis we have that $E \models Z$. Hence, $E \models \Phi$.

- If $\Phi = Z_i$. By the assumption, V has a winning strategy for $HG_{\vee}(E, \Phi)$. Hence, V has a winning strategy for $HG_{\vee}(E, \Psi_i)$. By the inductive hypothesis we have that $E \models \Psi_i$. Hence, $E \models \Phi$.

□

Recall the second part of Theorem 5.4.4: $(T^0, \ldots, T^m, \ldots, T^k) \not\models \Phi$ *iff player R has a history-free winning strategy for* $HG_{\vee}((T^0, \ldots, T^m, \ldots, T^k), \Phi)$. This part of the proof is dual to the one presented.

Proof. Statement 2.
The proof is dual to the one from *Statement* 1.
Proof strategy: to illustrate the existence of a history-free winning strategy for R, we show that player R can always move so as to preserve *false* configurations when going to the next position.

($\Rightarrow$) Assume that $(T^0, \ldots, T^m, \ldots, T^k) \not\models \Phi$. This can be written as
$(T_0^0, \ldots, T_0^m, \ldots, T_0^k) \not\models \Phi_0$. Let $\sigma_1 Z_1.\Psi_1, \ldots, \sigma_n Z_n.\Psi_n$ be the fixed point subformulae

of Φ_0 in decreasing order of size. Let $\mathcal{T}$ be the set of k-tuples of trees reachable from $(T_0^0,\ldots,T_0^m,\ldots,T_0^k)$. Then define valuations $\mathsf{V}_0,\ldots,\mathsf{V}_n$ as follows:

$$\begin{aligned}\mathsf{V}_0 &= \mathsf{V}\\ \mathsf{V}_{i+1} &= \mathsf{V}_i[\mathtt{T}_{i+1}/Z_{i+1}],\end{aligned}$$

where $\mathtt{T}_{i+1}=\{T\in\mathcal{T} : T\models_{\mathsf{V}_i}\sigma_{i+1}Z_{i+1}.\Psi_{i+1}\}$. Intuitively, valuation V_n captures the meaning of all bound variables. In other words, some position given as $((T_i^0,\ldots,T_j^m,\ldots,T_l^k),\Psi)$ is *false* whenever the formula $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)\not\models_{\mathsf{V}_n}\Psi$ holds. Hence we have that $(T_0^0,\ldots,T_0^m,\ldots,T_0^k)\not\models_{\mathsf{V}_n}\Phi_0$ by the assumption.

Next, a refined valuation, using the greatest fixed point approximants that make a certain game configuration false, is defined. Let $\nu Y_1.\Psi'_1,\ldots,\nu Y_q.\Psi'_q$ be all greatest fixed point formulas in Φ_0 in decreasing order of size. Define a signature as a sequence of ordinals $\alpha_1,\ldots,\alpha_q$ and assume lexicographic ordering on these. Given some signature $s=\alpha_1,\ldots,\alpha_q$, its associated valuation V_n^s is defined as before:

$$\begin{aligned}\mathsf{V}_0^s &= \mathsf{V}\\ \mathsf{V}_{i+1}^s &= \mathsf{V}_i^s[\mathtt{T}_{i+1}/Z_{i+1}].\end{aligned}$$

The only difference is that now $\mathtt{T}_{i+1}$ depends on the fixed point:

1. When $\sigma_{i+1}=\mu$ then $\mathtt{T}_{i+1}=\{T\in\mathcal{T} : T\models_{\mathsf{V}_i^s}\sigma_{i+1}Z_{i+1}.\Psi_{i+1}\}$.
2. When $\sigma_{i+1}=\nu Y_j$ then $\mathtt{T}_{i+1}=\{T\in\mathcal{T} : T\models_{\mathsf{V}_i^s}\sigma Y_j^{\alpha_j}.\Psi'_j\}$.

The existence of a smallest signature s such that $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)\not\models_{\mathsf{V}_n^s}\Psi$ if $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)\not\models_{\mathsf{V}_n}\Psi$ follows from Proposition C.0.11. For a *false* configuration $((T_i^0,\ldots,T_j^m,\ldots,T_l^k),\Psi)$, define a *signature* to be the least s such that $(T_i^0,\ldots,T_j^m,\ldots,T_l^k)\not\models_{\mathsf{V}_n^s}\Psi$.

