TU WIEN Informatics

Automated Security Analysis of Blockchain Protocols

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

DI Sophie Rain, BSc
Registration Number 01425316

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Laura Kovács, MSc

The dissertation has been reviewed by:

Ruzica Piskac	Vincent Cheval

Vienna, May 13, 2025

Sophie Rain

Erklärung zur Verfassung der Arbeit

DI Sophie Rain, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.
Ich erkläre weiters, dass ich mich generativer KI-Tools lediglich als Hilfsmittel bedient habe und in der vorliegenden Arbeit mein gestalterischer Einfluss überwiegt. Im Anhang „Übersicht verwendeter Hilfsmittel“ habe ich alle generativen KI-Tools gelistet, die verwendet wurden, und angegeben, wo und wie sie verwendet wurden. Für Textpassagen, die ohne substantielle Änderungen übernommen wurden, haben ich jeweils die von mir formulierten Eingaben (Prompts) und die verwendete IT- Anwendung mit ihrem Produktnamen und Versionsnummer/Datum angegeben.

Wien, 13. Mai 2025

Sophie Rain

Acknowledgements

A doctorate is a challenging time! I was extremely lucky, and I am deeply thankful that I had such an amazing support structure in my colleagues, friends, and family. First of all, I want to express my heartfelt gratitude to my supervisor, Laura Kovács, without whom I never would have ever considered an academic career path. No matter how busy she was, she always made time to guide and encourage me throughout my journey. Thank you, Laura, for your unwavering support, insightful input, and invaluable feedback!

Further, I am deeply indebted to my colleague and dear friend Anja Petković Komel. She came into our research group at a time when I faced a difficult phase in my PhD programme. Anja was interested in my work and from there on joined me in my research efforts. We had countless highly fruitful but still very enjoyable brainstorming, coding, and writing sessions. She offered me emotional support whenever I needed it and helped me pressure myself less. I am so very thankful to her - this thesis would not have been possible without her!

I also want to thank the students whose master's theses I supervised, Lea Salome Brugger and Ivana Bocevska, for the very smooth collaborations. Both did outstanding work and integrated nicely in our "Checkmate Team", also on a personal level. I greatly appreciated the positive dynamic and connection we shared throughout the process. Moreover, I am grateful to my dear colleague and co-author, Michael Rawson. He always supported us with his extensive expertise in automated reasoning and coding, coming up with brilliant ideas when we least expected it.

Another important part of my life as a doctoral candidate is the office mates: I am deeply thankful to my office mates Anton, Márton, Jakob and Sarah for sharing so many funny moments, highly philosophical discussions, insights and very convenient quick fixes to problems that would have taken me hours to figure out on my own. These thanks extend to everyone in the Forsyte group for the great time we had during lunch and coffee breaks. Special thanks go to Beatrix, who constantly keeps all the bureaucratic stones out of our way; we really appreciate it!

Finally, I want to thank my family: My parents Christa and Wilfried, who paved the way for me to pursue the path I wanted; My sister Hannah, who I claim is the person who believes in me the most; and last but not least my husband Constantin, who endured all the lows with me and celebrated all the highs! Thank you for all your love and unwavering support!

This research was partially funded by the ERC CoG ARTIST 101002685, the ERC CoG BROWSEC 71527, and the ERC StG SYM-CAR 639270; the TU Wien Doctoral College SecInt; the Austrian Science Fund (FWF) projects PROFET P31621, LogiCS W1255-N23 and SpyCoDe 10.55776/F85; the Austrian Research Promotion Agency (FFG) (COMET K1 SBA, COMET K1 ABC); the WWTF ICT22-007 grant ForSmart; the Vienna Business Agency project Vienna Cybersecurity and Privacy Research Center (VISP); the Austrian Federal Ministry for Digital and Economic Affairs; the National Foundation for Research, Technology and Development; the Christian Doppler Research Association through CDL-BOT; the United States-Israel Binational Science Foundation (BSF) grant No. 2016260 and Grant No. 1810/18 from the Israeli Science Foundation, Len Blavatnik and the Blavatnik Family Foundation; the Amazon Research Awards FOREST (2020) and QuAt (2023); and the Netidee 2022 Fellowship 6123.

Abstract

Blockchain protocols are a popular target for attacks since they manipulate valuable funds while operating in a pseudo-anonymous environment, where "code is law". That means stealing funds through a loophole in a protocol is considered legal, and the stolen funds can be used unrestrainedly by the attacker. Such attacks can target different aspects of a blockchain protocol: the implementation, the cryptography, and the incentive structures. When it comes to ensuring the security of a blockchain protocol, all of them have to be considered. Most existing research on blockchain security focuses on implementation or cryptography aspects. However, blockchain protocols rely heavily on economic incentives to ensure correct execution, and it is hence essential to also study them. Such incentives can be captured using game-theoretic analysis of the behavior of blockchain users.

In this thesis, two aspects of blockchain security are addressed. Primarily, a comprehensive framework for game-theoretic security analysis is developed, combining extensive form games, symbolic payoffs, and automated reasoning techniques to rigorously model and evaluate a protocol's incentive structures. Additionally, this dissertation extends to the security of a protocol's implementation, proposing methods for formalizing and verifying unbounded summations in blockchain protocols. Thereby, this thesis enables both theoretical and practical advancements in blockchain security.

Regarding game-theoretic security, this thesis discusses how to ensure the incentives favor the intended behavior in every case. It provides a formal definition of game-theoretic security and advocates the use of extensive form games with symbolic utilities to model blockchain protocols. This dissertation further presents automation techniques for game-theoretic security analysis, making even the analysis of immense game models feasible. These techniques are implemented in the two versions of the automated game-theoretic security analysis tool CheckMate presented in this thesis. Our tool goes beyond conventional automation and enables piece-wise, so-called compositional automated analysis, allowing for greater scalability.

Moreover, approaches that target the security of the implementation have struggled to fully formally capture summation with an arbitrary, unbounded number of summands. In this work, a generalization to Presburger arithmetic is introduced that can express properties about summations. It is further shown how this generalization can be used to formalize verification problems of smart contracts, a specific type of blockchain protocols, thereby enhancing implementation security efforts.

Kurzfassung

Blockchain Protokolle sind ein beliebtes Ziel für Angriffe, weil sie wertvolle Güter verwalten und in einer pseudoanonymen Umgebung agieren, in der Code "Gesetz" ist. Das heißt Schwachstellen in Protokollen auszunützen, um Gelder zu stehlen, gilt als legal und die gestohlenen Gelder stehen der Angreifer*in uneingeschränkt zur Verfügung. Solche Angriffe können unterschiedliche Aspekte eines Blockchain Protokolls betreffen: die Implementierung, die Kryptographie und die Anreizstrukturen. Um die Sicherheit eines Blockchain Protokolls zu garantieren, müssen alle Aspekte betrachtet werden. Der Großteil der bestehenden Forschung konzentriert sich auf die Implementierung oder die Kryptographie. Da Blockchain Protokolle jedoch stark auf finanzielle Anreize angewiesen sind, um eine korrekte Ausführung sicherzustellen, ist es wesentlich auch diese zu untersuchen. Solche Anreize können mittels spieltheoretischer Analysen des Verhaltens der Blockchain Nutzer*innen erfasst werden.

In dieser Arbeit werden zwei Bereiche der Blockchain Sicherheit behandelt. Einerseits wird ein umfassender Ansatz zur spieltheoretischen Sicherheit entwickelt, der Spiele in Extensivform, abstrakte Gewinne und automatisches Schlussfolgern kombiniert, um die Anreizstrukturen von Protokollen rigoros zu modellieren und zu evaluieren. Andererseits erstreckt sich diese Dissertation auch zur Sicherheit der Implementierung, durch die Entwicklung von Methoden zum Formalisieren und Verifizieren von unbeschränkten Summationen. Dadurch werden sowohl theoretische als auch praktische Fortschritte in der Blockchain Sicherheit ermöglicht.

Hinsichtlich der spieltheoretischen Sicherheit diskutiert diese Arbeit wie sichergestellt werden kann, dass die Anreize in jedem Fall das gewünschte Verhalten fördern. Sie liefert eine formale Definition der spieltheoretischen Sicherheit und plädiert für die Verwendung von Spielen in Extensivform mit symbolischen Nutzenfunktionen zur Modellierung von Blockchain Protokollen. Darüber hinaus werden Automatisierungstechniken für die spieltheoretische Sicherheitsanalyse präsentiert, die es ermöglichen, selbst riesige Modelle zu analysieren. Diese Techniken wurden in den beiden Versionen unseres automatisierten Tools für spieltheoretische Sicherheitsanalysen, CheckMate, implementiert. Es geht über herkömmliche Automatisierung hinaus und ermöglicht eine stückweise, also zusammensatzbare, automatisierte Analyse, wodurch eine höhere Skalierbarkeit erreicht wird.

Weiters haben Ansätze zur Sicherheit der Implementierung bisher Schwierigkeiten, Summationen mit einer beliebigen Anzahl von Summanden vollständig formal zu erfassen. In dieser Arbeit wird eine Verallgemeinerung der Presburger Arithmetik eingeführt, die Eigenschaften von Summationen ausdrücken kann. Es wird zudem gezeigt, wie diese Verallgemeinerung genutzt werden kann, um Fragestellungen der Verifikation von Blockchain Protokollen zu formalisieren und so die Bemühungen zur Implementierungssicherheit zu verbessern.

Contents

CHAPTER 1

Overview

The age of digitalization has redefined the concept of security, giving it an entirely new meaning. It became essential for humanity that the digital systems they used were secure. This applies to everyday services such as emails or online banking, but also to digitally managed infrastructures such as traffic control and power grids, just to name a few. As a consequence, the research area of cybersecurity emerged to address the challenge. Frameworks were introduced to test digital systems, verify their implementation in software, and analyze their cryptographic premises – if any were applied, as they are in internet protocols, for example.

Then, in 2008, the first idea of blockchains was presented [Nak08] and led to the digitalization of yet another crucial part of modern society: means of payment, finance, and banking. A blockchain itself can be thought of as a chain of blocks, implemented by a linked list. Each item, called a block, contains information and is linked to the previous block by storing the previous block's hash value, thereby creating a chain. In the case of a cryptocurrency, such as Bitcoin [Nak08], the stored information in the blocks documents transactions. Besides tracking balances, most cryptocurrencies also support more complex constructs, such as, for example, a 2-of-3 Multisig [GBC23]: this is an account that is managed by three users and requires two out of three signatures to spend its funds. Such more involved concepts rely on protocols, which are executed by the users of a blockchain to reach a specific common goal, such as spending funds of a Multisig. There are also cryptocurrencies, such as the one on the Ethereum blockchain [But14], which enable finance products similar to the ones existing in traditional finance, like auctions or crowdfunding. These applications are called decentralized finance [Sch21], and their underlying protocols are called smart contracts [SEM18], which are pieces of source code that are stored on the blockchain and can be executed.

Many of those blockchain protocols and smart contracts rely on incentives and punishment mechanisms to ensure the users follow the instructions. The novelty for security considerations in this context is that the incentives are native to the technology. That

means the technology *enables* finance products of a currency but the incentives *are* of that currency *and* part of the technology. Hence, this is the first technology where incentives became an integral part of this very digital system. Therefore, to reason about the security of blockchain technology, it is essential to also study the nature of incentives and their impact on the behavior of blockchain users, which can be captured using game-theoretic analysis.

This dissertation addresses the challenge of the security aspects that are induced by incentive mechanisms in blockchain protocols. It introduces automated techniques to analyze and verify game-theoretic security.

1.1 Blockchain Protocols

Throughout this thesis, a *protocol* is meant to be a set of rules and guidelines to achieve a particular task. A blockchain protocol is thus a set of rules and guidelines to maintain or enable different aspects of blockchain technology. For simplicity, smart contracts, as introduced above and studied in Chapter 6, are also considered blockchain protocols. There are various different types of blockchain protocols. One of those, the so-called *off-chain protocols*, are particularly relevant to this dissertation, as they rely on interesting game-theoretic principles implemented through incentive and punishment mechanisms. An off-chain protocol enables secure transactions without having to publish each of them on the blockchain. Many different kinds of off-chain protocols exist [GMSR$^+$20], such as virtual channels [AME$^+$21], watchtowers [ALW20], payment-channels [DW15, AEE$^+$21, AKWZ21], state channels [DEF$^+$19, MBB$^+$19], virtual payment hubs [DEFM19], etc.

In the most widely adopted off-chain construction, Bitcoin's Lightning Network [PD16], parties deposit money in a shared address, called a channel, agree off-chain on new deposit distribution states, and publish the latest distribution state on the blockchain in the end. This concept can also be generalized to paths of such channels. By updating the distribution states one after another, money is routed from one endpoint of the path to the other, thereby achieving the same result as a regular on-chain transaction would. The Lightning Network relies on a punishment mechanism to disincentivize parties to publish outdated states on the blockchain and, in the case of channel paths, on an incentive mechanism, that motivates the intermediaries of the channel path to forward the transaction. In addition to the incentive structures, the Lightning Network also relies on cryptographic concepts and correct implementation, as every blockchain protocol does.

The nature of blockchain protocols makes them a popular target for attackers since they manipulate valuable funds while operating in a pseudo-anonymous environment in which everything that is technically possible is "allowed". As the blockchain community says: "Code is law". That means if a protocol has a loophole through which a malicious person can steal funds, the stolen funds officially belong to the attacker and they can use it unrestrainedly. Such attacks can target different aspects of a blockchain protocol:

The implementation, which is mostly critical for smart contracts as they support complex dependencies. There have been overflow attacks [How], where malicious users caused an integer variable to overflow on purpose. The overflow was applied such that the number of funds in the attacker's account increased drastically. Another attack type is re-entrancy. During such an attack, a specific line of code in a smart contract is entered multiple times, thereby subtracting a certain value over and over [AES25].

The used cryptographic concepts. Blockchain technology makes heavy use of cryptography. A common example is private keys (passcodes), which, among other applications, allow users to sign transactions. An example of a flaw in a cryptographic concept occurred in the Elliptic Curve Digital Signature Algorithm (ECDSA) used in Bitcoin. If a nonce, that means a random value that was used during a signing process, is reused or predictable, it can leak the private key due to the mathematical properties of ECDSA [Ami].

The incentive structures. It is possible that during protocol design incentives are introduced to motivate the participants to behave in a certain way. However, a developer can overlook how such mechanisms may award malicious actions in certain cases. An example of such a situation is the wormhole attack [MMSS$^+$19]. There, two users of the Lightning Network collude to "steal" funds that were a third party's incentive.

1.2 Game-Theoretic Security

When it comes to ensuring the security of a blockchain protocol, all three different aspects (implementation, cryptography, and incentives) have to be considered.

The following non-blockchain-related analogy aims to provide an intuition about the different aspects. When a password is used to log into some account, the security depends on 1) what the webpage does with the password that was typed in (implementation), 2) whether a hacker can steal the password (cryptography), and 3) whether the user wants to share their password (incentives). This is, of course, a drastic simplification, and depending on the application, the categories might overlap.

The security of the incentive structures relies on game-theoretic considerations and is therefore called *game-theoretic security*. This thesis mainly focuses on game-theoretic security. In particular, it studies how to make sure the incentives work as intended, which means that they favor the intended behavior in every case, no matter the circumstances. These concepts are thoroughly motivated and defined in Chapter 2. It further presents how to apply automated techniques to such a game-theoretic security analysis in Chapters 3–5. Chapter 6 is concerned with an aspect of implementation security: verifying correct summation with an unbounded number of summands.

To facilitate the reading of this dissertation, game theory, as well as relevant game-theoretic concepts, are briefly introduced.

Game Theory. Game theory is a field of mathematics that studies strategic interactions between decision-makers, called players, in situations where the outcome for each player

depends on the actions of all involved. It provides a framework to analyze and predict the behavior of individuals or groups in competitive, as well as cooperative settings. Game theory is widely used in various areas, such as economics, political science, and social sciences, to study decision-making, market competition, and evolutionary behavior, see, e.g., [PR20]. The core object of game theory is the *game*, a mathematical model of the studied social interaction.

Games. Similar to the variety of blockchain protocols is the variety of game types. However, any game essentially consists of three components: the players, the possible actions, and the resulting outcomes. They further share basic properties: The number of players is finite; at each step of the game it is known whose turn it is and what their possible actions are; and there is a pay-off for every player at the end of the game. Whether players cooperate, a game may take forever, or, if all the information is known by every player depends on the type of the game. This thesis considers finite games that do not contain any probabilistic actions. Moreover, all relevant information is assumed to be known by all players.

The most commonplace type of game is the *normal form game (NFG)*. In an NFG all the players choose one action simultaneously. Then, the game is over and the players receive their pay-off. The well-known game rock-paper-scissors is an example of an NFG. In this thesis, *extensive form games (EFGs)* are considered; this choice is motivated in detail in Chapter 2. EFGs are still rather simple games but allow multiple turns. The turns happen sequentially, in contrast to NFGs. The game tic-tac-toe is an EFG for instance. While not being able to express probabilistic behavior, EFGs allow for deriving deterministic security results, which is aimed for in this dissertation. Both NFGs and EFGs are *perfect information games*, which means all players know all the facts. In contrast to those, there are also *imperfect information games*, where the players only have partial knowledge of the facts. Most card games (e.g. Poker) are imperfect information games. Another important class of games is *stochastic games*. Those are games in which – in addition to the player's choices – chance plays a role. Roulette and dice games in general are stochastic games.

Players. Most social interactions can be modeled as games. Since the behavior of blockchain users is a social interaction, game theory applies as well. Every user is a player, who might aim to correctly use decentralized finance products, although they may have different objectives, such as (maliciously) maximizing their own profit. In this thesis, players with three different objectives are considered, as motivated in Chapter 2. The first type of player always follows the rules and guidelines of the protocol and thus aims to benefit from using the finance product as intended by its developers. Such players are called *honest*. The second type of player is the *rational* one; a rational player always behaves the way it yields the most financial profit for them, even if they have to violate the protocol's rules. The last type of player is called *Byzantine*. Byzantine players do not care about their pay-off or the rules and behave randomly; such players might have other goals, such as for example decreasing trust in the system.

As elaborated in this thesis (Chapter 2), to prove a blockchain protocol game-theoretically secure, one has to show that honest and rational behavior are identical. That means, for the incentive structures of a protocol to be considered secure, behaving honestly has to yield the best payoff. Additionally, a game-theoretically secure protocol should be resilient to Byzantine users. These security properties have been thoroughly studied and defined in terms of game-theoretic properties in Chapter 2.

1.3 Automated Reasoning

When analyzing different aspects of blockchain security, automation is generally desirable. Machines are much more reliable than humans when it comes to rather simple but repetitive tasks. They are also much faster. To enable automation even when it comes to verifying mathematical properties, that contain symbolic variables, as it does in this thesis, *automated reasoning* is required. By providing sound rules, mechanized methods have been developed with which it is possible to correctly and automatically reason about mathematical formulae [GV20, BT18, Mac95].

The game models that have to be studied to analyze the game-theoretic security of blockchain protocols are huge. Consider, for example, the routing of payments along a path of channels in the Lightning Network as introduced in Section 1.1: Modeling the routing along a path of three channels (hence four users) exceeded 200 GB of allocated disk space and resulted in a game with 144 342 306 nodes. Details on this are provided in Chapter 5. It is obvious that games of this size can hardly be analyzed manually. A manual verification of its game-theoretic security would not only take months – if not years – but is also very error-prone, cumbersome, and repetitive. It is just not feasible.

Straightforward automation of the analysis is not possible since the game-theoretic properties to be verified are of mathematical nature and contain symbolic variables. Hence, automated reasoning techniques have to be employed to successfully implement *automated game-theoretic security analyses*. While Chapters 3–4 address this challenge, Chapter 5 goes beyond conventional automation and enables piece-wise, so-called *compositional* automated analysis, overcoming the bottleneck of exceeded storage space.

1.4 Thesis Structure and Contributions

Each chapter of this thesis is based on a publication, which is stated at the beginning of the respective chapter. All of them have been peer-reviewed and published at a highly renowned computer science conference, except the one of Chapter 5, which is currently under review. The chapters are self-contained, including their own introductions, preliminaries, related work, and conclusion sections to allow for independent reading. The contributions of this dissertation are the following.

Chapter 2 – Introducing Game-Theoretic Security In this chapter, we advocate the use of game-theoretic concepts to reason about incentive structures in blockchain

protocols, and in particular, the use of extensive form games (EFGs) with *symbolic payoffs* to model blockchain protocols as games. The symbolic payoffs enable gap-free security analysis, as they represent all possible values of balances, transferred amounts, etc, through all-quantifying the variables in the symbolic payoffs. We further define game-theoretic security as the intersection of the three game-theoretic properties *weak immunity*, *collusion resilience*, and *practicality*. As a proof of concept, we also modeled two capabilities of Bitcoin's Lightning Network (the closing phase and the routing of payments) as EFGs and manually analyzed their game-theoretic security according to our definition with mathematical rigor. Our models are the first to accurately capture these aspects of the Lightning Network including arbitrary deviations. Thereby, we formalized precise criteria for when Lighting's closing and routing are secure and established first guidelines on the game-theoretic modeling of blockchain protocols.

Results of Chapter 2 have been published in [RAKM23] and presented at the IEEE Computer Security Foundations Symposium 2023.