Now, we are ready to define a history-free winning strategy for player F. This is done by case analysis of *false* player F positions. If the position is $(E,\Psi_1\wedge\Psi_2)$, then it has to be that $E\not\models_{\mathsf{V}_n^s}\Psi_1\wedge\Psi_2$. Choose one of the conjuncts (Ψ_i which is false, at least one is guaranteed to be false) and add the rule "at position $(E,\Psi_1\wedge\Psi_2)$ choose (E,Ψ_i), where $i\in\{1,2\}$)". For the case when the position is $(E,[K]_l\Psi)$ and K is a set of events ($K\subseteq A$), we proceed in a similar way. Let s be the position's signature and thus $E\not\models_{\mathsf{V}_n^s}[K]_l\Psi$. Thus, there is some $a\in K$ s.t. $E\xrightarrow{a}_l E'$ and $E'\not\models_{\mathsf{V}_n^s}\Psi$. Hence, the rule for player F is "at position $(E,[K]_l\Psi)$ choose (E',Ψ)". These rules result in a history-free strategy. It is only left to show that the strategy is winning.

Assume the opposite, i.e. assume that V can win. It is clear (by the assumption) that the start position is false. We will show that if player R uses the strategy defined above, a true position can never be reached. Note that a signature is identified with each position. The initial position is (E_0, Φ_0) and $E_0 \not\models^{s_0}_{\mathsf{V}_n} \Phi_0$. Assume that (E_m, Φ_m) is the current position in the play and $E_m \not\models^{s_m}_{\mathsf{V}_n} \Phi_m$. If the position is final, then by the rules R is the winner. Otherwise, the play is not complete and we have to explore how the game is extended into (E_{m+1}, Φ_{m+1}):

- If $\Phi_m = \Psi_1 \vee \Psi_2$ (is false), then both Ψ_1 and Ψ_2 are false. Then V choses Ψ_i and the next position (E_{m+1}, Φ_{m+1}) has to be false, where $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$, and $E_{m+1} \not\models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. Note that $s_{m+1} \leq s_m$ (informally s_{m+1} cannot be greater than s_m for sure).
- If $\Phi_m = \langle K \rangle_l \Psi$, then V choses an $a \in K$ such that $E \xrightarrow{a}_l E'$ and the next position (E_{m+1}, Ψ) has to be false, where $E_{m+1} = E'$ and $\Phi_{m+1} = \Psi$, and $E_{m+1} \not\models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. Note that again $s_{m+1} \leq s_m$.
- If $\Phi_m = \Psi_1 \wedge \Psi_2$, then R uses the strategy to chose the next position. This preserves the falsehood as s_m is the same signature as when defining the strategy. Note that $s_{m+1} \leq s_m$.
- If $\Phi_m = [K]_l \Psi$, then R uses the strategy to chose a specific $a \in K$ such that $E \xrightarrow{a}_l E'$ and the next position (E_{m+1}, Ψ) has to be false, where $E_{m+1} = E'$ and $\Phi_{m+1} = \Psi$, and $E_{m+1} \not\models^{s_m+1}_{\mathsf{V}_n} \Phi_{m+1}$. This preserves the falsehood as s_m is the same signature as when defining the strategy. Note that again $s_{m+1} \leq s_m$.
- If $\Phi_m = \sigma_i Z_i . \Psi_i$, then the next configuration is the false position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = Z_i$. "The signature may increase if $\sigma_i = \nu$".
- If $\Phi_m = Z_i$ and $\sigma_i = \mu$, then the next configuration is the false position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$ and $s_{m+1} = s_m$.
- If $\Phi_m = Z_i$ and $\sigma_i = \nu$, then the next configuration is the false position (E_{m+1}, Φ_{m+1}), where $E_{m+1} = E_m$ and $\Phi_{m+1} = \Psi_i$ and $s_{m+1} < s_m$, since the fixed point has been in effect unfolded.

So far, we have seen that F wins finite plays. Now, consider an infinite play. Proposition C.0.8 guarantees the existence of a unique Z_i, which occurs infinitely often and subsumes any other Z_j occurring infinitely often. Consider the infinite play from position (E_k, Φ_k) on, i.e. play $(E_k, \Phi_k) \ldots$, such that any occurrence of Z_j is subsumed by Z_i. Let $k_1, k_2, \ldots$ be the positions in $(E_k, \Phi_k) \ldots$ where Z_i occurs. It is impossible for Z_i to correspond to a greatest fixed point formula. Otherwise, there would be a strictly decreasing sequence of signature sets $s_{k1} > s_{k2} > \ldots$. But that is impossible

and hence Z_i corresponds to a least fixed point formula. Note that the other direction is dual to the respective part of the proof of *Statement* 1.