Chapter 3 – Automated Game-Theoretic Security Analysis This part of the thesis presents a framework we developed that allows users to *automatically analyze* the game-theoretic security of a blockchain protocol based on its EFG model with symbolic payoffs. It is based on automated reasoning techniques such as *SMT solving*. To this end, we encoded game-theoretic concepts such as *strategies* and *histories* as first-order logic formulae. Also, the security properties defined in Chapter 2 were encoded in first-order logic. The encoding was proven sound and complete. We implemented the approach in a prototype and evaluated it on eight benchmarks, including the models introduced in Chapter 2. The security analysis terminated in a matter of seconds for all benchmarks, except for the routing of payments from Chapter 2, which is a tree with 21,688 nodes. Analyzing this benchmark took 1,222 seconds. For game models that violate a security property, we additionally present a method to automatically compute a weakest precondition: Adding this precondition as an assumption ensures the game model satisfies the security property. Similarly, we introduce counterexamples to the security properties and implement an algorithm to extract them as part of our framework. Note that the publication this chapter is based on, [BKK+23a], makes an implicit assumption about the structure of the game tree when reasoning about counterexamples to practicality. In this thesis, this assumption is lifted in Chapter 3 Definition 3.10 and Theorem 3.4. Lastly, for the models that satisfy the security properties, we provide a witness strategy as proof.

Partial results of Chapter 3 have been published at the ACM Conference on Computer and Communications Security 2023 in [BKK+23a].

Chapter 4 – The Game-Theoretic Security Tool CheckMate Based on the encoding presented in Chapter 3, we developed the game-theoretic security analysis tool CheckMate. In this chapter, we describe what the tool can do, how it operates, and how to use it. Moreover, constraints from the previous chapter are lifted (linear expressions),

concepts are refined (case splitting, counterexamples to practicality, weakest precondition generation), and the entire tool has a new implementation compared to its prototype in Chapter 3. We evaluated CheckMate on 15 benchmarks, out of which we introduced seven as new game-theoretic security benchmarks. The results show significant scaling improvements compared to its prototype.

The results of Chapter 4 have been published in [RBK+24] at the International Conference on Logic for Programming Artificial Intelligence and Reasoning 2024.

Chapter 5 – Compositional Game-Theoretic Security When it comes to game models based on actual blockchain protocols, CheckMate, as introduced in Chapter 4, still does not scale well enough, due to millions of game nodes. In fact, such game models are not even constructable as they exceed 200 GB of allocated disk space. We, therefore, in Chapter 5, propose a *compositional*, that is, a divide-and-conquer-like approach to game-theoretic security analysis. We establish theoretical results on the compositionality of the security properties. Thereby, we define variations of the properties that allow for compositional reasoning and prove them equivalent to their originals. Based on these new versions, we develop a divide-and-conquer algorithm to decide the game-theoretic security of a game model and prove it sound and complete. We implemented the algorithm as the next generation of CheckMate and evaluated it on the benchmark set of Chapter 4, which shows significant improvements in runtime. Most importantly, our methodology supports *subtree-supertree reasoning* due to its compositional structure. That means a part of the game model – a so-called subtree – can be analyzed independently ahead of time. This functionality allows for iteratively generating and analyzing parts of a game model, which drastically decreases the required disk space as the already analyzed part of the model does not have to be stored. Further, we also present methods to conveniently extract witness strategies, weakest preconditions, and counterexamples without causing major overhead.

Chapter 6 – Further Reasoning about Implementation Security This chapter goes beyond game-theoretic security and is concerned with an aspect of *implementation security*: It studies *summation* with an arbitrary unbounded number of summands. Smart contracts can manipulate the total number of assets, called tokens. This can happen through the minting or burning of such tokens. Other smart contracts might want to change the balances of some users' tokens through a transaction while ensuring the total number of assets remains unchanged (keyword: integer overflow). Implementation security approaches have struggled to fully formally capture such unbounded summations. Chapter 6 makes the following contributions towards this challenge: We introduce a generalization to Presburger arithmetic that can express properties about summations and show how verification problems of smart contracts can be formalized using it. We study the decidability of our extension and present different encodings of it to first-order logic, also considering theory-specific reasoning. Furthermore, we evaluate the encodings using SMT solvers and first-order theorem provers using 31 new benchmarks, implementing summation-related transitions of smart contracts and their properties. The experiments

showcase the applicability of our results towards implementation security of blockchain protocols.

Results of Chapter 6 have been published at the International Conference on Computer Aided Verification 2021 in [ERI+21a].

Finally, **Chapter 7 – Summary and Outlook** concludes this thesis. It summarizes the main results, gives an outlook on possible future work, and aims to inspire continuation based on our findings.

1.5 Impact

Included Peer-Reviewed Publications. The following peer-reviewed publications are included in this thesis. The author of this thesis is either the main contributor or one of the main contributors to the mentioned publications. The work of [ERI+21a], has one other main author, whose main contribution was the (un-)decidability analysis in Section 6.4, which is why the proofs of this section are omitted in this thesis.

[RAKM23] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In Proceedings of IEEE 36th Computer Security Foundations Symposium, pages 31–46, Los Alamitos, CA, USA, 2023.

[BKK+23a] Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. In Proceedings of 30th ACM SIGSAC Conference on Computer and Communications Security, pages 1407–1421, New York, NY, USA, 2023.

[RBK+24] Sophie Rain, Lea Salome Brugger, Anja Petković Komel, Laura Kovács, and Michael Rawson. Scaling Checkmate for Game-Theoretic Security. In Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, pages 222–231, Stockport, UK, 2024.

[ERI+21a] Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up Smart Transitions. In Proceedings of 33rd International Conference on Computer Aided Verification, pages 317–340, Cham, Switzerland, 2021.

Extended Versions and Preprints. In addition to the publications listed in the previous paragraph, this thesis also incorporates results from the following extended versions, respectively, a preprint of work that is currently under review.

[RAKM21] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. arXiv preprint, 2021.

[BKK+23b] Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. EasyChair Preprint no. 10853, 2023.

[BKK+25] Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. Divide and Conquer: A Compositional Approach to Game-Theoretic Security. EasyChair Preprint no. 15785, 2025.

[ERI+21b] Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up Smart Transitions. arXiv preprint, 2021.

Other Publications. Working towards this thesis, also inspired other work. Meant to foster diversity in computer science, the author of this thesis also co-founded the workshop series "Abenteuer Informatik für Volksschule" which led to the following conference paper.

[LRKF23] Martina Landman, Sophie Rain, Laura Kovács, and Gerald Futschek. Reshaping Unplugged Computer Science Workshops for Primary School Education. In Proceedings of 16th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, pages 139–151, Lausanne, Switzerland, 2023.

Awards and Honors. The research within this thesis brought the following additional recognitions to the thesis author.

- In October 2022 the Christina Hörbiger Prize was awarded in acknowledgment of the efforts and endeavors made towards this thesis.
- In November 2022 a Netidee stipend was granted from the Internet Stiftung to support the research that led to this thesis.
- In July 2023, the author of this thesis was listed among TU Wien's 30 under 30, an institution to highlight young talents at TU Wien.
- In May 2024, the Best Presentation Award was given to the author of this thesis for the presentation of [RBK+24] at the 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning.

CHAPTER 2

Introducing Game-Theoretic Security

This chapter is based on article [RAKM21]:

Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. arXiv preprint, 2021.

This article is an extended version of the publication [RAKM23]:

Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In Proceedings of IEEE 36th Computer Security Foundations Symposium, pages 31–46, Los Alamitos, CA, USA, 2023.

2.1 Problem Statement

Blockchain technologies are emerging as a revolutionary paradigm to perform secure decentralized financial applications. Nevertheless, a widespread adoption of cryptocurrencies, such as Bitcoin [Nak08] and Ethereum [Woo14], is severely hindered by their inherent limitations on transaction throughput [HSHS20, CMVSM18]. For instance, while Bitcoin can support tens of transactions per second and the confirmation time is about an hour, traditional credit networks like Visa can comfortably handle up to 47,000 transactions per second.

Off-chain protocols [GMSR+20] are recognized as one of the most promising scalability solutions, achieving a seemingly contradictory property: the bulk of transactions is performed off-chain, and yet in a secure fashion. The idea is to leverage the blockchain only in case of disputes, resorting otherwise to off-chain, peer-to-peer transactions. Bitcoin's Lightning Network [PD16] is the most widely adopted off-chain instantiation, hosting at

the time of writing bitcoins worth more than 170M USD, in a total of more than 27,000 nodes and more than 76,000 channels. In a nutshell, parties deposit money in a shared address, called a channel, and can later on perform arbitrarily many off-chain transactions with each other by redistributing the deposit on the channel. In the end, the channel can be closed and the latest state (i.e., deposit distribution) is posted on-chain. Off-chain transactions are not limited to the end-point of the channel, but they can be routed over paths of channels (so-called multi-hop payments). Besides such payment channel networks, an entire ecosystem of off-chain protocols [GMSR+20] (virtual channels, watchtowers, payment-channel hubs, state channels, side-chains, etc.) is under development for Bitcoin [MMSS+19, AEE+21, AME+21, AMKM21, AKWZ21, DW15], Ethereum [DEFM19, DEF+19, MBB+19, ALW20], as well as other cryptocurrencies [TMSS20].

The cryptographic protocols underlying these off-chain constructions are rather sophisticated and, most importantly, rely on game-theoretic arguments to discourage malicious behavior. For instance, the Lightning Network relies on a punishment mechanism to disincentivize parties to publish old states on-chain and on an unlocking mechanism where parties first pay a neighbor and then retrieve the paid amount from the other to ensure the atomicity of multi-hop payments (i.e., either all channels are consistently updated or none is).

Off-chain protocols are typically subject to rigorous security analyses, which, however, concentrate on cryptographic properties and do not capture the game-theoretic ones. In particular, most protocols are proven secure in the Universal Composability framework [CDPW07], proving that the cryptographic realization simulates the ideal functionality. This framework, however, was developed to reason about security in the classical honest/Byzantine setting: in particular, the ideal functionality has to model all possible parties' behavior, rational and irrational, otherwise it would not be simulatable, but reasoning on whether or not certain behavior is rational is outside of the model and thus left to informal arguments. This is not just a theoretical issue, but a practical one, as there is the risk of letting attacks pass undetected: for instance, the Wormhole attack [MMSS+19] constitutes a rational behavior in the Lightning Network, which is thus admitted in any faithful model thereof, although it undermines its incentive mechanism. The first step towards closing this gap in cryptographic proofs is to come up with a *faithful game-theoretic model for off-chain protocols* in order to reason about security in the presence of rational parties. We address this challenge in this work, advocating the use of Extensive Form Games (EFGs) for the game-theoretic security analysis of off-chain protocols. In particular, we introduce two instances of EFGs to model the closing and the routing of the Lightning Network.

2.1.1 Related Work

A game-theoretic model for off-chain protocols is initiated and introduced in [ZBPBS21]. This work suffers, however, from several limitations, which make it unsuitable to conduct faithful security analyses. Firstly, the game model considers only honest closing of channels, i.e., all deviations – such as posting an old state – are ignored: this makes

it impossible to reason about the security of basic channel operations. Secondly, the pay-offs are represented as constants, which neglects the dependency of the channel's balance on its security properties. Further, fees are not considered at all, thereby ignoring their impact on Lightning protocols. For instance, the routing game to model the security of multi-hop payments fails to capture already identified attacks in payment channel networks, like the Wormhole attack [MMSS+19] that targets the fee distribution among players. Additionally, Lightning is vulnerable to the Griefing attack [KSHB19], where a significant amount of money is locked. In our work, we overcome the aforementioned limitations by defining a stronger closing phase model, by aligning the utilities to the monetary outcome, by considering all possible deviations of parties during closing, and by revising the relevant security properties. We demonstrate the importance of precision in game-theoretic protocol models by modeling the Wormhole attack, as well as the Griefing attack.

Our work further complements other game-theoretic advancements in the area, most prominently the following lines of research.

Incentivizing Watchtowers A major drawback of payment channel protocols is that channel participants must frequently be online and watch the blockchain to prevent cheating. To alleviate this issue, the parties can employ third parties, or so-called watchtowers, to act on their behalf in case their counterparty misbehaves. Correctly aligning the incentives of watchtowers to yield a secure payment channel protocol is, however, challenging. This is the main focus of several works [MBB+19, ALS+18, ALW20, AKWZ21]. As their objective is to incentivize external parties, their models do not apply in our work.

Payment Channel Network Creation Games Avarikioti et al. [AHWW20, ASW19] study payment channel networks as network creation games. Their goal is to determine which channels a rational node should establish to maximize its profit. Ersoy et al. [ERE20] undertake a similar task; they formulate the same problem as an optimization problem, show it is NP-hard, and present a greedy algorithm to approximate it. Similarly to our work, all these works assume rational participants. However, we aim to model the security of the protocols, in contrast to these works that study the network creation problem graph-theoretically.

Blockchains with Rational Players Blockchains incentivize miners to participate in the network via monetary rewards [Nak08]. Therefore, analyzing blockchains under the lens of rational participants is critical for the security of the consensus layer. There are multiple works in this direction: Badertscher et al. [BGM+18] present a rational analysis of the Bitcoin protocol. Eyal and Garay [ES14] introduce an attack on the Nakamoto consensus, effectively demonstrating that rational miners will not faithfully follow the Bitcoin protocol. This attack is generalized in [KKS+17, SSZ16]. Consequently, Kiayias et al. [KKKT16] analyze how miners can deviate from the protocol to optimize their expected outcome. Later, Chen et al. [CPR19] investigate the reward allocation schemes

in longest-chain protocols and identify Bitcoin's allocation rule as the only one that satisfies a specific set of desired properties. On a different note, several works study the dynamics of mining pools from a game-theoretic perspective [Eya15, TJS16] or introduce network attacks that may increase the profit of rational miners [HKZG15, NKMS16]. An overview of game-theoretic works on blockchain protocols can be found in [LNW+19].

All these works, however, focus on the consensus layer (Layer-1) of blockchains and as both the goals and assumptions are different from the application layer (Layer-2), the models introduced there cannot be employed for our purposes. For instance, payment channel protocols occur off-chain and thus game-based cryptographic assumptions of the blockchain do not apply. In addition, consensus protocols investigate the expected reward of miners ,which is a probabilistic problem, whereas we ask if any honest player could lose money, which depends on the behavior of the other players and is fundamentally deterministic.

Game-based definitions have also been proposed for the security analysis of smart contracts [CGV18, CGP19]. These models, however, target an on-chain setting and are thus not suitable for reasoning about the specifics of off-chain constructions (e.g., closing games, routing games, etc.).

2.1.2 Our Contributions

In this work, we take the first steps towards closing the gap between security and game-theoretic analysis of off-chain protocols. Specifically, we introduce the first game-theoretic models that are expressive enough to reason about the security of off-chain protocols. We model off-chain protocols as games and then analyze whether or not certain security properties are satisfied. The design of our models is driven by two principles: (a) all possible actions should be represented, and (b) the utility function should mirror the monetary outcome realistically. We aim to ensure that *honest participants do not suffer any damage (P1)*, whereas *deviating from the protocol yields a worse outcome for the adversary (P2)* We will use weak immunity (Definition 2.4) to implement (P1), and collusion resilience (Definition 2.16) together with practicality (Definition 2.15) for (P2). While we believe that our approach of implementing principles (a) and (b) is easily extensible to other off-chain protocols, in this work, we focus on the Bitcoin Lightning Network, which constitutes the most widely adopted off-chain protocol. Our technical contributions can be summarized as follows:

- We refine existing game-theoretical concepts in order to reason about the security of off-chain protocols (Section 2.3).
- We introduce the Closing Game G_c, the first game-theoretic security model that accurately captures the closing phase of Lightning channels, encapsulating arbitrary deviations from the protocol specification (Section 2.4).
- We perform a detailed security analysis of G_c, formalize folklore security corner cases of Lightning, and present the strategy that rational parties should follow to

close their channels in order to maximize their expected outcome relative to the current and previous distribution states (Section 2.5).

- We identify limitations in prior work [ZBPBS21] on game-based modeling of multi-hop payments, putting forward a new game-based definition that is precise enough to cover the Wormhole and the Griefing attack (Section 2.6). We further show how to model Fulgor protocol [MMSK+17], a variant of Lightning's routing that prevents the Wormhole attack. Our formalization leverages game theory concepts introduced in Section 2.3 and Section 2.4, thereby demonstrating the theoretical expressiveness of our framework to analyze complex protocols.

In conclusion, our work brings game-theoretical foundations to enforce security of off-chain protocols, by providing a rigorous analysis over security properties expressed through formal requirements over game strategies. We believe the provided rigor in our work opens up new venues for automating security analysis via game-theoretic arguments, a challenge which we aim to tackle in future work.

2.2 Background and Preliminaries

2.2.1 Payment Channel Networks

A payment channel [AEE+21] can be seen as an escrow (or multi-signature), into which two parties Alice A and Bob B transfer their initial coins with the guarantee that their coins are not locked forever and the agreed balance can be withdrawn at any time. After that, A and B can pay each other off-chain by signing and exchanging messages that reflect the updated balances in the escrow. These signatures can be used at any time to close the channel and distribute the coins on-chain according to the last channel state. In order to discourage parties from posting an old state on-chain, a punishment mechanism is in place. In particular, in Lightning [PD16], once A closes the channel, she has to wait a mutually agreed time before getting her coins. Meanwhile, B has the opportunity to withdraw all the coins in the channel (by posting a so-called revocation transaction), including the ones assigned to A, if the state posted on-chain by A is not the last one they mutually agreed on. Such a punishment mechanism is of game-theoretic nature: parties can indeed post an old state on-chain, yet they are discouraged to do so.

In particular, Lightning payment channels operate as follows: First, Alice and Bob create a funding transaction where they input their respective coins; the funding transaction has a single output that can only be spent if both A and B provide their signature (2-out-of-2 multi-signature). Then, the two parties create the first commitment transaction, i.e., a transaction that spends the output of the funding transaction and returns the initial coins to both parties. In other words, the input of the commitment transaction is the output of the funding transaction while the output of the first commitment transaction is two-fold: the first output returns the coins to A and the second output to B. However, the commitment transaction each party holds is not the same. Specifically,

the commitment transaction of A has an additional spending condition, a timelock t that signifies the revocation period and is pre-agreed between the two parties; in A's commitment transaction, B's output is spendable immediately. Symmetrically, in B's commitment transaction B's output has a timelock t while A's output is spendable immediately. Note that a timelock t is a condition that allows the coins of the output to be spent on-chain only after time t has elapsed from the publication of the transaction. After A and B sign and exchange the respective first commitment transactions, they proceed to sign the funding transaction and publish it on-chain. This order is important to avoid hostage situations[1]. As soon as the funding transaction is securely published on-chain, A and B can transact off-chain by creating every time a new commitment transaction that depicts the current balance of the joint capital among the two parties. Every time a new commitment transaction is created, the parties reveal a secret to their counterparty that allows their counterparty to spend their own coins immediately (e.g., A can spend B's coins from the previous commitment) if the previous commitment transaction appears on chain (revocation transaction). To close a Lightning channel, the two parties can either collaborate and spend the output of the funding transaction, or each of them can close the channel unilaterally by publishing the last commitment transaction. Since the commitment transactions each party hold have a timelock, in case of cheating, i.e., publication of a previous commitment transaction on-chain, the counterpart can immediately spend the cheating party's coins, claiming all the coins of the channel, thus punish the cheating party for misbehaving.

Technically, A and B do not just lock their initial funds but also a certain small amount which will be used as a fee for the closing transaction of the channel. Note that every on-chain transaction requires such a fee f. The fee for the opening transaction is paid upon the opening of the channel and is thus irrelevant to our consideration. However, in case A posts an old state on-chain and B performs the revocation transaction – which is an on-chain transaction – to prove it, B has to carry the additional transaction fee alone. These facts have an important impact on our game-theoretic models.

In the following, we refer to *honest closing* when a party unilaterally closes the channel by posting the last commitment transaction or when the parties close collaboratively, where both parties sign to spend the funding transaction output directly.

Off-chain transactions are not limited to the end-points of a channel, as they can be performed whenever sender and receiver are connected by a path of channels with enough capacity. The cryptographic approach to do so exploits hash-time-locked contracts (HTLC) [DW15]. Assume players A and B do not share a channel. Instead, A has a channel with E_1; E_1 has a channel with I; I has a channel with E_2; and E_2 has a channel with B, as illustrated in Figure 2.1. Player A can now send an amount m to player B via the intermediaries E_1, I, and E_2, where each intermediary charges a fee f for the

[1] If the funding transaction is published on-chain before the first commitment transactions are signed, a party may hold the other hostage since none of the parties can close the channel unilaterally but only in collaboration.

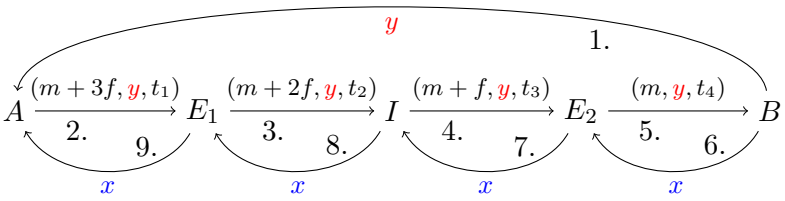


Figure 2.1: Routing in Lightning using HTLCs.

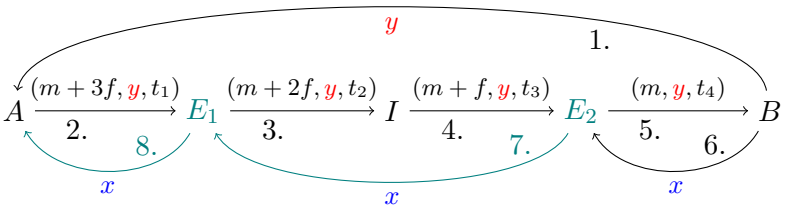


Figure 2.2: Wormhole Attack in Lightning.

routing service, hence A should pay $m + 3f$. The core idea is that A pays E_1, E_1 pays I, and so forth until B gets paid.