□

Bibliography

[1] Peter Aczel. An introduction to inductive definitions. In K. Jon Barwise, editor, *Handbook of Mathematical Logic*, pages 739–782. North-Holland, Amsterdam, 1977. Cited on page 79.

[2] Peter Aczel. *Non-well-founded Sets*. Number 14 in Lecture Notes. Center for the Study of Language and Information, Stanford University, 1988. Cited on pages 65 and 79.

[3] Jiri Adámek and Hans-E. Porst. On tree coalgebras and coalgebra presentations. *Theoretical Computer Science*, 311(1-3):257–283, 2004. Cited on page 37.

[4] Johan Agat. Transforming out timing leaks. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '00, pages 40–53, New York, NY, USA, 2000. ACM. Cited on page 50.

[5] Bowen Alpern and Fred B. Schneider. Defining liveness. Technical report, Ithaca, NY, USA, 1984. Cited on pages 2, 18, and 48.

[6] Rajeev Alur, Pavol Černý, and Swarat Chaudhuri. Model Checking on Trees with Path Equivalences. In *TACAS 2007*, volume 4424 of *Lecture Notes in Computer Science*, pages 664–678. Springer, 2007. Cited on page 118.

[7] Rajeev Alur, Pavol Černý, and Steve Zdancewic. Preserving Secrecy Under Refinement. In *ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming*, volume 4052 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 2006. Cited on pages 4, 118, and 149.

[8] Henrik Reif Andersen. A Polyadic Modal μ-Calculus. Technical Report 1994-145, Technical University of Denmark (DTU), 1994. Cited on pages 7, 8, 10, 13, 30, 32, 96, 97, 103, 121, 127, and 128.

[9] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. In *CSFW '04: Proceedings of the 17th IEEE workshop on Computer Security Foundations*, page 100, Washington, DC, USA, 2004. IEEE Computer Society. Cited on pages 60 and 118.

[10] Yves Bertot and Ekaterina Komendantskaya. Inductive and Coinductive Components of Corecursive Functions in Coq. *CoRR*, abs/0807.1524, 2008. Cited on page 68.

[11] Aaron Bohannon, Benjamin C. Pierce, Vilhelm Sjöberg, Stephanie Weirich, and Steve Zdancewic. Reactive noninterference. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 79–90, New York, NY, USA, 2009. ACM. Cited on pages 21, 22, 59, and 92.

[12] Julian Bradfield, Javier Esparza, and Angelika Mader. An effective tableau system for the linear time μ-calculus. In Friedhelm Meyer and Burkhard Monien, editors, *Automata, Languages and Programming*, volume 1099 of *Lecture Notes in Computer Science*, pages 98–109. Springer Berlin Heidelberg, 1996. Cited on page 119.

[13] Julian Bradfield and Colin Stirling. Modal mu-calculi. In *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007. Cited on pages 7 and 39.

[14] David Clark, Sebastian Hunt, and Pasquale Malacaria. A static analysis for quantifying information flow in a simple imperative language. *Journal of Computer Security*, 15(3):321–371, August 2007. Cited on page 150.

[15] Edmund M. Clarke, E. Allen Emerson, and A. Prasad Sistla. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986. Cited on page 2.

[16] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, Cambridge, MA, USA, 1999. Cited on page 2.

[17] Michael R. Clarkson, Andrew C. Myers, and Fred B. Schneider. Quantifying information flow with beliefs. *Journal of Computer Security*, 17(5):655–701, October 2009. Cited on page 150.

[18] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. In *CSF '08: Proceedings of the 2008 21st IEEE Computer Security Foundations Symposium*, pages 51–65, Washington, DC, USA, 2008. IEEE Computer Society. Cited on pages 2, 3, 5, 8, 9, 13, 18, 19, 35, 39, 41, 57, 58, 64, 93, 116, 118, and 147.

[19] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18:1157–1210, September 2010. Cited on pages 2, 3, 4, 10, 13, 18, 35, 39, 57, 58, 59, 64, 93, 147, 148, and 149.