A key security property in multi-hop payments is *atomicity*: either all payments are successful, and the deposit in each channel is updated accordingly, or none is. To achieve this property, the Lighting protocol proceeds as follows. First, the receiver B generates a secret x and sends its hash $h(x) = y$ to the sender A (see action 1 in Figure 2.1). Then A creates an HTLC for E_1, where she locks $m + 3f$ with lock y and time-out t_1. That means only E_1 can claim the money and only by providing a value whose hash is y within time t_1 (action 2 in Figure 2.1). Although E_1 does not know such a value yet and can therefore not unlock, E_1 can nevertheless proceed by creating another HTLC for I also locked with y and a time-out t_2 (action 3 in Figure 2.1). Thereafter, I and E_2 continue in the same way (actions 4 – 5 in Figure 2.1). Actions 1 – 5 of Figure 2.1 are called the *locking phase*. Note that in order to allow everybody to unlock their HTLCs in the subsequent steps, the time-outs have to be decreasing $t_1 > t_2 > t_3 > t_4$. Once B receives the conditional payment, he can reveal x to E_2 and the conditional payment is unlocked (action 6 in Figure 2.1). The others can now unlock the HTLCs one after the other from right to left (actions 7 – 9 in Figure 2.1), which is called the *unlocking phase*. Finally, A paid $m + 3f$, B received m and each intermediary was rewarded with f.

We note that atomicity is achieved by a game-theoretic argument: intermediaries can, in principle, stop the protocol either in the locking phase or in the unlocking phase. In the former, they would lose the transaction fee f, while in the latter, they would lose the payment amount m, $m + f$, $m + 2f$ respectively. Thus, they are incentivized to act once they have committed to participate.

The Wormhole Attack The aforementioned routing protocol is proven to be vulnerable to the *Wormhole attack* [MMSS+19], which is depicted in Figure 2.2. The attack is as follows: E_1 and E_2 collude, and bypass I in the unlocking phase, thus stealing I's participation reward f. Until actions 6 in Figure 2.1 and Figure 2.2, the behavior is identical. Then, E_2, knowing x, forwards x to E_1 (offline) instead of unlocking the HTLC from I (action 7 in Figure 2.2). This way, E_1 can unlock A's HTLC and claim the money (action 8), but I will never be able to unlock. After a certain time the remaining HTLCs time-out and the locked money returns to the creators.

Therefore, the parties A and B are not affected. However, E_1 and E_2 collectively earn $3f$ instead of the $2f$ they deserve, stealing the fee f from I, who locked resources in the locking phase of the protocol. This attack undermines the incentive of intermediaries to route payments.

The Griefing Attack It describes the scenario when a player, assume B for simplicity, ignores the proposed payment and refuses to proceed [KSHB19]. This way, money is locked in the conditional payments for a considerable amount of time. While [MBS+22] studies the Griefing attack through probabilistic modelling and [BMR20] provides mitigation techniques, to the best of our knowledge there is no formal security analysis of this attack at present. Our work addresses this limitation and shows that Lightning's routing module is indeed susceptible to the Griefing attack.

In the sequel we consider the behavior as illustrated in Figure 2.1 as the only *honest routing* behavior.

2.2.2 Game-Theoretic Definitions

We now introduce the game-theoretic concepts relevant for our formalization. We denote real numbers by $\mathbb{R}$ and tuples as $\sigma = (\sigma_1, ..., \sigma_n)$. We write $\sigma[\sigma_i'/\sigma_i]$ to denote the tuple resulting from substituting σ_i by σ_i' in σ, that is $\sigma[\sigma_i'/\sigma_i] = (\sigma_1, ..., \sigma_{i-1}, \sigma_i', \sigma_{i+1}, ..., \sigma_n)$. We understand games as static objects in which finitely many players can choose finitely many times from a finite set of actions. A game yields a certain positive or negative utility for each player. We briefly overview the very common Normal Form Games, also called Strategic Games [OR94], in which each player chooses an action *only once*, called strategy.

Definition 2.1 (Normal Form Game – NFG)**.** *A* Normal Form Game (NFG) *is a tuple* $\Gamma = (N, \mathscr{S}, u)$*, where* N *is the set of game* players*,* $\mathscr{S} = \bigtimes_{p \in N} \mathscr{S}_p$ *the set of* joint strategies σ *and* u *the* utility function*:*

- $\mathscr{S}_p$ *is the non-empty set of strategies player* p *can choose from. Thus, a joint strategy* $\sigma \in \mathscr{S}$ *is a tuple of strategies* $\sigma = (\sigma_{p_1}, ..., \sigma_{p_{|N|}})$*, with* $\sigma_{p_i} \in \mathscr{S}_{p_i}$*.*
- $u = (u_{p_1}, \ldots, u_{p_n})$*, where* $u_{p_i} : \mathscr{S} \to \mathbb{R}$ *assigns player* p_i *its utility for every joint strategy* $\sigma \in \mathscr{S}$*.*

Table 2.1: NFG Γ_C with players A, B.

A \ B	U	C	$\mathfrak{I}$
U	$(1/2, 1/2)$	$(0, 1)$	$(0, 1)$
C	$(1, 0)$	$(1, 1)$	$(-1, -1)$
$\mathfrak{I}$	$(1, 0)$	$(-1, -1)$	$(-1, -1)$

In what follows we fix an arbitrary game Γ and give all definitions relative to it. To formalize an optimal outcome on game strategies, we use Nash Equilibria.

Definition 2.2 (Nash Equilibrium – NE)**.** *A* Nash Equilibrium *is a joint strategy $\sigma \in \mathscr{S}$ s.t. no player p_i can increase their utility by unilaterally deviating from $\sigma = (\sigma_{p_1}, ..., \sigma_{p_{|N|}})$. Formally,*

$$\forall p \in N \; \forall \sigma'_p \in \mathscr{S}_p : \; u_p(\sigma) \geq u_p(\, \sigma[\sigma'_p/\sigma_p] \,) \; . \tag{2.1}$$

Another important concept is *weakly dominated strategies*, expressing the strategies a rational player would not play since they yield worse utilities.

Definition 2.3 (Weakly Dominated Strategy)**.** *A strategy $\sigma^d_p \in \mathscr{S}_p$ of player p is called* weakly dominated *by strategy $\sigma'_p \in \mathscr{S}_p$, if it always yields a utility at most as good as σ'_p and a strictly worse utility at least once:*

$$\forall\, \sigma \in \mathscr{S} : \; u_p(\, \sigma[\sigma^d_p/\sigma_p] \,) \leq u_p(\, \sigma[\sigma'_p/\sigma_p] \,) \text{ and} \tag{2.2}$$

$$\exists\, \sigma \in \mathscr{S} : \; u_p(\, \sigma[\sigma^d_p/\sigma_p] \,) < u_p(\, \sigma[\sigma'_p/\sigma_p] \,) \; . \tag{2.3}$$

Example 2.1. Consider the NFG Γ_C in Table 2.1, which was introduced in [ZBPBS21] to model closing. In this game Γ_C, there are two players $N = \{A, B\}$ and each players can choose from the same strategy set $\mathscr{S}_A = \mathscr{S}_B = \{U, C, \mathfrak{I}\}$. Here, strategy U captures unilateral closing, that is publishing the latest state on-chain. Further, strategy C corresponds for closing collaboratively, that is publishing a mutually signed transaction. Finally, strategy $\mathfrak{I}$ stands for ignoring, that is doing nothing. The utility for each joint strategy is given in Table 2.1, where player A's strategies are listed in the left column of Table 2.1 and the strategies of B are given in the top row of Table 2.1.

Applying Definition 2.2, the joint strategies (C, C), $(U, \mathfrak{I})$ and $(\mathfrak{I}, U)$ are Nash Equilibria: for each of these joint strategies, neither A nor B can deviate in order to increase their own utility. Comparing the second and the third row of Table 2.1, we see that A's utility is always as least as good in the second row as it is in the third row. Hence, strategy C weakly dominates strategy $\mathfrak{I}$ for player A, by Definition 2.3; the same property also holds for player B. By comparing the other pairs of rows/columns of Table 2.1, we see that there is no other weak dominance in Γ_C.

2.2.3 Game-Theoretic Security Properties of Off-Chain Protocols

We now present existing game-theoretic concepts [ZBPBS21, OR94] implying security properties of off-chain protocols. In Section 2.3, we extend these concepts towards another type of games, called Extensive Form Games, enabling our security analysis in Section 2.4. We focus on two security properties ensuring that (P1) honest players do not suffer damage, and (P2) subgroups of rational players do not deviate from a respective strategy. A protocol is compliant to these properties, if the strategy implementing the intended behavior satisfies them; we call such a strategy an *honest strategy*.

(P1) *No Honest Loss.* As the utility function of a game is supposed to display the monetary and intrinsic value of a certain joint strategy, property (P1) is expressed using *weak immune strategies* defined next.

Definition 2.4 (Weak Immunity)**.** *A joint strategy $\sigma \in \mathscr{S}$ in an NFG Γ is called* weak immune*, if every player p that follows σ gets utility at least 0, regardless of how the other players behave:*

$$\forall p \in N \;\; \forall \sigma' \in \mathscr{S} : \quad u_p(\, \sigma'[\sigma_p/\sigma'_p] \,) \geq 0 \,. \tag{2.4}$$

Example 2.2. In the game Γ_C of Table 2.1, the only weakly immune strategy is (U, U). This is the case, because as long as A chooses U, player B can take any strategy and A will never get negative utility (similarly, vice-versa).

(P2) *No Deviation.* Even though the concept of Nash Equilibria seems to be a good candidate to ensure (P2) at first glance, they have two crucial shortcomings. First, a Nash Equilibrium only ensures that a single player cannot profit from deviating, but it does not imply that two or more players cannot do so. Second, there might be Nash Equilibria, which are weakly dominated by another strategy for a specific player. Such Nash Equilibria will therefore not be played by rational parties and hence should not be considered to satisfy (P2).

The solution proposed for NFGs in [ZBPBS21] is to consider strategies σ compliant to (P2), if they are both *strongly resilient* (fixing the former shortcoming) and *practical* (fixing the latter) as defined subsequently.

Strong resilience extends Nash Equilibria by considering deviations of multiple players.

Definition 2.5 (Strong Resilience – SR)**.** *A joint strategy $\sigma \in \mathscr{S}$ in an NFG Γ is* strongly resilient (SR) *if no proper subgroup of players $S := \{s_1, ..., s_j\}$ has an incentive in deviating:*

$$\begin{aligned} \forall S \subset N \quad & \forall \sigma'_{s_i} \in \mathscr{S}_{s_i} \quad \forall p \in S : \\ & u_p(\sigma) \geq u_p(\, \sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}] \,) \,. \end{aligned} \tag{2.5}$$

Table 2.2: NFG Γ'_C obtained from IDWDS over Table 2.1.

A \ B	U	C
U	$(1/2, 1/2)$	$(0, 1)$
C	$(1, 0)$	$(1, 1)$

We note that in games with two players (i.e. two-player games), strong resilience and Nash Equilibrium are identical. As such, in Γ_C from Table 2.1, the joint strategies (C, C), $(U, \mathfrak{I})$ and $(\mathfrak{I}, U)$ of Example 2.1 are also strongly resilient.

To define practicality of a strategy, we first introduce the concept of *iterated deletion of weakly dominated strategies (IDWDS).*

Definition 2.6 (Iterated Deletion of Weakly Dominated Strategies – IDWDS)**.** *The* iterated deletion of weakly dominated strategies *(IDWDS) of an NFG* Γ *is defined as iteratively rewriting* Γ *by omitting* all *weakly dominated strategies of all players. This is repeated until no strategy is weakly dominated anymore. The resulting game* Γ' *is thus a subgame of* Γ*.*

Note that when IDWDS is applied to a game Γ, then every Nash Equilibrium of the resulting game Γ' is also a Nash Equilibrium of Γ. Since all weakly dominated strategies of every player are removed at each step, the generated game is unique. Details and proofs can be found in [OR94].

We now define practical strategies, in order to ensure that no single strategy is weakly dominated at any iteration.

Definition 2.7 (Practicality)**.** *A strategy is* practical *if it is a Nash Equilibrium of the NFG* Γ' *after iterated deletion of weakly dominated strategies.*

Example 2.3. Let us consider Γ_C from Table 2.1. We know from Example 2.1 that only $\mathfrak{I}$ is weakly dominated for both A and B. Therefore, according to Definition 2.6, strategy $\mathfrak{I}$ has to be removed from both players' strategy set. This yields the game Γ'_C as listed Table 2.2.

Note that there are no weakly dominated strategies in Γ'_C . Thus, any Nash Equilibrium of Γ'_C is also practical strategy of Γ_C. By comparing utilities, we derive that the only Nash Equilibrium of Γ'_C is the joint strategy (C, C).

An alternative approach for expressing (P2) is by requiring a strategy σ to be both a *strong Nash Equilibrium* (a property similar to SR) and practical, instead of SR and practical.

Definition 2.8 (Strong Nash Equilibrium – sNE)**.** *A joint strategy* σ *is a* strong Nash Equilibrium (sNE) *if for every group of deviating players* $S := \{s_1, ..., s_j\}$ *and all possible*

deviations $\sigma'_{s_i} \in \mathscr{S}_{s_i}$, $i \in \{1, ..., j\}$ *at least one player* $p \in S$ *has no incentive to participate, that is*

$$\begin{aligned} \forall S \subseteq N,\, S \neq \emptyset \quad & \forall \sigma'_{s_i} \in \mathscr{S}_{s_i} \quad \exists p \in S : \\ & u_p(\sigma) \geq u_p(\, \sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}]). \end{aligned} \tag{2.6}$$

Example 2.4. In Γ_C from Table 2.1, all NE are also sNE. For the joint strategy (C, C), this is easy to see. However, it is also the case for $(U, \mathfrak{I})$ and $(\mathfrak{I}, U)$, since any deviation yields a utility of at most 1. Thus, at least one player's utility does not increase by deviating from $(U, \mathfrak{I})$, $(\mathfrak{I}, U)$ respectively.

A detailed comparison of the various concepts ensuring (P2), including their strengths and weaknesses, is given in Section 2.3.

2.3 EFG-based Modeling of Off-Chain Protocols

So far we considered games in which each party takes only one action. We now extend our definitions to handle adaptive strategies, i.e., games in which parties take several actions and choose at each step which action to take based on the actions previously chosen by other parties. As we will see, this is necessary for faithfully modeling off-chain protocols and overcoming the limitations of previous work [ZBPBS21]. For that, we overview the concept of extensive form games (EFGs) in Section 2.3.1. We show how to lift NFG-based security definitions to EFGs in Section 2.3.2. Finally, we show that these definitions do not yet suffice to yield an accurate security model of off-chain protocols, and introduce a refined security definition based on the concept of collusion resilience in Section 2.3.3.

2.3.1 Extensive Form Games (EFG)

To formalize strategies where players make multiple choices one after the other, we advocate the usage of Extensive Form Games (EFGs) [OR94], which extend NFGs as follows.

Definition 2.9 (Extensive Form Game – EFG)**.** *An* Extensive Form Game (EFG) *is a tuple* $\Gamma = (N, \mathscr{H}, P, u)$*, where* N *and* u *are as in NFGs. The set* $\mathscr{H}$ *captures* game histories*,* $\mathscr{T} \subseteq \mathscr{H}$ *is the set of* terminal histories*, and* P *denotes the* next player function*, satisfying the following properties.*

The set $\mathscr{H}$ *of histories is a set of sequences of actions with*

1. $\emptyset \in \mathscr{H}$*;*
2. *if the action sequence* $(a_k)_{k=1}^{K} \in \mathscr{H}$ *and* $L < K$*, then also* $(a_k)_{k=1}^{L} \in \mathscr{H}$*;*
3. *a history is terminal* $(a_k)_{k=1}^{K} \in \mathscr{T}$*, if there is no action* a_{K+1} *with* $(a_k)_{k=1}^{K+1} \in \mathscr{H}$*.*

The next player function P

1. *assigns the next player $p \in N$ to every non-terminal history $(a_k)_{k=1}^K \in \mathscr{H} \setminus \mathscr{T}$, that is $P((a_k)_{k=1}^K) = p$;*
2. *after a non-terminal history $h = (a_k)_{k=1}^K \in \mathscr{H}$, it is player $P(h)$'s turn to choose an action from the action set $A(h) = \{a : (h, a) \in \mathscr{H}\}$.*

A strategy *of player p is a function σ_p mapping every $h \in \mathscr{H}$ with $P(h) = p$ to an action from $A(h)$. Formally,*

$$\sigma_p : \{h \in \mathscr{H} : P(h) = p\} \to \{a : (h, a) \in \mathscr{H}, \forall h \in \mathscr{H}\} ,$$

such that $\sigma_p(h) \in A(h)$. The set of all strategies of a player p is $\mathscr{S}_p$, and the set of all joint strategies is $\mathscr{S} = \bigtimes_{p \in N} \mathscr{S}_p$.

Note that the set of terminal histories $\mathscr{T}$ is uniquely determined by $\mathscr{H}$ and therefore does not explicitly occur in the tuple Γ. Since histories h are just sequences of actions $h = (a_k)_{k=1}^K = (a_1, ..., a_K)$, we denote histories by the variable h, the abstract sequence $(a_k)_{k=1}^K$, or the explicit sequence $(a_1, ..., a_K)$, depending on the context in which they are used. We note that EFGs can conveniently be represented as trees, as described below.

Definition 2.10 (EFG as Tree)**.** *Considering an EFG $\Gamma = (N, \mathscr{H}, P, u)$, the following tree $G = (V, E)$ represents Γ.*

- *For every history $h \in \mathscr{H}$, there exists exactly one node $v_h \in V$. This is labeled by $P(h)$, the next player, if h is not terminal ($h \notin \mathscr{T}$), or by $u(\sigma)$, the joint utility of playing a game with history h, if h is terminal ($h \in \mathscr{T}$) and the joint strategy σ yields history h.*
- *Two nodes v_h, $v_{h'} \in \mathscr{H}$ are connected via an oriented edge $(v_h, v_{h'}) \in E$ iff $h' = (h, a)$. This edge is labeled a.*

Let us illustrate EFGs and their tree-based representation through the following example.

Example 2.5. The game tree in Figure 2.3 results from the extensive form game $\Gamma_E = (N, \mathscr{H}, P, u)$ with the two players $N = \{A, B\}$, where the set of histories is $\mathscr{H} = \{\emptyset, (a), (b), (b, c), (b, d), (b, d, e), (b, d, f), (b, d, f, g), (b, d, f, i)\}$. The next player function P assigns player A after histories $\emptyset$ and (b, d), and player B after (b) and (b, d, f). Finally, the utility function u assigns joint utility $(2, 2)$ to strategies that yield history (a), utility $(3, 1)$ for strategies with history (b, c), utility $(1, 1)$ for strategies with history (b, d, e), and $(0, 2)$ for strategies resulting in (b, d, f, i). A strategy $\sigma = (\sigma_A, \sigma_B)$ in Γ_E is for example: A chooses a after history $\emptyset$: $\sigma_A(\emptyset) = a$; and f after (b, d): $\sigma_A((b, d)) = f$; B takes c after (b): $\sigma_B((b)) = c$, and g after (b, d, f): $\sigma_B((b, d, f)) = g$. Following this strategy until we read a leaf yields history (a). A different strategy $\sigma' = (\sigma'_A, \sigma'_B)$, which also yields history (a), is for example $\sigma'_A(\emptyset) = a$, $\sigma'_A((b, d)) = e$ and $\sigma'_B = \sigma_B$.

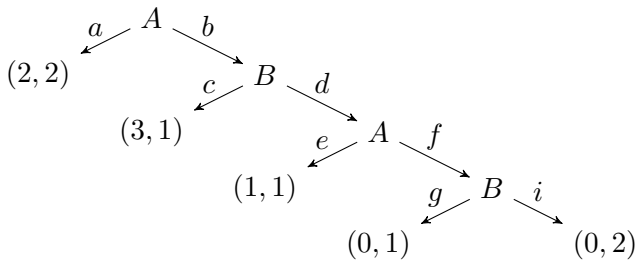


Figure 2.3: An EFG Γ_E.

As depicted in the tree-based representation of Figure 2.3, we note that the utility of joint strategies in an EFG is uniquely determined by their associated history (i.e., path). In the context of EFGs, the concept of *Nash Equilibria* remains as given in Definition 2.2. In addition to Nash Equilibria, another useful concept for EFGs is the *Subgame Perfect Equilibrium*, which we will use to characterize the strategies played in practice by rational parties. To this end, we first introduce the notion of subgames of EFGs. A subgame of an EFGs can be seen as a subtree determined by a certain history (i.e., whose root node is the last history node), and is formalized below.

Definition 2.11 (Subgame of EFG)**.** *The* subgame *of an EFG* $\Gamma = (N, \mathscr{H}, P, u)$ *associated to history* $h \in \mathscr{H}$ *is the EFG* $\Gamma(h) = (N, \mathscr{H}_{|h}, P_{|h}, u_{|h})$ *defined as follows:* $\mathscr{H}_{|h} := \{h' \mid (h, h') \in \mathscr{H}\}$, $P_{|h}(h') := P(h, h')$, *and* $u_{|h}(h') := u(h, h')$.

Example 2.6. Consider the EFG Γ_E from Figure 2.3 . The subgame of Γ_E associated with history (b, d) is the subtree rooted in A.

By adjusting the concept of Nash Equilibrium to subgames, we derive the following property of joint strategies.