[20] Ádám Darvas, Reiner Hähnle, and David Sands. A theorem proving approach to analysis of secure information flow. In Dieter Hutter and Markus Ullmann, editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 193–209. Springer, 2005. Cited on pages 60 and 118.

[21] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 2002. Cited on page 13.

[22] Dominique Devriese and Frank Piessens. Noninterference through secure multi-execution. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP 10, pages 109–124, Washington, DC, USA, 2010. IEEE Computer Society. Cited on page 150.

[23] Deepak D'Souza, Raveendra Holla, K. R. Raghavendra, and Barbara Sprick. Model-checking trace-based information flow properties. *Journal of Computer Security*, 19:101–138, January 2011. Cited on pages 60 and 118.

[24] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science (FOCS'91)*, pages 368–377. IEEE Computer Society Press, October 1991. Cited on page 144.

[25] Diana Fischer, Erich Grädel, and Łukasz Kaiser. Model Checking Games for the Quantitative mu-Calculus. *Theory Comput. Syst.*, 47(3):696–719, 2010. Cited on page 150.

[26] Riccardo Focardi and Roberto Gorrieri. A taxonomy of security properties for process algebras. *Journal of Computer Security*, 3(1):5–34, 1995. Cited on pages 20 and 60.

[27] Wan Fokkink. *Introduction to Process Algebra*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 2000. Cited on page 57.

[28] Oliver Friedmann. An exponential lower bound for the latest deterministic strategy iteration algorithms. *Logical Methods in Computer Science*, 7(3), 2011. Cited on page 144.

[29] Oliver Friedmann and Martin Lange. Solving parity games in practice. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis*, ATVA '09, pages 182–196, Berlin, Heidelberg, 2009. Springer-Verlag. Cited on pages 10, 34, 121, 126, 130, 131, and 144.

[30] Oliver Friedmann and Martin Lange. A Solver for Modal Fixpoint Logics. *Electron. Notes Theor. Comput. Sci.*, 262:99–111, May 2010. Cited on pages 121 and 127.

[31] Joseph A. Goguen and José Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982. Cited on page 2.

[32] Joseph A. Goguen and José Meseguer. Unwinding and Inference Control. *IEEE Symposium on Security and Privacy*, pages 75–86, 1984. Cited on pages 2, 4, 22, and 59.

[33] Erich Grädel. Back and Forth Between Logics and Games. In *Lectures in Game Theory for Computer Scientists*, pages 99–145. Springer, 2011. Cited on page 98.

[34] Jan Friso Groote, Aad Mathijssen, Michel Reniers, Yaroslav Usenko, and Muck van Weerdenburg. The Formal Specification Language mCRL2. In Ed Brinksma, David Harel, Angelika Mader, Perdita Stevens, and Roel Wieringa, editors, *Methods for Modelling Software Systems (MMOSS)*, number 06351 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2007. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. Cited on pages 10, 121, and 127.

[35] Gopal Gupta, Neda Saeedloei, Brian DeVries, Richard Min, Kyle Marple, and Feliks Kluźniak. Infinite computation, co-induction and computational logic. In *Proceedings of the 4th international conference on Algebra and coalgebra in computer science*, CALCO'11, pages 40–54, Berlin, Heidelberg, 2011. Springer-Verlag. Cited on page 64.

[36] Reiner Hähnle, Jing Pan, Philipp Rümmer, and Dennis Walter. Integration of a security type system into a program logic. In *Proceedings of the 2nd international conference on Trustworthy global computing*, TGC'06, pages 116–131, Berlin, Heidelberg, 2007. Springer-Verlag. Cited on page 60.

[37] J. Thomas Haigh and William D. Young. Extending the Noninterference Version of MLS for SAT. *IEEE Transactions on Software Engineering*, 13(2):141–150, February 1987. Cited on pages 4 and 22.

[38] I. Hasuo, K. Cho, T. Kataoka, and B. Jacobs. Coinductive predicates and final sequences in a fibration. In *To appear at: Mathematical Foundations of Program Semantics (MFPS 2013)*, 2013. Cited on pages 7, 8, and 147.

[39] Marieke Huisman and Henri-Charles Blondeel. Model-checking secure information flow for multi-threaded programs. In S. Mödersheim and C. Palamadessi, editors, *Proceedings of Theory of Security and Applications, TOSCA 2011, Saarbruecken, Germany*, volume 6993 of *Lecture Notes in Computer Science*, pages 148–165, Berlin, 2011. Springer Verlag. Cited on page 118.