Definition 2.12 (Subgame Perfect Equilibrium)**.** *A* subgame perfect equilibrium *is a joint strategy* $\sigma = (\sigma_1, ..., \sigma_n) \in \mathscr{S}$, *s.t.* $\sigma_{|h} = (\sigma_{1|h}, ..., \sigma_{n|h})$ *is a Nash Equilibrium of the subgame* $\Gamma(h)$, *for every* $h \in \mathscr{H}$. *The strategies* $\sigma_{i|h}$ *are functions that map every* $h' \in \mathscr{H}_{|h}$ *with* $P_{|h}(h') = i$ *to an action from* $A_{|h}(h')$.

2.3.2 EFG Extensions for Security Properties

While EFGs enable us to incorporate choices made at different times, yielding different options for the next player, they come with the following limitation. The intended (i.e., honest) behaviors in off-chain protocols only specify a terminal history (i.e., a path from root to leaf), rather than a strategy. For instance, an honest history may specify to close the channel collaboratively, but it does not capture a player's behavior once a player deviated. To address this limitation, we introduce the following notion of an extended strategy in EFGs.

Definition 2.13 (Extended Strategy)**.** *Let β be a terminal history in an EFG Γ. Then, all strategies σ_β that result in history β are* extended strategies *of β.*

Example 2.7. Recall Figure 2.3. In Example 2.5, we consider the terminal history (a) and provide two extended strategies of (a), they are σ and σ'. A strategy, which is not an extended strategy of (a) is for instance $\sigma'' = (\sigma''_A, \sigma''_B)$, where $\sigma''_A(\emptyset) = b$, $\sigma''_A((b,d)) = e$ and $\sigma''_B = \sigma_B$. This is the case because by following the choices of A and B in σ'', we end up in (b,c).

While EFGs can in principle be translated to NFGs, as explained in [OR94], analyzing the security properties (P1)-(P2) over the translated NFGs may yield unexpected results. We shortly exemplify this point in Example 2.8, but similar issues also occur in larger games. We thus lift NFG-based definitions to EFGs, enabling the analysis of (P1) and (P2). Since EFGs have a utility function just as NFGs do, which assigns values after the game, the NFG concepts of weak immunity, strong resilience and sNE remain the same for EFGs.

Definition 2.14 (EFG Properties)**.** *A joint strategy $\sigma \in \mathscr{H}$ of an EFG Γ is called weak immune, strongly resilient, or a strong Nash Equilibrium, if it satisfies the formulae of Definition 2.4, Definition 2.5 or Definition 2.8 respectively.*

Practicality in NFGs, however, relies on IDWDS, which fails to incorporate the sequential nature of EFGs, and hence must be adjusted for EFGs. This is because NFG actions happen simultaneously, while EFG players choose their actions sequentially. We first present an example to showcase that applying the NFG definition of practicality to an EFG, by using its translation to an NFG, leads to overlooking rational strategies.

Example 2.8. Let us consider the EFG Γ_E from Figure 2.3, with two players A and B. The compact translation of Γ_E to an NFG Γ_N is given in Table 2.3. Histories of Figure 2.3, where players choose twice, such as (b,d,f), are translated to Table 2.3 as the joint strategy $(b; f\ , d)$. Hence, the NFG strategy $b; f$ of player A means choosing action b first, and, if A gets to choose again, A takes f. Player A's strategies are displayed in the rows, whereas player B's are shown in the columns of Table 2.3. Strategy $d; g$, for example, denotes choosing d in the first turn and g in the second turn, unless the game ends before. For readability, strategies with identical utilities in any case are merged together, e.g., having only a instead of both $a; e$ and $a; f$.

According to definition of practicality for NFGs (see Definition 2.7), the only practical strategy in Γ_N is $(a,\ d; i)$, which results in a utility of $(2,2)$. This is because for A strategy $b; e$ weakly dominates $b; f$ and for B strategy $d; i$ weakly dominates both c and $d; g$. After deleting those (in blue), the red strategy $b; e$ of A becomes weakly dominated by a. Thus, after removing $b; e$ only the joint strategy $(a,\ d; i)$ remains and is therefore a Nash Equilibrium of the resulting game.

However, in the EFG Γ_E the comparison of strategies has a certain order, as not all choices are made simultaneously. Thus, when it comes to B choosing between option c

Table 2.3: Compact View of Γ_E, Translated to an NFG Γ_N.

A \ B	c	$d\,;g$	$d\,;i$
a	$(2,2)$	$(2,2)$	$(2,2)$
$b\,;e$	$(3,1)$	$(1,1)$	$(1,1)$
$b\,;f$	$(3,1)$	$(0,1)$	$(0,2)$

and d, choosing c is also a rational action because in any case B gets utility 1. This is the case, since the subgame after d, will end in the subgame perfect and practical $(1,1)$, if played by rational players. Following this argumentation, we claim that $(b;e,c)$, yielding history (b,c) should also be considered rational and thus practical.

Example 2.8 demonstrates that it is advisable to adapt the NFG concept of practicality for EFGs, and that a näive application can be problematic since information may be lost during the transformation from EFG to NFG [OR94]. We therefore propose to use subgame perfect equilibria for comparing EFG strategies, and define *practicality for EFGs* as follows.

Definition 2.15 (Practicality for EFG)**.** *A strategy of an EFG* Γ *is* practical *if it is a subgame perfect equilibrium of* Γ*.*

2.3.3 Security Strategies for Off-Chain Protocols

We now leverage the previously introduced EFG-based definitions (Section 2.3.2) to faithfully model the security of off-chain protocols. In particular, we propose the novel concept of *collusion resilience* for addressing (P2), and compare it to existing formalizations of property (P2).

In [ZBPBS21], strong resilience and practicality were used to model the no deviation property of (P2): We identify unwanted properties of strong resilience and we thus investigate variations of it. Specifically, we show that strong Nash Equilibria do not imply strong resilience nor vice-versa (Lemma 2.1), and therefore define the *collusion resilience* property of a joint strategy. Intuitively, collusion resilience considers the sum of the utilities of the deviating parties, since rational players may collude or be controlled by the same entity.

Definition 2.16 (Collusion Resilience – CR)**.** *A joint strategy* $\sigma \in \mathscr{S}$ *in an EFG/NFG* Γ *is called* collusion resilient (CR) *if no strict subgroup of players* $S := \{s_1, ..., s_j\}$ *has a joint incentive in deviating from* σ*. That is,*

$$\forall S \subset N \quad \forall \sigma'_{s_i} \in \mathscr{S}_{s_i} : \\ \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\ \sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}]\ ). \tag{2.7}$$

Table 2.4: Overview of Implications and Counterexamples.

$\rightarrow$	SR	$\mathsf{SR}_{\subseteq}$	sNE	CR
SR		Γ_3	Γ_3	✓
$\mathsf{SR}_{\subseteq}$	✓		✓	✓
sNE	Γ_1	Γ_1		Γ_1
CR	Γ_2	Γ_2	Γ_3	

In addition, we also consider a slight adaptation of strong resilience, $\mathsf{SR}_{\subseteq}$, where the deviation of the entire set of players N is also allowed, as it is for sNE.

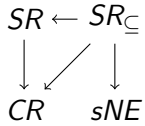
Definition 2.17 (Strong Subset Resilience – $\mathsf{SR}_{\subseteq}$)**.** *A joint strategy $\sigma \in \mathscr{S}$ is called* strongly subset resilient ($\mathsf{SR}_{\subseteq}$)*, if no player of any subgroup $S \subseteq N$, $S := \{s_1, ..., s_j\}$ has an incentive to deviate from σ:*

$$\begin{aligned} \forall S \subseteq N \quad & \forall \sigma'_{s_i} \in \mathscr{S}_{s_i} \quad \forall p \in S : \\ & u_p(\sigma) \geq u_p(\, \sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}] \,) \; . \end{aligned} \tag{2.8}$$

We now formalize how the resilience properties relate to each other, which motivates our definition of (P2).

Lemma 2.1 (Resilience Properties)**.** *Let $\sigma \in \mathscr{S}$ be a joint strategy. The following and only the following implications hold.*

1. *σ is $\mathsf{SR}_{\subseteq}$ $\Rightarrow$ σ is SR, CR, sNE.*
2. *σ is SR $\Rightarrow$ σ is CR.*



Proof. We start by showing property (2). Let σ be SR and let $S = \{s_1, ..., s_j\} \subset N$, $\sigma'_S \in \mathscr{S}_S$ be arbitrary but fixed. Then, for all $p \in S$ we have $u_p(\sigma) \geq u_p(\sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}])$ and thus also $\sum_{p\in S} u_p(\sigma) \geq \sum_{p\in S} u_p(\sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}])$. Hence σ is CR and the implication is proven. For implication (1) we see that $\mathsf{SR}_{\subseteq} \Rightarrow$ SR is trivial. If the property is satisfied for every $S \subseteq N$, then it is also satisfied for every $S \subset N$. By (2) and the transitivity of implication we also get $\mathsf{SR}_{\subseteq} \Rightarrow$ CR. For the last implication let σ be $\mathsf{SR}_{\subseteq}$ and let $S = \{s_1, ..., s_j\} \subseteq N$, $S \neq \emptyset$ and $\sigma'_S \in \mathscr{S}_S$ be arbitrary but fixed. Then there exists some $p \in S$ and by definition all $p \in S$ satisfy $u_p(\sigma) \geq u_p(\sigma[\sigma'_{s_1}/\sigma_{s_1}, ..., \sigma'_{s_j}/\sigma_{s_j}])$. Therefore, σ is sNE.

To prove that no other implication holds between those four concepts, we provide three counterexamples. An overview of which game disproves which implication is given in Table 2.4.

The three-player NFG Γ_1 in Table IV shows a joint strategy (H_1, H_2, H_3) that is sNE, but not CR (refer to Example III.5). Using the just proven (1) and (2), we get that (H_1, H_2, H_3) is also not SR nor $\mathsf{SR}_{\subseteq}$.

The three-player game Γ_2 (Table V) shows a strategy (H_1, H_2, H_3) that is CR, but not SR (see Example III.5) and thus also not $\mathsf{SR}_{\subseteq}$ (property (1)). It is, however, sNE: The only relevant deviation from (H_1, H_2, H_3) is (D_1, H_2, D_3), as it yields a different utility $(3, 0, -2)$ instead of $(1, 1, 1)$. While player P_1 profits in this case, player P_3 does not. One deviating player not profiting suffices for a strong Nash Equilibrium, thus (H_1, H_2, H_3) is sNE.

To prove the remaining implications incorrect, we consider the two-player game Γ_3 in Table 2.5. We can easily see that (H_1, H_2) is not $\mathsf{SR}_{\subseteq}$, nor sNE. This is the case, as all players $\{P_1, P_2\}$ can deviate to play (D_1, D_2), which yields a strict increase for both. However, since no player profits from deviating alone, (H_1, H_2) is still SR and CR. □

The next example further motivates why we decided to formalize (P2) in terms of *collusion resilience*.

Example 2.9. Consider the games Γ_1 and Γ_2, respectively defined in Tables 2.6-2.7. The games Γ_1 and Γ_2 show that there exist cases where both strong resilience and strong Nash Equilibria fail to correctly state whether rational players will deviate, while collusion resilience does not.

Let us study Γ_1 first. There are three players P_1 on the left, P_2 in the "3rd dimension" who only has one possible strategy, and P_3 at the top. Let us consider the joint strategy $\sigma = (H_1, H_2, H_3)$. Since P_2 does not have another choice, P_2 can never deviate. Player P_1 deviating alone yields the same utility as σ and is thus irrelevant. The same holds for P_3. The only deviation that makes a difference, is if P_1 and P_3 change strategy together to (D_1, H_2, D_3). By doing so, P_1 profits and receives 5 instead of 1, but P_3 loses by getting -2 instead of 1. Thus, P_3 does not have an incentive to do so, unless the two players collude for their mutual benefit and share their payoffs. This way they receive 1.5 each instead of 1 each, which poses a serious threat to σ and should thus not be considered satisfying (P2). However, (H_1, H_2, H_3) is sNE, since P_3 has no incentive in

Table 2.5: Game Γ_3.

	H_2	D_2
H_1	$(1, 1)$	$(1, 1)$
D_1	$(1, 1)$	$(2, 2)$

Table 2.6: Three Player Game Γ_1.

H_2	H_3	D_3
H_1	$(1,1,1)$	$(1,1,1)$
D_1	$(1,1,1)$	$(5,0,-2)$

Table 2.7: Three Player Game Γ_2.

H_2	H_3	D_3
H_1	$(1,1,1)$	$(1,1,1)$
D_1	$(1,1,1)$	$(3,0,-2)$

deviating with P_1, if their utilities are not shared, but it is not CR, since

$$2 = u_{P_1}(H_1, H_2, H_3) + u_{P_3}(H_1, H_2, H_3) \tag{2.9}$$
$$< u_{P_1}(D_1, H_2, D_3) + u_{P_3}(D_1, H_2, D_3) = 3\ . \tag{2.10}$$

In the similar game Γ_2, on the contrary, P_3 has no incentive in deviating from $\sigma = (H_1, H_2, H_3)$ together with P_1, also if their utilities are shared. Such a deviation yields 0.5 each, instead of 1 each in σ. Hence, there is no incentive to change strategy for one or more players and therefore (H_1, H_2, H_3) should be considered satisfying (P2). Nevertheless, according to Definition 2.5, (H_1, H_2, H_3) is not SR, since at least one of the deviating parties P_1, P_3 profits from choosing (D_1, H_2, D_3), although P_3 has no reason to play along. However, in Γ_2, (H_1, H_2, H_3) is CR as

$$2 = u_{P_1}(H_1, H_2, H_3) + u_{P_3}(H_1, H_2, H_3) \tag{2.11}$$
$$\geq u_{P_1}(D_1, H_2, D_3) + u_{P_3}(D_1, H_2, D_3) = 1\ . \tag{2.12}$$

Remark 1 (Formalizing ((P1) and (P2))**.** Based on the resilience properties of Lemma 2.1, we say *(P2) is satisfied by a joint strategy* σ, if σ is CR and practical. In addition, a joint strategy σ *satisfies (P1), if* σ *is weak immune*, as in [ZBPBS21].

We conclude this section by defining secure game strategies/histories, as follows.

Definition 2.18 (Secure Strategy)**.** *A strategy* σ *of an NFG/EFG is* secure *if it is weak immune, practical and CR.*

When discussing security in the setting of EFGs, we are interested mainly in assessing whether a *history* is secure, as the protocol only defines an honest history instead of a full strategy. By applying Definition 2.13, we state the following security characterization.

Definition 2.19 (Secure History)**.** *A terminal history* β *of an EFG is* secure *if there exist extended strategies* σ_1*,* σ_2*, and* σ_3 *of* β*, such that* σ_1 *is weak immune,* σ_2 *is practical and* σ_3 *is CR.*

We note that we do not have to find a secure extended strategy for the history to be secure, as aiming for one joint secure strategy in an EFG would be unnecessarily restrictive. Instead, our goal is to make sure that rational parties follow the honest history, no matter

what their actual strategy is. In particular, an honest player follows the honest history by default, a rational player does so because of practicality and collusion resistance. Weak immunity further ensures that honest players as well as rational one cannot be damaged by Byzantine players while following the honest history. Hence, the strategy each player has in mind does not matter, since in a secure protocol weak immune, practical, and collusion resilient, strategies are overlapping along the honest history. This is the case because in Definition 2.19 we require σ_1, σ_2, and σ_3 to all yield the same history, namely β. We can therefore admit that an honest player has a weak immune strategy in mind, while a rational player has a practical one, as long as these overlap on the honest history.

2.4 Closing Games of Off-Chain Protocols

We now define a new two-player EFG, called the *Closing Game* G_c, in order to model closing phase properties of off-chain protocols, in particular of the Lightning Network. As explained in Section 2.2.1, to close a channel, a party can unilaterally publish a channel state on-chain, which does not necessarily have to be the latest one. The one who closes, however, has to wait a certain amount of time until the money can be used. Meanwhile, the other party can steal all the money from the channel in case the state published on-chain is not the latest one: this ensures that rational players close their channel only with the latest state. Alternatively, the parties can collaboratively sign a new transaction to split the money. In this case, no one has to wait.

Our closing game overcomes the limitations of previous work [ZBPBS21] in representing dishonest closing attempts, by modeling how closing can be achieved after a failed collaborative closing attempt and by also considering the additional fee f to be paid in a revocation transaction.

To the best of our knowledge, our closing game G_c is the most accurate model for the security analysis of off-chain protocols, notably of the Lightning Network. In our model of the closing phase we make the following assumptions for a channel between A and B at the moment where the closing phase is initiated.

- The fair split of the channel's funds is $a \to A$, $b \to B$ and $a > 0$, $b > 0$.
- The benefit of closing the channel is α. Closing a channel yields a benefit, since it unlocks assets.
- The opportunity cost of having to wait for one's funds upon closing is ϵ.
- When both players agree to update the channel we assume a fair deal in the background which yields a profit of ρ for both parties.
- Publishing a revocation transaction on-chain costs a fee $f > 0$.

Further, to properly model utilities in the closing game G_c, we define the following total order, which is crucial for analyzing security properties of G_c. For capturing total order

properties in the setting of G_c, we extend the set $\mathbb{R}$ of real numbers by the infinitesimal numbers α, ϵ and ρ.

Definition 2.20 (Utility Order)**.** *We consider the total order $(\mathbb{U}, \preccurlyeq)$, where $\mathbb{U}$ is the group resulting from closing $\mathbb{R} \dot{\cup} \{\alpha, \epsilon, \rho\}$ under addition. The total ordering $\preccurlyeq$ is uniquely defined by the following conditions.*

1. *On $\mathbb{R}$, the relation $\preccurlyeq$ is the usual less than or equal relation $\preccurlyeq|_{\mathbb{R}} := \leq$.*
2. *The values α, ϵ and ρ are greater than 0,*

$$\forall \xi \in \{\alpha, \epsilon, \rho\} : \ -\xi \prec 0 \prec \xi \ . \tag{2.13}$$

3. *The values α, ϵ and ρ are closer to 0 than any real number,*

$$\forall x \in \mathbb{R}, \xi \in \{\alpha, \epsilon, \rho\}, x > 0 : \ \xi \prec x, \ -x \prec -\xi. \tag{2.14}$$

4. *Additionally, α, ϵ and ρ have the order $\rho \prec \epsilon \prec \alpha$.*

In general, unlocking funds gives additional financial freedom even if there is some processing delay; therefore, we choose $\epsilon \prec \alpha$ in Definition 2.20. Additionally, once the parties initiate the closing phase, it is reasonable to assume that no potential update significantly benefits both parties. In contrast, both parties are interested in avoiding the opportunity cost, i.e., the cost of having to wait for their funds upon closing, therefore, we set $\rho \prec \epsilon$ in Definition 2.20.

Remark 2. While the ordering conditions of Definition 2.20 may seem to be restrictive, lifting them comes with the burden of considering a high number of possible variable orderings. In particular, one would need to consider (number of variables)! orderings, which would highly complicate the formal analysis task. Approximating or clustering the number of orderings, while weakening conditions in Definition 2.20, is an interesting venue for future work.

Based on the utility ordering of Definition 2.20, we introduce our *Closing Game for Player A* below.

Definition 2.21 (Closing Game $G_c(A)$ of Player A)**.** *The* Closing Game $G_c(A) = (N, \mathscr{H}, P, u)$ *is an EFG with two players $N = \{A, B\}$. The tree representation of $G_c(A)$ in Figure 2.4 defines $\mathscr{H}$, P and u[2], with the actions of the game being summarized in Table 2.8.*

[2]The subgames S_i, S_i' are given in Figures 2.5 and 2.7.

Table 2.8: Possible Actions in $G_c(A)$.

H	Close unilaterally and *honestly* without reacting to a previous move, such as a collaborative closing attempt.
D	Close unilaterally but *dishonestly* (without reacting to a previous move) with a profit of $d_A \in (0, b]$ in A's case, $d_B \in (0, a]$ in B's case.
C_h	Try to close *collaboratively* and *honestly*, that is proposing a fair split.
C_c	Try to close *collaboratively* but by *cheating* the other party by $c \in (0, b]$, that means proposing an unfair split.
S	*Signing* the collaborative closing attempt of the other player.
$\mathfrak{I}$	*Ignore* the previous action and do nothing.
P	*Prove* other party tried to close dishonestly. That means stating a revocation transaction. We assume its publication requires a fee of $f > 0$ and that the attempt to do so is always successful, that is, that the miners behave honestly.
U^+	Propose an *update* of the channel where player A's balance is *increased* by $p_A \in (0, b]$.
U^-	Propose an *update* where player A's balance is *decreased* by $p_B \in (0, a]$.
$\mathfrak{A}$	Agree to a proposed update.

The Closing Game The game $G_c(A)$ in Figure 2.4 assumes a current channel state (a, b), that means value a for player A and value b for player B. It starts with a closing action of player A: There are two ways of closing an off-chain channel, unilaterally or collaboratively, which can each be done honestly (using the channel state (a, b) as a basis) or dishonestly (e.g., using an outdated state). If A closes unilaterally and honestly (action H), we reach a leaf; unilateral but dishonest closing (action D), gives player B the option to prove A was cheating (action P), or to just ignore A's dishonesty (action $\mathfrak{I}$); collaborative closing is more involved. Independent of whether A proposed an honest split (a, b) (action C_h) or a dishonest split $(a + c, b - c)$ (action C_c) player B can choose from the following actions: B can agree to the proposed split by signing it (action S), can ignore A's closing attempt (action $\mathfrak{I}$), can unilaterally close themselves, both honestly (action H) and dishonestly (action D), which gives again A the chance to prove it (action P) or ignore it (action $\mathfrak{I}$). Further player B can even propose a channel update, where player A's balance increases (action U^+), or where it decreases (action U^-), which leads to subgame $S_{3,4}$ or $S'_{3,4}$ respectively. If player B chose action $\mathfrak{I}$, it is again player A's turn to close unilaterally (actions H, D), propose a channel update (actions U^+, U^-) or to not do anything (action $\mathfrak{I}$) which causes the funds to stay locked in the channel.