[40] James W. Gray III. Toward a Mathematical Foundation for Information Flow Security. In *IEEE Symposium on Security and Privacy*, pages 21–35, 1991. Cited on page 150.

[41] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 117–123. ACM/SIAM, 2006. Cited on page 144.

[42] Gil Kalai. Linear programming, the simplex algorithm and simple polytopes. *Mathematical Programming*, 79:217–233, 1997. Cited on page 144.

[43] Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983. Cited on pages 2, 4, 10, 30, and 122.

[44] Dexter Kozen and Nicholas Ruozzi. Applications of metric coinduction. In Till Mossakowski, Ugo Montanari, and Magne Haveraaen, editors, *CALCO*, volume 4624 of *Lecture Notes in Computer Science*, pages 327–341. Springer, 2007. Cited on pages 58 and 150.

[45] Dexter Kozen and Alexandra Silva. Practical coinduction. Technical Report 1813/30510, Cornell University, 2012. Cited on pages 7, 8, and 147.

[46] Leslie Lamport. The temporal logic of actions. *ACM Transactions On Programming Languages And Systems*, 16(3):872–923, May 1994. Cited on page 149.

[47] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973. Cited on page 66.

[48] Marina Lenisa. From set-theoretic coinduction to coalgebraic coinduction: some results, some problems. *Electronic Notes in Theoretical Computer Science*, 19, 1999. Cited on page 52.

[49] Leonid Libkin. *Elements Of Finite Model Theory (Texts in Theoretical Computer Science. An Eatcs Series)*. Springer Verlag, 2004. Cited on page 27.

[50] Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Transactions on Information Systems Security*, 12(3):19:1–19:41, January 2009. Cited on page 150.

[51] Zohar Manna and Amir Pnueli. *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA, 1992. Cited on page 2.

[52] Heiko Mantel. Possibilistic Definitions of Security - An Assembly Kit. In *Proceedings of the 13th IEEE Workshop on Computer Security Foundations*, pages 185–199, Washington, DC, USA, 2000. IEEE Computer Society. Cited on pages 22, 71, 78, and 93.

[53] Heiko Mantel. Unwinding possibilistic security properties. In *Proceedings of the 6th European Symposium on Research in Computer Security*, pages 238–254, London, UK, 2000. Springer-Verlag. Cited on pages 60 and 118.

[54] Heiko Mantel. Preserving Information Flow Properties under Refinement. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 78–91, Oakland, CA, USA, May 14–16 2001. IEEE Computer Society. Cited on page 149.

[55] Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, Saarbrücken, Germany, July 2003. Cited on pages iii, v, xvii, 4, 5, 9, 13, 19, 22, 23, 24, 63, 65, 67, 71, 72, 74, 75, 76, 77, 78, 79, 80, 81, 82, 86, 89, 92, and 170.

[56] Jiří Matoušek, Micha Sharir, and Emo Welzl. A subexponential bound for linear programming. In *Proceedings of the eighth annual symposium on Computational geometry*, SCG '92, pages 1–8, New York, NY, USA, 1992. ACM. Cited on page 144.

[57] René Mazala. Automata logics, and infinite games. chapter Infinite games, pages 23–38. Springer-Verlag New York, Inc., New York, NY, USA, 2002. Cited on pages 33 and 34.

[58] Daryl McCullough. Specifications for multi-level security and a hook-up. *IEEE Symposium on Security and Privacy*, 0:161–166, 1987. Cited on pages 23, 70, and 93.

[59] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proceedings of the 1994 IEEE Symposium on Security and Privacy*, SP '94, pages 79–93, Washington, DC, USA, 1994. IEEE Computer Society. Cited on page 23.

[60] John McLean. A general theory of composition for a class of possibilistic properties. *Software Engineering, IEEE Transactions on*, 22(1):53 –67, jan 1996. Cited on pages 42, 45, and 47.

[61] Jonathan Millen. Unwinding Forward Correctability. In *In Proceedings of the Computer Security Foundations Workshop*, pages 2–10. IEEE, 1994. Cited on pages 4 and 22.

[62] Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties. In *Proceedings of the 17th Nordic Conference on Secure IT Systems*, volume 7617 of *Lecture Notes in Computer Science*, pages 121–136. Springer, October 2012. Cited on page 65.