The *utility function* u of $G_c(A)$ in Figure 2.4 assigns player $p \in N$ the money player p received minus the money player p deserved based on the latest channel state. The values of closing (α), updating (ρ) and waiting ($-\epsilon$) are also considered in Figure 2.4. As discussed in Section 2.2.1, the fee needed for the closing transaction is assumed to have been reserved among the locked funds in the channel all the time and is spent upon closing, therefore not affecting the players' channel balance.

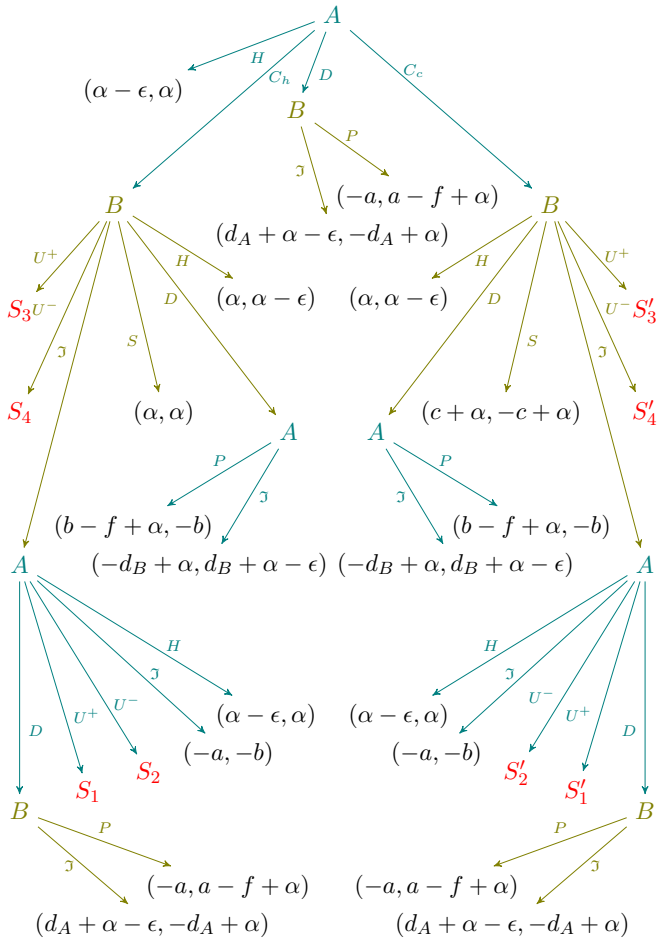


Figure 2.4: Closing Game $G_c(A)$, with channel state at the root of the game of $a > 0$ for A and $b > 0$ for B.

Subgames of the Closing Game The subgames $S_{1,2}$ and $S'_{1,2}$ in Figure 2.5 cover the case where a channel update is proposed by A, although A has already signed a collaborative closing attempt. In S_1 the closing attempt was honest, hence $y = 0$ (in Figure 2.5) and the update is from channel state (a, b) to $(a + p_A, b - p_A)$, hence $x = p_A$ (in Figure 2.5). In S_2, it is also $y = 0$ and the proposed channel update is $(a - p_B, b + p_B)$, thus $x = -p_B$. In $S'_{1,2}$, the closing attempt was dishonest, therefore $y = c$. The channel updates are as before, thus $x = p_A$ for S'_1 and $x = -p_B$ for S'_2. Similarly, subgames $S_{3,4}$ and $S'_{3,4}$ in Figure 2.7 cover the case where a channel update is proposed by B, although A has already signed a collaborative closing attempt. As in the first case, we have $y = 0$ for the honest closing attempt in $S_{3,4}$ and $y = c$ for dishonest collaborative closing in $S'_{3,4}$. Further in S_3 and S'_3, the proposed update is $(a + p_A, b - p_A)$, hence $x = p_A$, whereas in S_4 and S'_4 it is $(a - p_B, b + p_B)$, thus $x = -p_B$.

The closing game for player B, $G_c(B)$ is defined similarly to $G_c(A)$, with the roles of A

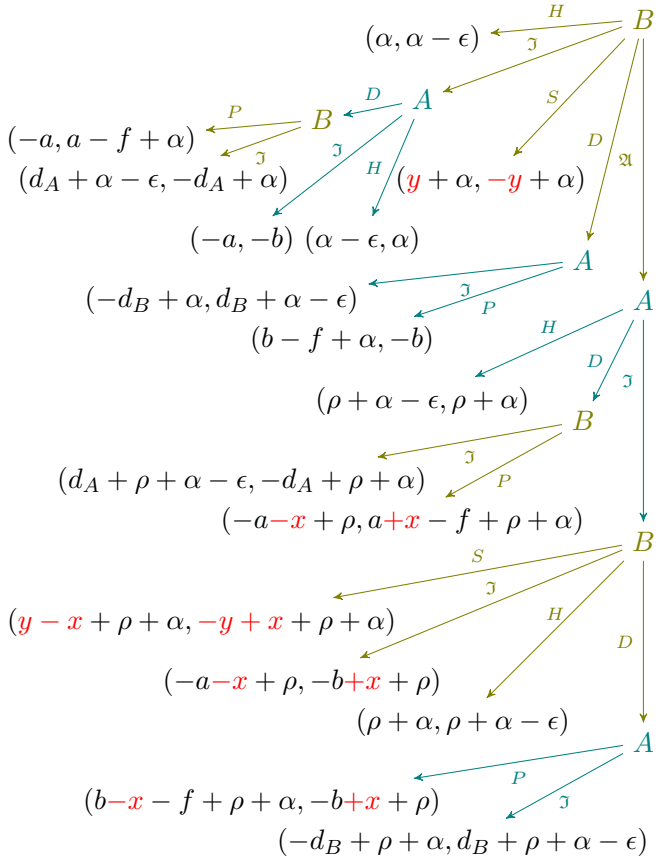


Figure 2.5: Subgames $S_{1,2}$, $S'_{1,2}$ with Update $(a, b) \mapsto (a + x, b - x)$.

and B being swapped in Definition 2.21. Based on the closing games $G_c(A)$ and $G_c(B)$, we consider the closing phase in an off-chain channel as in Figure 2.6 and defined below.

Definition 2.22 (Closing Phase)**.** *The* closing phase *of an off-chain channel modeled by a closing game $G_c(A)$ is initiated in one of three ways: (i) A starts with a closing action C, and thus triggers the closing game $G_c(A)$; (ii) A does not start a closing action, thus performing action ignore $\mathfrak{I}$, but B starts with a closing action C and triggers $G_c(B)$; or (iii) none of the players A and B ever start closing, that is B also choosing action $\mathfrak{I}$, in which case the money stays locked in the channel. Then, we get the EFG Γ_C from Figure 2.6 modeling the closing phase of $G_c(A)$ and $G_c(B)$.*

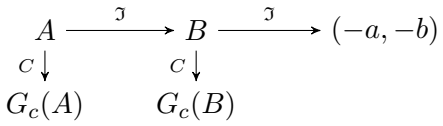


Figure 2.6: Closing Phase Γ_C.

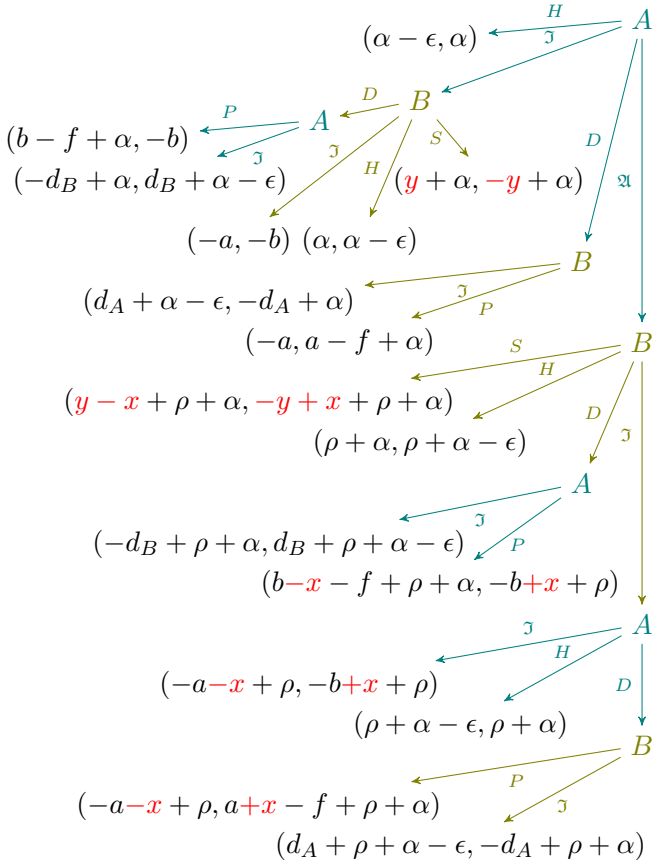


Figure 2.7: Subgames $S_{3,4}$, $S'_{3,4}$ with Update $(a,b) \mapsto (a+x, b-x)$.

2.5 Closing Games for Secure Lightning Channels

We now show that the closing games from Definition 2.21 precisely capture secure closing phases in Lightning channels [PD16]. Namely, the following two terminal histories of closing games model the honest behavior of Lightning: (i) history (H) from Figure 2.4 represents unilateral honest closing of A, yielding utility $(\alpha-\epsilon,\alpha)$; and (ii) history (C_h, S) captures the attempt of A to close collaboratively and honestly, while B signs, with a utility of (α,α). Our security analysis focuses on these two honest histories of Lightning channels.

Definition 2.23 (Honest Closing)**.** *The only* honest histories *in the closing game* $G_c(A)$ *are the terminal histories* (H) *honest unilateral closing and* (C_h, S) *honest collaborative closing. All strategies yielding one of the two histories are considered* honest strategies*.*

In the following, the values d_A (resp. d_B) defined in Table 2.8 (line D) represent the difference of funds between the latest state and the old one that is dishonestly posted on chain by A (resp. B). In other words, if (a,b) is the latest state, the one posted on chain

is $(a + d_A, b - d_A)$ (resp. $(a - d_B, b + d_B)$), thus enabling dishonest closing attempts of profit d_A for A (resp. d_B for B). The values $p_{A,B}$ (Table 2.8, lines U^+, U^-) and c (Table 2.8, line C_c) can respectively be chosen by A and B at the time of the action and do not depend on previous distribution states. Based on this setting, we derive the security properties (P1) and (P2) of Lightning channels as given below.

Theorem 2.1 (Weak Immunity of Honest Behavior – (P1))**.** *The terminal histories* (H) *of honest unilateral closing, and* (C_h, S) *of honest collaborative closing of* $G_c(A)$ *are weak immune, if the channel balances are higher than the fee required in a revocation transaction, that is if* $a \geq f$ *and* $b \geq f$*.*

Proof. Let $a, b \geq f$. For history (H), we consider any strategy σ, where A chooses H after the empty history $\emptyset$, B chooses S after (C_h), P after (D) and H after (C_c). Such a strategy σ yields terminal history (H). If we can show that σ is weak immune, also history (H) is weak immune by Definition III.11. Assume, player A honestly follows σ, i.e., choosing (H), then B's deviation from σ cannot affect the outcome. Thus, A's utility remains non-negative. The other way around, if B follows σ, A can deviate to any initial action C_h, D or C_c, player B's utility never drops below 0, by following strategy σ, as $a \geq f$. Since honest players cannot get negative utility, σ is weak immune.

Similarly, for (C_h, S), we consider any strategy σ', where A chooses C_h initially, player B chooses S after (C_h), P after (D) and H after (C_c). Further, player A takes P after (C_h, D) and H after $(C_h, \mathfrak{I})$, (C_h, U^+) and (C_h, U^-). This strategy σ' yields terminal history (C_h, S). Deviation of A has the same effects as before, never causing the honest B, who follows σ, negative utility. If B deviates now to one of U^+, U^-, $\mathfrak{I}$, D, or H, honest A, following σ, also never gets negative utility, since $b \geq f$. Therefore, σ' and hence history (C_h, S) are weak immune. □

Theorem 2.1 implies that as long as both players have a minimal balance of f in the channel, no honest player can lose money. As such, Theorem 2.1 establishes the security property (P1) ensuring "no honest loss" in the channel.

Further, to ensure the security property (P2) of "no deviation", we require that

$$a - p_B + d_A \geq f \quad \text{and} \tag{2.15}$$

$$b - p_A + d_B \geq f\ . \tag{2.16}$$

To understand the inequations (2.15)-(2.16) consider the history $(C_h, \mathfrak{I}, U^-, \mathfrak{A}, D)$ in Figure 2.4, respectively S_1'. This history formalizes the case where A attempts honest collaborative closing (action C_h) and B ignores it (action $\mathfrak{I}$). Then A proposes an update (action U^-) from state (a, b) to state $(a - p_B, b + p_B)$ and B agrees (action $\mathfrak{A}$). Finally, A closes dishonestly (action D) using the old distribution state $(a - p_B + d_A, b + p_B - d_A)$. Let us also study the options B has. By ignoring A's behavior (action $\mathfrak{I}$), B receives $b + p_B - d_A$ instead of the fair amount $b + p_B$, leaving B with a loss of d_A. By publishing

the revocation transaction (action P), B receives $a + b$ but has to pay the fee f for pushing it on the blockchain, which leads to a win of $a - p_B - f$. Therefore, the win should be greater than the loss; hence

$$a - p_B - f \geq -d_A \quad \Leftrightarrow \quad a - p_B + d_A \geq f \,, \tag{2.17}$$

in order for a rational B to publish the revocation transaction. This, in turn, yields a loss for A and hence discourages A from closing dishonestly, which is necessary for the incentive compatibility (P2) of Lightning's closing phase. By swapping A's and B's roles, we get the prerequisite formulated in (2.16). These extreme cases of dishonest closing subsume the others. Thus, the only preconditions we need in the following Theorem 2.2 are (2.15)–(2.16). In summary, formulas (2.15)–(2.16) ensure that ignoring any dishonest closing attempt is worse than publishing the revocation transaction. Property (P2) is then established by the following theorem.

Theorem 2.2 (Incentive-Compatibility – (P2))**.** *If* $a - p_B + d_A \geq f$ *and* $b - p_A + d_B \geq f$*,*

1. *honest unilateral closing* (H) *is* CR*, but* not *practical.*
2. *honest collaborative closing* (C_h, S) *is* CR*. It is practical iff* $c \neq p_A$*.*

Proof. Let us first prove collusion resilience CR of (H) and (C_h, S). As in the previous proof, we only have to find a strategy σ that yields history (H) and another strategy σ' yielding history (C_h, S), that are CR, to prove (H) and (C_h, S) CR, as defined in Definition III.11. Additionally, collusion resilience is defined on strict subsets of players. Thus, in a two-player game, it considers only deviations of single players and since the summation over one value is the value itself, CR is equivalent to being a Nash Equilibrium in this case. We, therefore, only have to check whether σ and σ' are Nash Equilibria.

For (H), we consider a strategy σ, where player A chooses H initially, player B chooses $\mathfrak{I}$ after (C_h), and $\mathfrak{I}$ after (C_c). Additionally, player B always chooses P after a history $(..., D)$, where the last action was D. Player A takes action H after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$. Further, B takes action $\mathfrak{I}$ after $(C_{h/c}, \mathfrak{I}, U^{+/-})$ (subgames S_1, S_2, S_1', S_2' in Figure 2.5). For A, we finally assume she takes action H after $(C_{h/c}, \mathfrak{I}, U^{+/-}, \mathfrak{I})$. This strategy yields history (H). Deviations from σ of player B cannot change the utility, hence, in particular, cannot increase his utility. Let us consider deviations of player A. A deviation to D at any point in the game leads to A losing all her funds a, which is a strict decrease in utility. This is the case because in σ player B always chooses P after D. Therefore this option is not a threat. If A deviates to C_h or C_c initially, we end up in $(C_h, \mathfrak{I})$, $(C_c, \mathfrak{I})$ respectively. Closing honestly (action H) here leads to the same utility as not deviating. Also, a deviation to $\mathfrak{I}$ does not lead to a better utility. The options she has left are taking U^+ or U^-. Either way, B takes $\mathfrak{I}$ and leaves A with similar choices to before: action H or action $\mathfrak{I}$, both of which do not yield a better utility for her. Since no player can increase their utility by deviating from σ, it is a Nash Equilibrium, and hence (H) is too.

To show that (C_h, S) is a Nash Equilibrium, we consider a strategy σ', where A picks C_h initially, B chooses S after (C_h), P after (D) and H after (C_c). Further, let A pick P after (C_h, D), H after $(C_h, \mathfrak{I})$ and $(C_h, U^{+/-})$ (subgames S_3, S_4 in Figure 2.7). This strategy σ' has terminal history (C_h, S). A deviation of player B, results in either the same utility (choosing action $\mathfrak{I}$, U^+, or U^- after (C_h) and having A taking H) or in strictly worse utility (choosing H or D, where A takes P). Every other deviation has no impact on the resulting history. Similarly, player A cannot profit from deviating. Choosing action H or D initially leads to a strict loss, as B plays P, whereas taking action C_c yields the same utility for A (as B will take action H). Every other deviation has no impact on the history. Hence, no player can increase their utility by deviating, which makes σ' and therefore (C_h, S) a Nash Equilibrium.

To prove the practicality properties, we compute all subgame perfect equilibria of $G_c(A)$. We compute subgame perfect equilibria bottom-up. That is, we start comparing the utility of subtrees with leaves only. In $G_c(A)$, these are for example the subgames after history $(C_h, \mathfrak{I}, D)$ or (C_c, D). For the latter, A is the player to choose the action. To compute the subgame perfect equilibrium, we have to compare all possible utilities for A after (C_c, D). We then replace this internal node labeled A with the utility that yields the best value for A and proceed until we reach the root. If there is no single best choice for a player, then all actions resulting in the best utility have to be considered. Applying this procedure to the subgames S_1-S_4 and S_1'-S_4' we get subgame perfect terminal history $(\mathfrak{A}, H)$ with utility $(\rho + \alpha - \epsilon, \rho + \alpha)$ for S_1. For S_2 we get terminal history (S) yielding (α, α) and $(\mathfrak{I}, H)$, yielding $(\alpha - \epsilon, \alpha)$. For S_3 and S_4 it is $(\mathfrak{I}, S)$ with (α, α). The subgame S_1' has practical history $(\mathfrak{I}, H)$, with $(\alpha - \epsilon, \alpha)$ if $c > p_A$, $(\mathfrak{A}, \mathfrak{I}, S)$ with $(\rho + \alpha, \rho + \alpha)$ if $c = p_A$ and $(\mathfrak{A}, H)$ with $(\rho + \alpha - \epsilon, \rho + \alpha)$ if $c < p$. The subgame S_2' has practical history $(\mathfrak{I}, H)$, yielding $(\alpha - \epsilon, \alpha)$. For S_3' and S_4' in Figure 2.7 we get $(\mathfrak{I}, H)$ with $(\alpha, \alpha - \epsilon)$ and additionally for S_3', if $c = p$, we also have $(\mathfrak{A}, S)$ yielding $(\rho + \alpha, \rho + \alpha)$. All of these results are based on the facts $a - p_B + d_A \geq f$ and $b - p_A + d_B \geq f$ since this causes the revocation transaction to always be better than ignoring the dishonest unilateral closing attempt.

Based on these preliminary results, we can now compute the subgame perfect equilibria for $G_c(A)$ considering multiple practical histories and case splits as stated: If $c = p_A$, then $(C_c, U^+, \mathfrak{A}, S)$ and $(C_c, \mathfrak{I}, U^+, \mathfrak{A}, \mathfrak{I}, S)$ are practical, both yielding $(\rho + \alpha, \rho + \alpha)$. If $c > p_A$, then the histories (C_h, S), $(C_h, U^+, \mathfrak{I}, S)$, $(C_h, U^-, \mathfrak{I}, S)$ and $(C_h, \mathfrak{I}, U^-, S)$ all leading to (α, α) are practical, as well as terminal history $(C_h, \mathfrak{I}, U^+, \mathfrak{I}, H)$, yielding $(\rho + \alpha - \epsilon, \rho + \alpha)$. For $c < p_A$, all the histories and their utilities from $c > p_A$ are practical. Additionally $(C_c, \mathfrak{I}, U^+, \mathfrak{A}, H)$ is subgame perfect in this case and also results in utility $(\rho + \alpha - \epsilon, \rho + \alpha)$.

This shows that (H) is never practical and (C_h, S) is practical if and only if $c \neq p_A$. □

Remark 3 (Explanation of $c \neq p_A$)**.** The condition $c \neq p_A$ in Theorem 2.2 has the following relevance. Player A can, in principle, choose to propose dishonest collaborative closing (action C_c), providing A an unfair advantage of value c. Then, either B (action

U^+) or A (B choosing action $\mathfrak{I}$ to ignore first, then A taking action U^+) can propose a channel update $(a, b) \mapsto (a + c, b - c)$. The value of the update p_A is now equal to the amount player A cheated with in C_c: $p_A = c$. In this special case, the closing game behaves differently. The described histories $(C_c, \mathfrak{I}, U^+)$ and (C_c, U^+) lead to the subgames S'_1 and S'_3 respectively. Let us consider S'_3 with $p_A = c$.