[63] Dimiter Milushev and Dave Clarke. Towards Incrementalization of Holistic Hyperproperties. In *Proceedings of the First International Conference on Principles of Security and Trust*, volume 7215 of *Lecture Notes in Computer Science*, pages 329–348. Springer, March 2012. Cited on pages 4, 6, 36, 43, 82, 88, 118, 167, and 174.

[64] Dimiter Milushev and Dave Clarke. Decidable incremental hyperproperty logics and model checking via games. Submitted for publication, April 2013. Cited on page 96.

[65] Dimiter Milushev and Dave Clarke. Incremental hyperproperty model checking via games. Submitted for publication, May 2013. Cited on page 96.

[66] Milad Niqui and Jan Rutten. Coinductive predicates as final coalgebras. In Ralph Matthes and Tarmo Uustalu, editors, *6th Workshop on Fixed Points in Computer Science, FICS 2009, Coimbra, Portugal, 12–13 September 2009. Proceedings.*, pages 79–85, 2009. Cited on pages 7, 8, 45, and 59.

[67] M. Otto. Bisimulation-Invariant Ptime and Higher-Dimensional mu-Calculus. *Theoretical Computer Science*, 224:237–265, 1999. Cited on page 32.

[68] Amir Pnueli. The temporal semantics of concurrent programs. In *Proceedings of the International Sympoisum on Semantics of Concurrent Computation*, pages 1–20, London, UK, 1979. Springer-Verlag. Cited on page 48.

[69] Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In D. Sangiorgi and J. Rutten, editors, *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, chapter 6, pages 233–289. Cambridge University Press, 2011. Cited on page 64.

[70] Thoralf Räsch. *Introduction to guarded logics*, pages 321–341. Springer-Verlag New York, Inc., New York, NY, USA, 2002. Cited on page 144.

[71] Andrew W. Roscoe. CSP and determinism in security modelling. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy*, SP '95, pages 114–127, Washington, DC, USA, 1995. IEEE Computer Society. Cited on page 118.

[72] John Rushby. Noninterference, transitivity and channel-control security policies. Technical Report CSL-92-02, SRI International, 1992. Cited on pages 4, 22, 23, and 60.

[73] Jan J. M. M. Rutten. Automata and Coinduction (An Exercise in Coalgebra). In Davide Sangiorgi and Robert de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 194–218. Springer, 1998. Cited on pages 8, 13, 18, 25, 26, and 27.

[74] Peter Y. A. Ryan and Steve A. Schneider. Process algebra and non-interference. *Journal of Computer Security*, 9(1/2):75–103, 2001. Cited on pages 4, 22, and 60.

[75] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, January 2003. Cited on pages 60 and 147.

[76] Davide Sangiorgi. *An introduction to bisimulation and coinduction*. Cambridge University Press, 2012. Cited on pages 15, 16, and 79.

[77] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 2011. Cited on pages 9 and 64.

[78] Sven Schewe. An optimal strategy improvement algorithm for solving parity and payoff games. In *Proceedings of the 22nd international workshop on Computer Science Logic*, CSL '08, pages 369–384, Berlin, Heidelberg, 2008. Springer-Verlag. Cited on pages 126, 143, and 144.

[79] Fred B. Schneider. Enforceable security policies. *ACM Transactions of Information Systems Security*, 3(1):30–50, 2000. Cited on pages 2, 18, and 150.

[80] Colin Stirling. Local model checking games. In *Proceedings of the 6th International Conference on Concurrency Theory, CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1995. Cited on pages 2, 110, 118, 119, and 144.

[81] Colin Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of the IGPL*, 7(1):103–124, 1999. Cited on pages 2, 102, 118, and 119.

[82] Colin Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001. Cited on pages 10, 53, 95, 96, 98, 100, 103, 110, 118, 119, 122, 173, 175, and 177.

[83] Colin Stirling and Perdita Stevens. Practical model-checking using games. In *TACAS 1998*, number 1384 in LNCS, pages 85–101, 1998. Cited on pages 11, 122, 140, 143, and 144.

[84] Tachio Terauchi and Alexander Aiken. Secure Information Flow as a Safety Problem. In Chris Hankin and Igor Siveroni, editors, *SAS*, volume 3672 of *Lecture Notes in Computer Science*, pages 352–367. Springer, 2005. Cited on pages 3, 58, and 150.