Assume A agrees to the update, action $\mathfrak{A}$, and player B signs the initially unfair collaborative closing attempt of A. Since in the meantime the channel was updated by the exact amount that A tried to cheat with, the pending collaborative closing now contains the fair split. Therefore, both players profit from this course of action, yielding utility $(\rho + \alpha, \rho + \alpha)$. The analog can be achieved in subgame S'_1 with the history $(\mathfrak{A}, \mathfrak{I}, S)$. In fact, for $p_A = c$, those histories are the only practical ones and provide the mutually best outcome possible.

However, updating to $(a + c, b - c)$ first and then closing honestly and collaboratively yields the exact same result. This is why we study the closing game without the possibility of updating after a closing attempt in the next section Section 2.5.1.

We now state our first main security theorem. Since (H) is not practical, a rational player will not play it. Hence, the terminal history (H) is not secure. We get the following security result instead for (C_h, S).

Theorem 2.3 (Security of $G_c(A)$)**.** *If $a \geq f$, $b \geq f$, $a - p_B + d_A \geq f$, $b - p_A + d_B \geq f$, and $c \neq p_A$, then the closing game $G_c(A)$ together with the honest behavior (C_h, S) is* secure.

Proof. As $a \geq f$ and $b \geq f$, we have that (C_h, S) is weak immune (Theorem 2.1). Since $a - p_B + d_A \geq f$, we derive $b - p_A + d_B \geq f$ and $c \neq p_A$, we have that (C_h, S) is also practical and CR (Theorem 2.2). Hence, by Definition 2.19, (C_h, S) is secure. □

Theorem 2.3 implies that for honest and rational players the action of collaborative closing followed by signing (C_h, S) is the best way to close an off-chain channel. It also implies that rational adversaries will cooperate. Further, Byzantine players represent no threat as long as their channel balances are high enough and they do not engage in special cases of channel updates after a collaborative closing attempt.

We note that for proving our security properties (P1)-(P2) in Theorem 2.1–Theorem 2.3, we rely on a succinct analysis of the finite graph properties of the closing game $G_C(A)$ from Figure 2.4. While automated approaches analyzing a finite number of graph properties exist, see e.g. [MMT05, SvS14], these approaches cannot handle (game) graphs where graph leaves contain variables, instead of specific numerical values, which is the case of $G_C(A)$. For such cases, automated reasoning tools, such as theorem provers, need to be combined with graph-theoretic manipulations of $G_C(A)$, an approach we aim to investigate as future work towards automating the security analysis (and proofs) of closing games.

2.5.1 Closing Games without Updates

We will now consider a variation of closing games without updates, as updating is not beneficial for at least one player upon closing. Furthermore, we avoid special cases such as the one described in Remark 3, which should be equivalent to updating before initiating $G_c(A)$, and then closing honestly and collaboratively. As such, the *closing game* $G_c(A)$ *without updates* results from removing all actions U^+ and U^- in Figure 2.4. For the resulting closing game $G_c(A)$ without updates we get the following security result similar to Theorem 2.3.

Theorem 2.4 (Security of $G_c(A)$ without Updates)**.** *If $a \geq f$ and $b \geq f$, then the closing game $G_c(A)$ without updates and together with both honest histories (H) and (C_h, S) is* secure*.*

Proof. We respectively fix honest strategies σ and σ' for histories (H) and (C_h, S); let σ' have A choosing C_h initially, P after (C_h, D) and H after $(C_h, \mathfrak{I})$, and then B choosing S after (C_h), P after (D) and H after (C_c). Infer that the deviation of A causes negative utility for B, whereas the deviation of B leads to non-negative utility for A as $b - f \geq 0$. By Theorem 2.1 we thus have that σ', and therefore (C_h, S), are weak immune. In addition, Theorem 2.1 implies that also (H) is weak immune.

To show practicality, we compute all subgame-perfect terminal histories. From $a \geq f$ and $b \geq f$ we have $a + d_A \geq f$ and $b + d_B \geq f$. Since closing with a dishonest behavior yields utility $a - f + \alpha$, $b - f + \alpha$ respectively, whereas ignoring a dishonest behavior leads to $-d_A + \alpha$ and $-d_B + \alpha$, we conclude that the best choice after action D is always P. Thus, A's best choice after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$ is H. Therefore, B has the two subgame perfect options $\mathfrak{I}$ and S after (C_h), and only $\mathfrak{I}$ after (C_c), yielding thus the following practical histories: history (C_h, S) with utility (α, α); and $(C_h, \mathfrak{I}, H)$, $(C_c, \mathfrak{I}, H)$, and (H) each with utility $(\alpha - \epsilon, \alpha)$. Therefore, both (H) and (C_h, S) are practical.

Note that every practical terminal history is a Nash Equilibrium, since if a deviation could benefit a player, the player would have chosen differently already. As CR is equivalent to Nash Equilibria in two-player games (by Definition 2.5 and Lemma 2.1), we use Definition 2.15 and Lemma 2.1 to conclude that practicality of (H) and (C_h, S) implies collusion resistance CR of (H) and (C_h, S). As (H) and (C_h, S) are both weak immune, practical and CR, by Definition 2.19 we infer that they are also secure. □

Remark 4. Note that the analysis of utilities in the closing game $G_c(A)$ crucially depends on constraints of the underlining ordering that we set in Definition 2.20, and thus on the values of variables $a, b, c, d_{A,B}, f$ in Table 2.8. In general, the bigger ϵ gets in Definition 2.20, the more discouraged is closing unilaterally in Table 2.8, and hence in Figure 2.4. Further, B is more likely to accept a dishonest collaborative closing attempt C_c, as it is better to lose c than to lose ϵ.

We further study what happens if a player has almost no funds left in a channel. In particular, we show that security properties, in particular weak immunity and practicality, are violated in this case, thereby formalizing the following folklore in the community.

Theorem 2.5 (Little Funds)**.** *If $a < f$, then only terminal histories that involve an explicit cheating attempt are weak immune in the closing game $G_c(A)$ without updates. A terminal history involves an explicit cheating attempt if one of its actions is C_c or D.*

Proof. Let σ be any strategy, yielding a history that does not involve an explicit cheating attempt. Then A can deviate to a strategy where A chooses D as its first action. In this case, the honest B gets negative utility, no matter whether B chooses P or $\mathfrak{I}$, since $a < f$. Hence, only histories that involve explicit cheating attempts can be weak immune. □

We next derive the following results on security properties. The results still assume $a, b > 0$. The case where at least one of these values is exactly zero is studied in Section 2.5.2.

Corollary 2.1. *If there exists an old channel state $(a + d_A, b - d_A)$, with $a + d_A < f$, then neither history (H) nor (C_h, S) is weak immune nor practical, but* CR*.*

Proof. We fix the old distribution state such that the difference d_A to the latest state is the value of A's dishonest closing attempt in the closing game. As $a + d_A < f$ implies $a < f$, Theorem V.5 applies. Therefore, neither (H) nor (C_h, S) are weak immune.

In order to show that they are also not practical, we prove instead that the only practical history is $(D, \mathfrak{I})$. Since $a + d_A < f$, $\mathfrak{I}$ is the best choice for B after (D), $(C_h, \mathfrak{I}, D)$ and $(C_c, \mathfrak{I}, D)$. Consequently, A will choose D after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$. If now $b + d_B \geq f$, then A's best choice is P after (C_h, D) and (C_c, D). Thus, B will take S after (C_h) and H after (C_c). In the other case, $b + d_b < f$, A's best option is $\mathfrak{I}$ after (C_h, D) and (C_c, D), thus B's best choice after (C_h) and (C_c) is D, which yields a negative utility for A. Therefore, in both cases, A's only subgame perfect action is D. Hence, $(D, \mathfrak{I})$ is the unique subgame perfect history.

For CR, we show instead that there exist extensions (Definition III.11) σ of (H) and σ' of (C_h, S) that are Nash Equilibria. Let σ be the strategy, where A chooses H, everyone chooses P after a dishonest closing attempt D, B chooses $\mathfrak{I}$ after (C_h) and (C_c) and A chooses H after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$. Then, player B's deviations have no impact, thus cannot increase his utility, and player A's deviations either lead to the same utility as σ, or to the strictly worse utility $-a$. Anyway, no player can deviate to increase their utility and therefore σ and thus (H) is CR. To prove (C_h, S) is CR, we consider the strategy σ', which is the same as σ, except A initially chooses C_h and B chooses S after (C_h). A deviation of A either leads to utility $\alpha - \epsilon$ for her, which is worse than σ's utility, or to utility $-a$, which is even worse. For B, a deviation either leads to the same utility α (taking $\mathfrak{I}$ after (C_h)), to a slightly worse $\alpha - \epsilon$ (choosing H after (C_h)) or to the way

worse $-b$ (D after (C_h)). Every other deviation has no impact on the history. Hence, as nobody profits from deviating, σ' is also a Nash Equilibrium. □

Corollary 2.2. *A rational party should* never*, in any channel, let the opponent's balance fall below f, because at that point the other party can always cause financial loss by closing dishonestly and unilaterally.*

Proof. Once the opponent's balance is below f, that party can start the closing game, therefore the opponent becoming A. Thus, by applying Theorem 2.5, it follows that the opponent can make the rational player lose money by closing unilaterally and dishonestly. If it is not the first time that A's balance is below f and the respective old state contains a higher balance for A than the latest one, then we are even in the situation of Corollary 2.1. It is thus rational for A (practical) to close dishonestly. □

We present an additional theorem, discussing the case where player B has little funds left in the channel. Since the roles of player A and B are arbitrary, it is of little importance because the results give stronger security guarantees than in the case where A has a low balance. Nevertheless, we state it for the sake of completeness.

Theorem 2.6. *If there exists an old state with $b + d_B < f$, but $a \geq f$, then*

1. *(H) is secure.*
2. *(C_h, S) is not practical, not weak immune, but CR.*

Proof. To prove (1), we start by showing weak immunity for a strategy σ with history (H). Consider σ, where A takes action H initially, player B chooses P after (D), S after (C_h) and H after (C_c). Then σ and thus (H) is weak immune, because B's deviations have no impact on the history and A's deviations can never bring B's utility below zero.

Next, we prove the practicality of (H) by computing all subgame perfect equilibria. Since $a \geq f$, the subgame perfect choice after (D), $(C_h, \mathfrak{I}, D)$ and $(C_c, \mathfrak{I}, D)$ is P. Thus, A chooses H after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$. Due to $b + d_B < f$, A's best option after (C_h, D) and (C_c, D) is $\mathfrak{I}$. Hence B's unique subgame perfect choice after (C_c) and (C_h) is D. Thus, A's only best response is H. Therefore, (H) is the only practical history. As practicality implies CR in our case, (H) is secure.

For (2), we just showed that (C_h, S) cannot be practical. Additionally, (C_h, S) is not weak immune, since B could deviate to D after (C_h), in which case A gets negative utility for sure, because of $b + d_B < f$.

Finally, we consider the strategy σ', with history (C_h, S), where A initially chooses C_h, B chooses S after (C_h), both take P in case of a dishonest unilateral closing attempt D, B takes H after (C_c), similarly A takes H after $(C_h, \mathfrak{I})$ and $(C_c, \mathfrak{I})$. Using similar argumentation as before, we conclude that any deviation of a player leads a utility as most as good as σ' for them, but never better. Hence, σ' is a Nash Equilibrium yielding terminal history (C_h, S). □

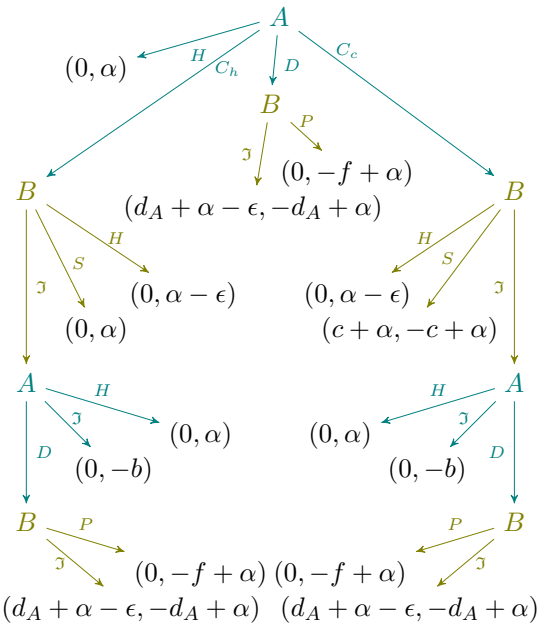


Figure 2.8: Closing game $G_c(A)$ with $a = 0$.

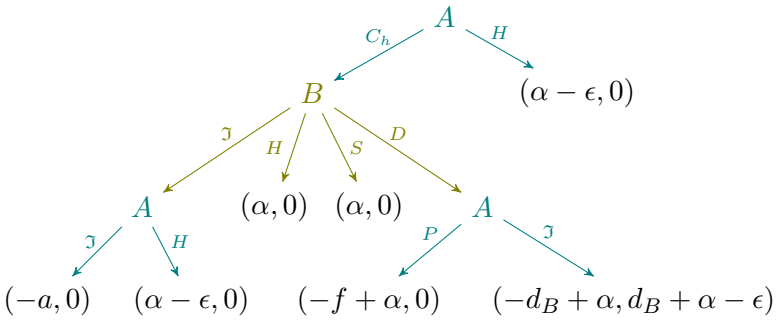


Figure 2.9: Closing game $G_c(A)$ with $b = 0$.

2.5.2 Edge Cases for Closing Games

So far, we have only considered cases where both balances a and b were strictly greater than zero. This is not necessarily the case. Therefore, we consider these cases here. In the first case, $a = 0$, B cannot close dishonestly, as there is no old state that increases his balance. The corresponding simplified game is presented in Figure 2.8.

If $b = 0$ (Figure 2.9), player A cannot close dishonestly, as she cannot take any money from B. Thus, both dishonest unilateral closing D and proposing an unfair split in a collaborative closing attempt C_c are not possible.

Finally, we present results about the two edge cases.

Theorem 2.7. *If $a = 0$ and $b > 0$ (Figure 2.8), then only histories that involve an explicit cheating attempt are weak immune. Additionally, (H) and (C_h, S) are practical if and only if $d_A \geq f$ in every previous state $(d_A, b - d_A)$. In any case, they are* CR.

Proof. We first show that only histories that involve an explicit cheating attempt can be weak immune. Let us consider a history h without a D or C_c action, then A does not initially choose D in h. However, if A deviates to D, then B's utility is negative. Thus, any such history h is not weak immune.

To show both (H) and (C_h, S) are CR, it suffices to show they are Nash Equilibria as before. We therefore consider any strategy σ, where A initially chooses H, player B chooses P after (D), S after (C_h) and H after (C_c). Further, player A takes action H after $(C_h, \mathfrak{I})$. The strategy σ yields history (H). No matter how player A deviates, she always gets utility 0, as she does in σ. Thus, she has no incentive to deviate. Since player B's deviations cannot change the history, he also has no incentive to do so. Therefore, σ and hence (H) is a Nash Equilibrium. Adapting σ, by making A first choice C_h we get strategy σ' which leads to history (C_h, S). As before, A's utility stays 0 no matter how she deviates from σ'. Also, player B cannot improve his utility by changing strategy. Hence, also σ' and therefore (C_h, S) is a Nash Equilibrium.

Towards practicality, we now compute all subgame perfect equilibria. Let $d_A \geq f$. In which case P is the subgame best choice for B after (D), $(C_h, \mathfrak{I}, D)$ and $(C_c, \mathfrak{I}, D)$. Further, after history $(C_c, \mathfrak{I})$, S it is never a best option for B, because it is strictly dominated by H. Therefore, A will get utility zero in any case. This makes (H) a practical history. Similarly for (C_h, S), since S is subgame perfect for B after (C_h).

If now $d_A < f$, then $\mathfrak{I}$ is subgame perfect for B after D. Thus, with similar argumentation as before, $(D, \mathfrak{I})$ is the only practical history. □

Theorem 2.8. *If $a > 0$ and $b = 0$ (Figure 2.9), then*

1. *(H) is secure.*
2. *(C_h, S) is not weak immune, but* CR. *It is practical iff $d_B \geq f$ in every previous state $(a - d_B, d_B)$.*

Proof. We prove (1.) first. The history (H) is weak immune, as B's strategy does not affect the history, since A's initial choice has to be H. Further, A's deviation is irrelevant for B, as he can never get negative utility in this game.

Practicality of (H). We compute subgame perfect equilibria. After history (C_h, D) the subgame perfect choice of A depends on whether $d_B \geq f$. In any case, D is subgame perfect for B after history (C_h). If A chose P, then it is as good as any other choice, yielding 0, otherwise it is the only best option resulting in a positive utility. Thus, A either gets $-f + \alpha$ or $-d_B + \alpha$ if she chooses C_h, both of which is negative. Hence, A's subgame perfect and therefore practical choice is H, yielding the history (H).

The fact that (H) is CR follows from practicality. This shows that (H) is secure if $b = 0$.

For (2), we start showing (C_h, S) is not weak immune. We consider any strategy σ' yielding the history (C_h, S). Assume now, B deviates to D after (C_h), then no matter what A's choice is, she will get a negative utility, thus (C_h, S) is not weak immune.

The collusion resilience of (C_h, S), can be shown by considering a strategy σ' with history (C_h, S), where we additionally fix that A chooses P after (C_h, D). Then B has no incentive to deviate as he always gets utility 0, and A has no incentive as α, which is her utility in σ', is the best possible outcome for her.

To finally show that (C_h, S) is practical iff $d_B \geq f$, we consider A's choice after (C_h, D). The option P is subgame perfect iff $d_B \geq f$. Thus, S is subgame perfect for B iff $d_B \geq f$. For $d_B < f$, D is the better option for B, yielding $(-d_B + \alpha, d_B + \alpha - \epsilon)$. Therefore, C_h is subgame perfect for A iff $d_B \geq f$, in which case the resulting history is (C_h, S). □

The weak immunity result of (H) might be misleading, as B can actually close dishonestly immediately (before A takes action). This is not represented here, but in $G_c(B)$, which is analog to $G_c(A)$ but with swapped roles, as defined in Definition 2.22.

2.5.3 Optimal Strategy for Closing Off-Chain

To summarize, our security analysis based on closing games for Lightning channels yields the following results. Theorem 2.4-Theorem 2.5, together with Corollary 2.1-Corollary 2.2, allow us to derive the optimal strategy for closing an off-chain channel for a rational and suspicious player. We next describe and illustrate this optimal strategy, highlighting the main steps of our security analysis based on Theorem 2.4-Theorem 2.5.

Without loss of generality, we assume the current state of the channel is (a, b).

The player, assumed to be player A, who initiated the closing phase shall:

- try to close honestly and collaboratively (action C_h), if there does not exist an old state $(a + d_A, b - d_A)$, where $d_A > 0$ and $a + d_A < 0$. In case the other player, that is player B, does not sign (action S), player A shall close honestly and unilaterally (action H).

  If player B closed dishonestly and unilaterally (action D), player A shall:

  - state the revocation transaction (action P), if the state used for cheating was $(a - d_B, b + d_B)$, where $d_B > 0$ and $b + d_B \geq f$.
  - ignore the cheating otherwise (action $\mathfrak{I}$), as it yields less loss.

- close dishonestly and unilaterally (action D), if there exists an old state $(a + d_A, b - d_A)$, where $d_A > 0$ and $a + d_A < f$. In this case, player A shall use the old distribution state $(a + d'_A, b - d_A)$, with the highest $d'_A > 0$ that still satisfies $a + d'_A < f$.

The reacting player, in this case assumed to be player B, shall:

- sign the collaborative honest closing attempt (action S), if applicable, if there is no old state $(a - d_B, b + d_B)$, $d_B > 0$ in which the funds of player B are less than f, that is if $b + d_B < f$.
- close honestly and unilaterally (action H), in case of a dishonest collaborative closing attempt (action C_c). This holds, if there is no old state $(a - d_B, b + d_B)$, $d_B > 0$ in which the player B's funds are less then f, that is $b + d_B < f$.
- otherwise ignore (action $\mathfrak{I}$) the collaborative and honest/dishonest closing attempt, if applicable, and close dishonestly and unilaterally (action D), using the old state $(a - d'_B, b + d'_B)$, with the highest $d'_B > 0$ that still satisfies $b + d'_B < f$.
- state the revocation transaction (action P), if player A tried to close dishonestly and unilaterally (action D) with state $(a + d_A, b - d_A)$, where $d_A > 0$ and $a + d_A \geq f$.
- ignore (action $\mathfrak{I}$) if player A closed dishonestly (action D), in the case where $a + d_A < f$, as it yields less loss.

Example 2.10. Let players A and B share a channel with initial balance $(5, 5)$ and let us assume the fee for publishing a revocation transaction $f = 2$. After the first update, let their state be $(3, 7)$. The optimal way for A to close now is C_h and for B to sign. Dishonest closing would cause B to publish the revocation transaction, yielding a loss of 3 for A and a profit of $3 - 2 = 1$ for B.

The next update could be $(1.8, 8.2)$. The best way to close for A is still C_h. Dishonest closing using $(3, 7)$, for example, would still cause B to publish the revocation transaction. Player B would in this case lose $1.8 - 2 = -0.2$, but he would lose more, $7 - 8.2 = -1.2$, by ignoring it.

Another update could be $(1, 9)$. Now the optimal strategy for A to close is D, using the old state $(1.8, 8.2)$. Ignoring the dishonest closing (action $\mathfrak{I}$) brings B -0.8, but proving A's cheating (action P) leads to $1 - 2 = -1$. Hence, a rational B will choose to ignore (action $\mathfrak{I}$), that means B does not publish the revocation transaction.

2.6 Beyond Closing Games for Off-Chain Security

Our game-theoretic analysis so far focused on using closing games to capture security properties of off-chain channels (Section 2.4), and in particular of Lightning channels (Section 2.5). In this section, we show that our game-theoretic formalism from Section 2.3 is expressive enough to analyse more complex protocols than just closing phases in Lightning channels. In particular, we introduce a new EFG, called the *Routing Game* in Section 2.6.1, and use this game in Section 2.6.2 to disprove security of Lightning's routing mechanism amid the Wormhole and Griefing attacks [MMSS+19, KSHB19]. We also discuss a natural extension of our analysis to model other off-chain protocols in Section 2.6.3.

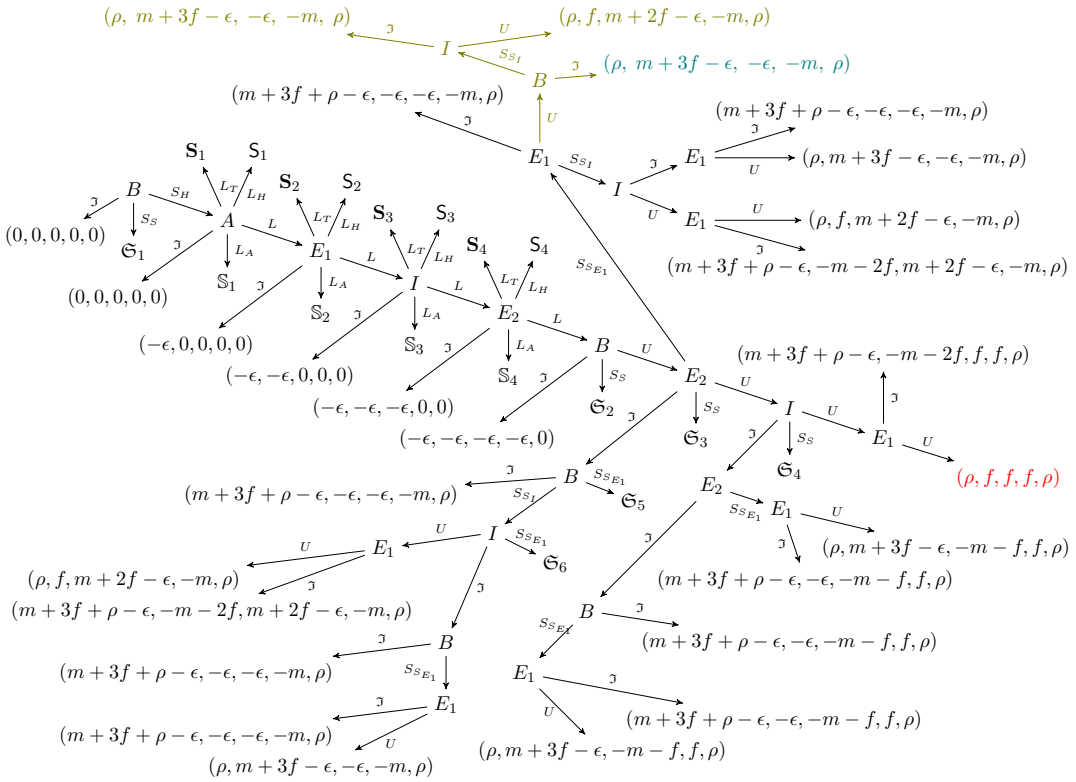


Figure 2.10: Partial Definition of the Lightning's Routing G_{rout} and the Fulgor Model G_{Ful}. The Olive Colored Subtree only applies for G_{rout}. The amount to be routed from player A to player B is $m > 0$, and each of the intermediaries should receive a fee f.

2.6.1 Routing Games for Lightning's Routing Module

We first propose a new EFG, called the *Routing Game*, showing that EFGs can capture actual attacks, in this case the Wormhole attack [MMSS+19] and the Griefing attack [KSHB19], which were overlooked for example in [ZBPBS21]. Specifically, the below defined routing game considers fees f, and supports actions allowing the intermediaries to choose not to claim their money using the secret x but instead to forward it to another intermediary (as explained in Section 2.2.1). Additionally, other deviations such as creating a conditional payment (i.e. HTLC) with a different hash value, a different amount, or a different time-out than expected are also considered. For simplicity, we chose to model our routing game below with five players; however, an arbitrary number of intermediaries can be modeled.

Definition 2.24 (Routing Game G_{rout})**.** *The* routing game $G_{rout} = (N_r, \mathscr{H}_r, P_r, u_r)$ *is an EFG with five players* $N = \{A, E_1, I, E_2, B\}$*, where*

- *the histories* $\mathscr{H}_r$*, the next player function* P_r*, and the utility function* u_r *are defined via the tree representation of Figure 2.10. The utility tuples in Figure 2.10 assign*

the first value to A, the second to E_1, the third to I, the fourth to E_2, and the last to B;

- *the actions of G_{rout} are as listed in Table 2.9.*

The Routing Game The game G_{rout} in Figure 2.10 assumes that player A wants to forward value $m > 0$ to player B and pays each intermediary a participation fee of $f > 0$. It starts with player B initializing the routing mechanism by coming up with a secret and sending its hash to player A (action S_H), or forwarding the secret to someone (action S_S), or not doing anything (action $\mathfrak{I}$). If player B initialized the mechanism, then it is player A's turn to set up an HTLC correctly (action L), incorrectly (actions L_A, L_T, L_H), or not to set up an HTLC (action $\mathfrak{I}$). If player A did set up an HTLC, independent of whether it was done correctly or not, the intermediaries have the same set of actions available, one after the other. When/If all HTLCs were set up, it is again player B's turn to either unlock their HTLC (action U), let it expire (action $\mathfrak{I}$), or share the secret with someone (action S_S). These three types of actions are what all the other players can also choose from once they know the secret (through either secret sharing or the unlocking of the HTLC they set up). Once all HTLCs are unlocked or expired, we reach a leaf.

Subgames of the Routing Game We note that our Routing Game G_{rout} has four types of subgames, as modeled in Figure 2.10 and described next: (i) subgames that result from sending the secret to another player $\mathfrak{S}_i$; (ii) subgames that result from locking a wrong amount of money in the HTLC $\mathbb{S}_i$; (iii) subgames that result from using a wrong time-out in an HTLC $\mathbf{S}_i$; and (iv) subgames that result from using a wrong hash value as lock in the HTLC S_i. We further note that Figure 2.10 only gives a partial model, as not all subgames are presented in Figure 2.10. However, within one type of subgame, the

Table 2.9: Possible Actions in G_{rout}.

S_H	*Sharing* the secret's *Hash* to enable the others to create HTLCs (action 1 in Figure 2.1, Section 2.2.1).
L	*Lock* money, as defined in actions 2–5 in Figure 2.1, in an HTLC.
U	*Unlocking* the money from an HTLC (actions 6–9 in Figure 2.1). Thereby, the secret is revealed to the HTLC's creator.
$\mathfrak{I}$	*Ignoring* all the previous actions and do nothing. If applicable, until the unlockable HTLC has timed out.
S_S	*Sending the Secret* to another player. If it is sent to a specific player (not leading to $\mathfrak{S}_i$) this player is indicated by another subscript.
L_H	*Locking* money in an HTLC, that uses a different *Hash-lock* than described in Figure 2.1.
L_A	*Locking* a different *Amount* of money in an HTLC, than described in Figure 2.1.
L_T	*Locking* money in an HTLC, whose *Time-out* is different from the values described in Figure 2.1.

game trees are similar. Therefore, we provide only one instance of each type, which are the subgames $\mathbb{S}_1$ (Figure 2.11), S_2 (Figure 2.12), and $\mathbf{S}_3$ (Figure 2.13). An instance of a secret-forwarding subgame $\mathfrak{S}$ capturing the Wormhole attack can be seen in G_{rout}, as the subtree after history $(S_H, L, L, L, L, U, S_{S_{E_1}})$.

Subgame $\mathbb{S}_1$ in Figure 2.11 describes the case where player A locks an amount of money in the HTLC which deviates from the expected $m + 3f$. The action L_w (analogous to action L in the main tree) means that the subsequent players follow along and forward the wrong amount to player B, which means the locked value differs from the honest one by $-w$ in each step. Subgame S_2 in Figure 2.12 illustrates the case that player E_1 creates her own secret and uses its hash z as the lock of her HTLC. Action L_z (analogous to action L in the main tree) describes the reusing of hash lock z in the next HTLCs. Lastly, subgame $\mathbf{S}_3$ in Figure 2.13 handles the case, where player I uses a time-out t_3 of the HTLC which is later than the previous ones t_1 and t_2, thereby neglecting the decreasing ordering of time-outs. In Figure 2.13, the action $U_{<t_2}$ means unlocking the HTLC before t_2 times out, thus enabling the other players to unlock too. The action $U_{>t_1}$ stands for unlocking after t_1 (and thus t_2) timed out. Therefore, I and E_1 cannot unlock their respective HLTCs anymore. Finally, action $U_{[t_2,t_1]}$ means unlocking after t_2 has timed out, but the HTLC with time-out t_1 can still be unlocked.

Let us emphasize that the utility function u_r of G_{rout} assigns each player $p \in N$ the relative profit of their routing actions and does not mirror the individual channel balances. It also takes the value ρ of a successful payment and the opportunity cost ϵ into account.

As in the closing games $G_c(A)$ and $G_c(B)$, we aim to align utility and monetary outcome as tight as possible. We adjust the ordering $(\mathbb{U}, \preccurlyeq)$ of Definition 2.20 by not assuming that $\rho \prec \epsilon$, since achieving an update is the ultimate goal of the routing protocol. We also consider the utility relative to the amount due to each party.

2.6.2 Security Analysis of Lightning's Routing Module

Let us recall Figure 2.1 and Figure 2.2, where player A wants to pay another player B money of value m. Since A and B do not share a channel, the three intermediaries E_1, I, and E_2 support the payment, with each receiving a fee $f > 0$ for their collaboration if the payment is successful. Each player who creates an HTLC locks her money for a given time, yielding an opportunity cost of ϵ if the money is returned. If the transaction fails, before anyone has unlocked an HTLC, all parties get utility 0 or $-\epsilon$, depending on whether they created an HTLC or not. Otherwise, the intermediaries' utilities are according to their financial win/loss. The parties A and B both receive ρ once B is paid. Should the transaction fail after B is paid, but before A has paid, she has utility $m + 3f + \rho - \epsilon$; once E_1 collects the money, A's utility is ρ.

In the sequel, we consider the behavior from Figure 2.1 as the only *honest* history in G_{rout}, as also formalized next.

Definition 2.25 (Honest Routing)**.** *The only* honest history *in the routing game G_{rout} is*

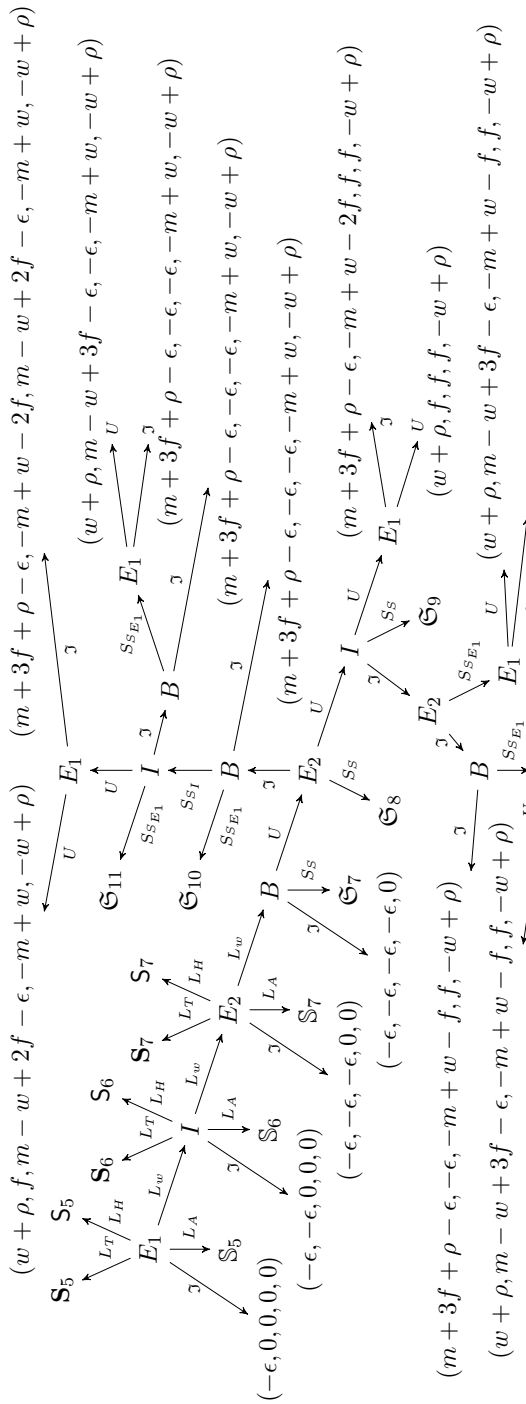


Figure 2.11: Subgame $\mathbb{S}_1$ with Locked Amount $m - w + 3f$.

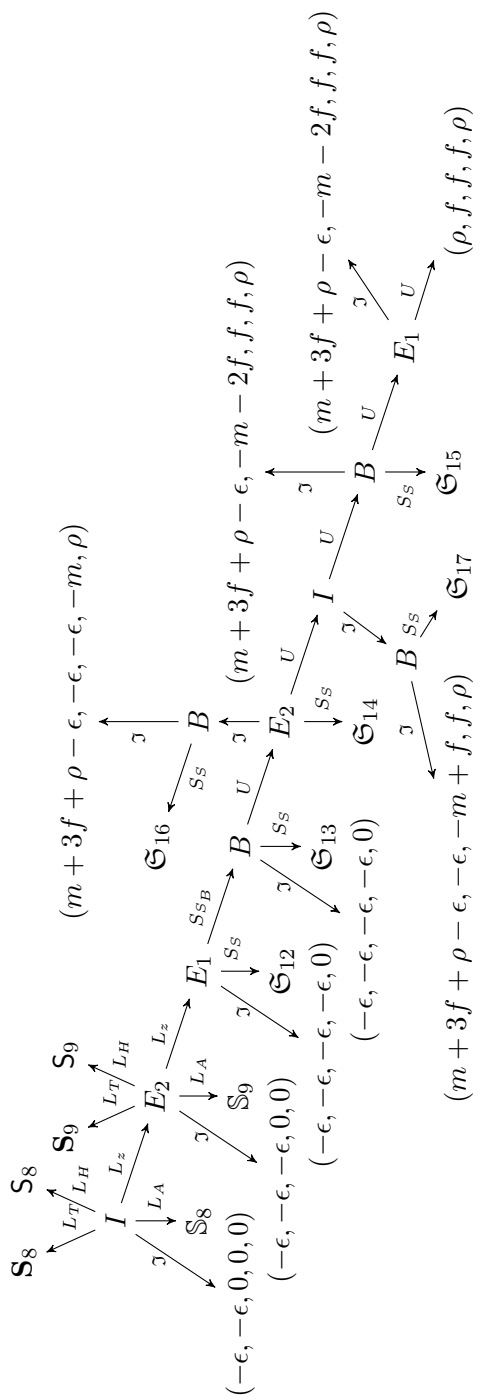


Figure 2.12: Subgame S_2 with Used Hash Lock z.

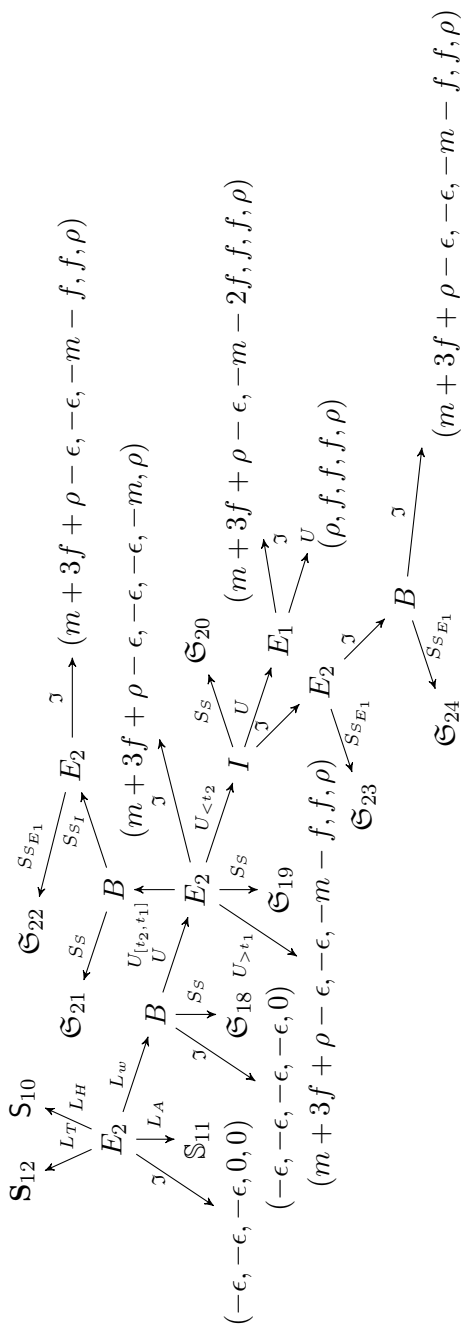


Figure 2.13: Subgame $\mathbf{S}_3$ with Time-Out Ordering $t_3 > t_1 > t_2 > t_4$.

the history $(S_H, L, L, L, L, U, U, U, U)$*. All strategies yielding this history are considered* honest strategies*.*

Using our model G_{rout} and its honest behavior, we derive the following result.

Theorem 2.9 (Vulnerability of G_{rout} to Wormhole Attacks)**.** *The honest behavior* $(S_H, L, L, L, L, U, U, U, U)$ *of the Routing Game* G_{rout} *is not* CR*.*

Proof. The utility of the honest behavior of the routing module $(S_H, L, L, L, L, U, U, U, U)$ is (ρ, f, f, f, ρ) (as indicated in red in Figure 2.10). Let us compare this behavior and utility to the deviating terminal history $(S_H, L, L, L, L, U, S_{S_{E_1}}, U, \Im)$ with a utility of $(\rho,\ m + 3f - \epsilon,\ -\epsilon,\ -m,\ \rho)$ (given in teal in Figure 2.10). It is not hard to argue that the collusion of E_1 and E_2 (and B by not sending the secret to I) strictly profits from the deviation, which yields a joint utility of $3f - \epsilon + \rho$, whereas the honest behavior only yields a joint utility of $2f + \rho$. As such, collusion resistance CR is violated, since no honest player can prevent the Wormhole attack from happening by following any honest strategy (that is, a strategy σ whose history is the honest behavior $(S_H, L, L, L, L, U, U, U, U)$). □

In conclusion, Theorem 2.9 formally proves that Lightning's routing module is susceptible to the Wormhole attack. We further extend this result by noting that not only can G_{rout} capture the Wormhole attack, but also the Griefing attack, as stated below.

Theorem 2.10 (Vulnerability of G_{rout} to Griefing Attack)**.** *The honest behavior, history* $(S_H, L, L, L, L, U, U, U, U)$*, of the Routing Game* G_{rout} *is not weak immune.*

Proof. For showing that history $(S_H, L, L, L, L, U, U, U, U)$ is not weak immune, we prove that no strategy that yields this history is weak immune. Let us consider any such strategy σ. Then, player A has to choose action L after B sent her the secret, that is history (S_H). Assume now E_1 deviates and chooses to ignore (action $\Im$). Then A's utility is $-\epsilon \prec 0$. Hence, history $(S_H, L, L, L, L, U, U, U, U)$ is not weak immune. □

We also obtain the following result as an immediate consequence of Theorem 2.9 and Theorem 2.10.

Corollary 2.3 (Security of Routing Module)**.** *The honest behavior, that is history* $(S_H, L, L, L, L, U, U, U, U)$*, of the Routing Game* G_{rout} *is not secure. Hence, the Routing Game* G_{rout} *is not secure.*

2.6.3 Further Routing Protocols Beyond the Lightning Network

We conclude this work by arguing that our EFG games, either closing or routing games, are not restricted to Lightning networks but can be used for other protocols as well. In the remainder of this section, we illustrate how to model Fulgor [MMSK+17], a payment channel network protocol that fixes the Wormhole attack, but not the Griefing attack.

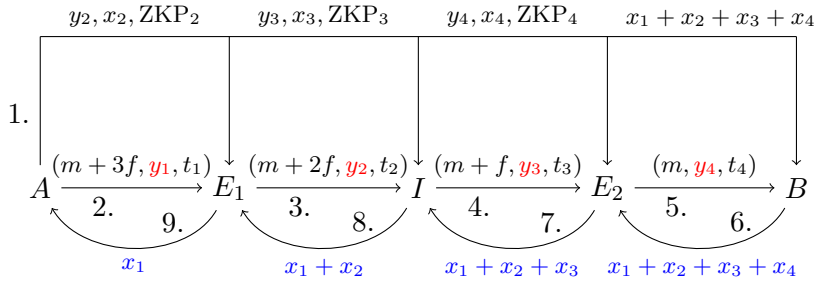


Figure 2.14: Routing in Fulgor.

The routing mechanisms used in Fulgor are similar to Lightning's routing and are similarly based on HTLCs. The main difference lies in the structure of the secrets and their hashes. Indeed, while Lightning uses the same secret x for every HTLC, Fulgor provides a different secret and hash lock for each player.