[85] Jens Vöge and Marcin Jurdziński. A Discrete Strategy Improvement Algorithm for Solving Parity Games. In *Proceedings of the 12th International Conference*

on Computer Aided Verification, CAV 2000, volume 1855 of *Lecture Notes in Computer Science*, pages 202–215. Springer-Verlag, 2000. Cited on pages 126, 143, and 144.

[86] A. Zakinthinos and E. S. Lee. A general theory of security properties. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, SP '97, pages 94–102, Washington, DC, USA, 1997. IEEE Computer Society. Cited on pages 23, 43, 69, 70, and 93.

[87] Steve Zdancewic and Andrew C. Myers. Observational determinism for concurrent program security. *Computer Security Foundations Workshop, IEEE*, 0:29, 2003. Cited on pages 20, 21, 42, and 68.

[88] Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998. Cited on pages 126, 143, and 144.

Curriculum Vitæ

Dimiter Vladimirov Milushev was born on May 14, 1981 in Sofia, Bulgaria. He graduated with honors from First English Language High School, Sofia in 1999. He received a *Magna cum Laude* double B.Sc. degree in Computer Science and Mathematics from Linfield College in 2003, as well as the *Outstanding Graduating Senior in Computer Science Award (in association with the Software Association of Oregon)*. In 2007 he received an M.Sc. in Information and Media Technology from TU Hamburg, as well as an MBA in Technology Management from the Northern Institute of Technology Management. Before joining KU Leuven as a PhD student, Dimiter worked at Sal. Oppenheim as a C++ developer.

Since October 2008 Dimiter has been a part of the DistriNet research group and worked on the verification of security-relevant hyperproperties under the supervision of Dave Clarke.

Dimiter attended the Marktoberdorf Summer School 2009 (Logics and Languages for Reliability and Security) and the Oregon Programming Languages Summer School 2010 (Logic, Languages, Compilation, and Verification). They were very influential and reassured Dimiter's interest in the application of formal methods for security.

During his PhD, Dimiter supervised 3 master thesis students and was a teaching assistant for the courses Fundamentals for Computer Science (Introduction to the Theory of Computation), Comparative Programming Languages and Object Oriented Programming.

List of Publications

International Conference Articles

- Dimiter Milushev and Dave Clarke. Towards incrementalization of holistic hyperproperties. In Pierpaolo Degano and Joshua D. Guttman, editors, proceedings of the *First International Conference on Principles of Security and Trust (POST 2012)*, volume 7215 of *Lecture Notes in Computer Science*, pages 329-348. 24 March – 1 April 2012. Tallinn, Estonia. Springer.
- Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties. In Audun Jøsang and Bengt Carlsson, editors, proceedings of the *17th Nordic Conference on Secure IT Systems (Nordsec 2012)*, volume 7617 of *Lecture Notes in Computer Science*, pages 121-136. 31 October – 2 November 2012. Karlskrona, Sweden. Springer.
- Dimiter Milushev, Wim Beck and Dave Clarke. Noninterference via symbolic execution. In Holger Giese and Grigore Rosu, editors, proceeding of the *IFIP International Conference on Formal Techniques for Distributed Systems joint international conference 14th Formal Methods for Open Object-Based Distributed Systems 32nd Formal Techniques for Networked and Distributed Systems (FMOODS & FORTE 2012)*, volume 7273 of *Lecture Notes in Computer Science*, pages 152-168. 13–16 June 2012. Stockholm, Sweden. Springer.
- Dimiter Milushev and Dave Clarke. Decidable incremental hyperproperty logics and model checking via games. Submitted for publication.
- Dimiter Milushev and Dave Clarke. Incremental hyperproperty model checking via games. Submitted for publication.

Technical Reports

- Dimiter Milushev and Dave Clarke. Towards incrementalization of holistic hyperproperties: extended version. *CW Reports, volume CW616*, 32 pages, Department of Computer Science, KU Leuven. December 2011. Leuven, Belgium.
- Dimiter Milushev and Dave Clarke. Coinductive unwinding of security-relevant hyperproperties: extended version. *CW Reports, volume CW623* 28 pages, Department of Computer Science, KU Leuven. August 2012. Leuven, Belgium.

Arenberg Doctoral School of Science, Engineering & Technology
Faculty of Engineering
Department of Computer Science
Scientific Computing Group
Celestijnenlaan 200A, box 2402
B-3001 Heverlee