Fulgor's routing mechanism is illustrated in Figure 2.14, where player A generates different secrets and hash locks at the beginning. The secrets and the hashes relate in the following way: $h(x_1) = y_1$, $h(x_1 + x_2) = y_2$, $h(x_1 + x_2 + x_3) = y_3$, and $h(x_1 + x_2 + x_3 + x_4) = y_4$. Therefore, a player only gets to know a sum of secrets when the right-hand party unlocks and subtracts the secret value received from A to unlock their HTLC. A also provides a zero-knowledge-proof ZKP_i for each intermediary [GO94] to prove that the secrets and hashes constructed this way guarantee successful unlocking of the left HTLC, which is essential to not losing funds.

The game-theoretical (EFG) model of Fulgor G_{Ful} reported in Figure 2.10 looks similar to the routing game G_{rout}, yet with one significant difference. Consider the history $(S_H, L, L, L, L, U, S_{S_{E_1}})$ in Figure 2.10, which enables player E_1 to unlock (action U) the HTLC created by A. In Fulgor, the same history does not enable E_1 to unlock the HTLC. As Figure 2.14 shows, the secrets that E_2 can share after action 6 are x_4 and $x_1 + x_2 + x_3 + x_4$. Further, E_1 only knows x_2; thus, there is no way to compute x_1. This is, however, the value needed to unlock the HTLC created by A. Indeed, Fulgor is not affected by the Wormhole attack. Nevertheless, similarly to Theorem 2.10, the honest behavior of Fulgor is not weak immune, as it is vulnerable to the Griefing attack.

2.7 Conclusion

Our work advocates the use of Extensive Form Games (EFGs) for the game-theoretic security analysis of off-chain protocols. In particular, we introduce two instances of EFGs to model the closing and the routing of the Lightning Network. By doing so, we take the first step towards closing the gap existing security proof techniques have due to using informal arguments about rationality. We express security properties as formal requirements over joint strategies in EFGs, allowing us to establish optimal strategies for closing off-chain and capturing security vulnerabilities amid attacks. Given the theoretical

expressiveness of our EFGs, future work includes the definitions of new games to capture a wider range of off-chain protocols. To overcome the burden of tedious manual analysis, we also plan to leverage SMT solving and/or automated theorem proving in order to provide automated security proofs.

CHAPTER 3

Automated Game-Theoretic Security Analysis

This chapter is based on article [BKK+23b]:

Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. EasyChair Preprint no. 10853, 2023.

This article is an extended version of the publication [BKK+23a]:

Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. In Proceedings of 30th ACM SIGSAC Conference on Computer and Communications Security, pages 1407–1421, New York, NY, USA, 2023.

3.1 Problem Statement

Applications of blockchain technology such as cryptocurrencies [Nak08] and decentralized finance [Woo14] are becoming increasingly popular. Establishing security guarantees of such applications is mostly driven by formal analysis of the underlying cryptographic protocols [MSCB13, Bla14, WLC+19, KNT20]. While powerful, these efforts cannot capture malicious actions that are possible in spite of formal cryptographic guarantees. Game-theoretic security analysis has therefore emerged [ZBPBS21, RAKM23], introducing variants of extensive form games (EFGs) [OR94] for embedding punishment mechanisms within blockchain analysis.

In a nutshell, game-theoretic security analysis enables reasoning about incentive-compatibility: that is, whether malicious yet cryptographically-possible behavior is discouraged via punishment mechanisms. It also enables detecting and even preventing scenarios

that could lead directly to security attack vectors [MMSS+19]. Feasibility of game-theoretic models for investigating an underlying protocol's security partially depends on the game completeness, that is, on expressing all possible interactions between players. In consequence, accurate models are likely to be rather large and complex games. For example, while [RAKM23] introduces a so-called Closing Game to precisely model the closing phase in Bitcoin's Lightning Network [PD16], we show there are trillions of possible joint strategies (combinations of player strategies) for the Closing Game (see Example 3.4). As such, manually analyzing game-theoretic security models is not practically viable.

In this work, we therefore introduce the CheckMate *framework for automating reasoning about game-theoretic security of blockchain protocols.* To the best of our knowledge, CheckMate provides the first automated reasoning framework for enforcing game-theoretic security, (dis)proving, for example, security of real-world protocols used within Bitcoin's Lightning Network (Section 3.6). Related reasoning approaches for (extensive form) games exist [KNPS20], but current techniques are limited to processing games with *numeric* values as game utility variables enacting punishment or reward mechanisms. In CheckMate, we advocate for the use of *symbolic* values, guaranteeing security for every possible numeric value, e.g., every possible account balance in decentralized finance applications.

The distinctive feature of CheckMate is a formalization of security properties over game strategies in such a way that the result can be (dis)proved using "only" first-order arithmetic reasoning with limited quantification (Section 3.4). To this end, *we turn applications of blockchain security into a satisfiability modulo theory (SMT) problem*, by showing that first-order linear real arithmetic provides an expressive logic to formulate and prove game-theoretic security (Lemma 3.1). Our first-order encoding is exact and, unlike the work of [KNPS21, KNPS20], does not feature probabilistic (reward) operators. Instead, we provide a decidable logic for game-theoretic security and omit the computational burden of reasoning with uncertainty.

The added value of our first-order encoding is witnessed when formalizing that deviating from the protocol is never rational (incentive compatibility), and that even if adversaries deviate, honest users are not financially harmed (Byzantine fault-tolerance) (Section 3.2). We show the formalization is sound and complete: our security proofs imply game-theoretic security and vice versa (Corollary 3.1). In this respect, we introduce novel reasoning approaches on top of SMT solving, scaling, and using formal verification not only for enforcing game-theoretic security, but also providing counterexamples and/or refining preconditions where security properties are violated (Section 3.5).

Contributions. We bring the following main contributions

(i) We formalize game-theoretic security properties as first-order arithmetic formulas over EFG joint strategies (Section 3.4), reducing security analysis of blockchain transactions to arithmetic reasoning over honest game histories with symbolic game utilities (Lemma 3.1). The use of symbolic utilities differentiates our work from

other game-theoretic frameworks [KNPS21, KNPS20]: symbolic utilities allow us to avoid concurrent game strategies while providing a deterministic, game-theoretic behavior. As such, we also avoid reasoning with probabilistic (reward) operators.

(ii) Since players may be willing to forgo some intangible assets, such as opportunity cost, but not actual resources, we introduce *weaker immunity* (Definition 3.4), strengthening the state of the art in game-theoretic security analysis. Our formalization is sound and complete with respect to their game-theoretic definitions (Corollary 3.1), and inhabits a decidable fragment of first-order arithmetic.

(iii) Unlike [RAKM23, ZBPBS21], our EFG security properties avoid the use of non-trivial sets, functions and quantifier alternations. We show that this arguably simple logical formalization is both sufficient and necessary to precisely capture game-theoretic security (Theorem 3.3). Moreover, we provide tailored automated reasoning approaches over EFG strategies and bring them into the landscape of SMT solving (Algorithm 3.1).

(iv) Since we reason about symbolic utilities, proving arithmetic relations naturally yields case splits. We guide these case distinctions via unsatisfiable (unsat) core computation in SMT solving (Section 3.5.1).

(v) For EFG properties that are not secure, we provide attack vectors as concrete counterexamples to a violated security property (Section 3.5.2, Algorithm 3.2). In addition, we give weakest ordering conditions on utilities which, if assumed as preconditions, ensure security (Section 3.5.3, Algorithm 3.3).

(vi) We implement our approach in the new tool CheckMate, a fully automated security reasoning engine for EFGs that requires no user guidance (Section 3.6). We evaluate CheckMate on challenging EFGs, including variants of real-world protocols namely closing and routing phases of Bitcoin's Lightning Network. Experiments demonstrate applicability and scalability of CheckMate, (dis)proving security of EFGs with trillions of strategies and thousands of nodes. While for readability's sake, our running examples use simplified game models of Lightning's closing and routing phases (Figures 3.1–3.2), our experiments show that CheckMate succeeds when analyzing the respective protocols in full (Table 3.1).

3.2 Motivating Examples

We motivate and illustrate our work by considering simplified versions of the closing and routing protocol phases of Bitcoin's Lightning Network [RAKM23]. Let us emphasize that our running examples from Figures 3.1–3.2 are simplified only for the sake of readability. In experiments, we also evaluate CheckMate on full models of the respective protocols of Bitcoin's Lightning Network (Section 3.6). In particular, the last two entries of Tables 3.1 and 3.2 report on our results when analyzing Bitcoin's Lightning Network in its *full* complexity.

The closing and routing phases of the Lightning Network are modeled as EFGs and visualized as trees in Figures 3.1 and 3.2. The start of an EFG is the root, and players' choices lead to different branches. The game ends when a leaf is reached, where a player's gain or loss is called their utility.

Example 3.1 (Simplified Closing Game)**.** In the Simplified Closing Game of Figure 3.1, player A starts the game and chooses between three options: closing honestly (H), collaboratively honestly (C_h), or dishonestly (D). If A chooses H, *both players get the benefit* of closing the channel α, but player A has to wait until the closing times out, so the utility is reduced by the opportunity cost ε. If A chooses C_h, player B gets to choose between ignoring (I), i.e. the funds remain locked, or signing (S), where both players get the benefit of closing α. If A chooses to close dishonestly with some deviating amount d_A, then if B chooses to ignore (I), the funds for B are lost; however, if B proves (P) the attempt was dishonest, all of A's funds (a) are redistributed to B, but the transaction fee f has to be paid. Note that the utilities in the leaves of the game tree are actual variables (α, a, f, etc.), not numeric values, and α, ε are infinitesimals (see Section 3.3).

Example 3.2 (Simplified Routing Game)**.** CheckMate can handle games with any finite number of players. To show how an attack vector can arise from collusion between players and outline the main structure of the model of the routing protocol, we include in Figure 3.2 an EFG with five players, modeling a Simplified Routing Game. Player A is the initiator of the routing transaction, player B is the receiver and players P_1, P_2 and P_3 are intermediaries. The routing starts when player B sends a hash of a secret to player A; this step is modeled with action S_H. Then, a so-called locking phase follows (the four actions L), where players lock funds for the next player in the routing path to unlock, provided they can present B's secret: player A locks the amount $m + 3f$ (where f represents the routing fee), player P_1 locks $m + 2f$, player P_2 locks $m + f$ and player P_3 locks m. Next, the game enters an unlocking phase, where players choose between the honest action U of unlocking their contracts, or to ignore unlocking (I_U), resulting in a state where all contracts still locked expire and a leaf is reached. The end utilities of players depend on which contracts are unlocked, which are expired and additionally on two subjective values: the benefit of updating, modeled as a positive infinitesimal ρ, and the opportunity cost, modeled as an infinitesimal ε.

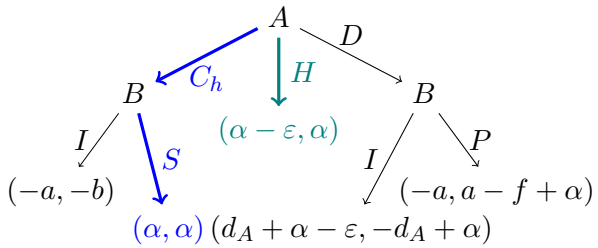


Figure 3.1: Simplified Closing Game with $\alpha, a, \varepsilon, f, d_A > 0$ and α, ε infinitesimals.

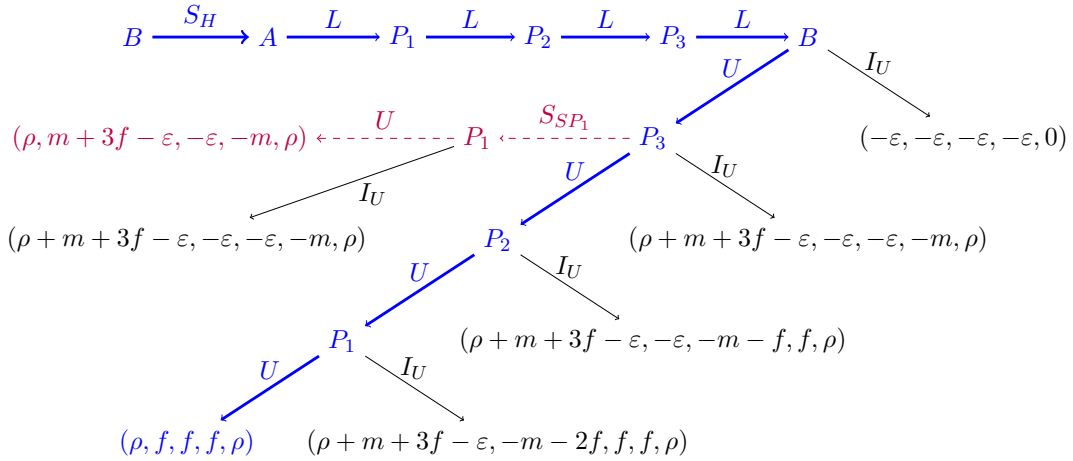


Figure 3.2: Simplified Routing Game with $m, f, \rho, \varepsilon > 0$ and ρ, ε infinitesimals.

The path that represents honest behavior is depicted in thick blue lines. CheckMate disproves security of the game, as the path depicted in dashed purple deviates from the honest behavior and corresponds to the so-called Wormhole attack [MMSS+19]. Within this dishonest behavior (attack), player P_3 can additionally choose to share the secret with player P_1 and thus bypass player P_2 entirely. If P_1 chooses to unlock, it results in a negative utility $(-\varepsilon)$ for player P_2, who is further deprived of earning the routing fee f for enabling the transaction, which they would get in the honest scenario depicted in blue.

3.3 Preliminaries

We introduce game-theoretic material needed for our work. We assume familiarity with standard first-order logic [Smu95], linear (real) arithmetic [Sch99], and SMT solving over both [BT18, DDM06a, GDM09].

3.3.1 Game Theory

As in the previous chapter, we define a *game* to be a static finite object with finitely many *players*. Players choose from finitely many *actions* until the game ends, whereupon they receive a *utility*. We focus on perfect-information *Extensive Form Games* (EFGs) in which players choose actions sequentially with full knowledge of all previous actions. Games may yield collective benefit or loss, i.e., they are not necessarily *zero-sum*.

The following definition of an extensive form game is more intuitive yet equivalent to the one in Chapter 2 (Definition 2.9), as indicated by Definition 2.10. Since the previous chapter advocates for the use of EFGs rather than the simpler normal form games (NFGs), a more technical definition, which is easier to relate to NFGs, was necessary.

Definition 3.1 (Extensive Form Game — EFG)**.** *An* extensive form game $\Gamma = (N, G)$ *is determined by a finite non-empty set of players* N *together with a finite tree* $G = (V, E)$*. A game path* $h = (e_1, ..., e_n), e_i \in E$ *starting from the root of* G *is called a* history*. We denote the set of histories by* $\mathscr{H}$*. There is a bijection between nodes* $v \in V$ *and histories* $h \in \mathscr{H}$ *that lead to these nodes.*

- *A history that leads to a leaf is called* terminal *and belongs to the set of* terminal histories $\mathscr{T} \subseteq \mathscr{H}$*. Terminal histories* t *are associated with a* utility *for each player.*
- Non-terminal histories h *are those histories that are not terminal* $h \in \mathscr{H} \setminus \mathscr{T}$*. Non-terminal histories* h *have a player* $P(h) \in N$ *whose turn it is, who may choose from a set of possible actions* $A(h)$ *to take after* h*.*

In game theory, utilities are usually real- or integer-valued numeric constants. Similarly to [RAKM23], in our work, we, however, consider utilities as *symbolic* terms in linear arithmetic, capturing all possible utilities with certain constraints. We evaluate variables and constants over the real numbers $\mathbb{R}$ extended by a finite set of *infinitesimals*, closer to zero than any real number. Infinitesimals model subjective inconveniences or benefits that do not relate directly to funds, such as opportunity cost. For our purposes, we model infinitesimals by considering linear terms over $\mathbb{R} \times \mathbb{R}$, ordered lexicographically: the first component represents the real part, the second the infinitesimal. We write real for the first projection and avoid writing pairs; that is, we write 0 for $(0, 0)$. In the sequel, we use $a, b, c \ldots$ for real variables, and write $\alpha, \beta, \gamma, \ldots$ for infinitesimals. The utility term $a + \alpha - \varepsilon$ is therefore represented in our work as $(a, 0) + (0, \alpha) - (0, \varepsilon)$, that is $(a, \alpha - \varepsilon)$.

Example 3.3. The Simplified Closing Game has two players $N = \{A, B\}$. After empty history $\emptyset$, it is the turn of player $P(\emptyset) = A$ to choose from actions $A(\emptyset) = \{H, C_h, D\}$. After terminal history (H), player A receives utility $\alpha - \varepsilon$ and B receives α.

While utilities depend only on terminal histories, we relate utilities to *joint strategies*, facilitating the formulation of security properties in the sequel. The next definition is also consistent with the analog ones from Chapter 2.

Definition 3.2 (EFG Properties)**.** *Let* $\Gamma = (N, G)$ *be an EFG.*

Joint Strategy *A* joint strategy σ *is a function mapping every non-terminal history* $h \in \mathscr{H} \setminus \mathscr{T}$ *to an action* $a \in A(h)$*. The set of joint strategies is* $\mathscr{S}$*.*

Single Strategy *A* strategy $\sigma_p \in \mathscr{S}_p$ *of player* p *is a function mapping non-terminal histories* $h \in \mathscr{H} \setminus \mathscr{T}$ *with* $P(h) = p$ *to an action* $a \in A(h)$*. Similarly, a group of players* $S \subset N$ *may have a strategy* $\sigma_S \in \mathscr{S}_S$*.*

Strategy Deviation *If player* p deviates *from a joint strategy* $\sigma \in \mathscr{S}$ *with another strategy* $\tau_p \in \mathscr{S}_p$*, the resulting joint strategy is denoted* $\sigma[\tau_p/\sigma_p]$*. Similarly, for a deviating group of players* $S \subset N$*, we write* $\sigma[\tau_S/\sigma_S]$*.*

Resulting History *The* resulting terminal history $H(\sigma)$ *of a strategy* σ *is the unique history obtained by following chosen actions in* σ *from root to leaf.*

Extended Strategy *An* extended joint strategy $\beta \in \mathscr{S}$ *of a history* $h \in \mathscr{T}$ *is a strategy whose resulting history is* h*. Thus,* $H(\beta) = h$*.*

Players' Subhistories *Let* $\mathscr{H}_t^S$ *denote the* set of non-empty histories *leading to terminal history* t *where the last turn was one of the players'* $p \in S$*. That is,* $\mathscr{H}_t^S := \{(h, a) \mid \exists h'.\, t = (h, a, h') \wedge P(h) \in S\}$*. For simplicity, let* $\mathscr{H}_t^p := \mathscr{H}_t^{\{p\}}$*.*

Utility Function *The* utility function $u_p(\sigma)$ *assigns a utility for every joint strategy* $\sigma \in \mathscr{S}$ *to player* $p \in N$*. We sometimes write all player utilities for a joint strategy as* $u(\sigma)$*, denoting a tuple of size* $|N|$*. Since utilities only depend on* σ*'s terminal history* $h = H(\sigma)$*, we define* $u_p(h) := u_p(\sigma)$*.*

Subgames Subgames $\Gamma_{|h}$ *of* Γ *are formed from the same set* N *of players and a subtree of* G*, and are therefore identified by a history* h *leading to the subtree* $G_{|h}$*. Histories* $\mathscr{H}_{|h}$ *of* $\Gamma_{|h}$ *are histories in* $\mathscr{H}$ *with prefix* h*, and similarly for the utility function* $u_{|h}$ *and strategies* $\sigma_{|h} \in \mathscr{S}_{|h}$*.*

Example 3.4. In Figure 3.1, a joint strategy τ could be player A taking action H initially, with player B taking S after (C_h) and P after (D). Player A's single strategy τ_A takes action H initially. Player B receives $u_B(\tau) = \alpha$. The history resulting from τ is (H), and τ is a strategy extending history (H). The subgame for history (C_h) has players $N = \{A, B\}$ and has a tree where player B must choose between action I with utility $(-a, -b)$ and action S with utility (α, α).

The Simplified Closing Game has $3 \cdot 2 \cdot 2$ joint strategies as player A chooses one out of three possible actions, and independently of that, B picks one action out of two in both subtrees. Similarly, we reach the conclusion that the Closing Game [RAKM23] listed in Table 3.1 has $1.6307\ldots \cdot 10^{13}$ (16 trillion) joint strategies.

3.3.2 Game-Theoretic Security Properties

We restate the following concepts and definitions from Chapter 2 for readability. Since an adversary may perform an attack for one of two reasons (personal gain or harming somebody), a protocol is *game-theoretically secure* according to [ZBPBS21, RAKM23], if the following two properties hold:

(P1) (Byzantine Fault-Tolerance) Even in the presence of adversaries, honest players do not suffer loss; thus, in a secure protocol an honest player will not receive negative utility, independent of others' behavior. Therefore, there are no "attacks" where somebody is harmed.

(P2) (Incentive-Compatibility) Rational agents do not deviate from the honest behavior, as the honest behavior yields the best payoff. Hence, in a secure protocol, a rational "attacker" is behaving honestly and no adversary gets personal gain by deviation.

In the sequel, we fix an arbitrary EFG $\Gamma = (N, G)$ and give all definitions relative to Γ. Based on [RAKM23], property (P1) is ensured by *weak immunity* as follows.

Definition 3.3 (Weak Immunity)**.** *A joint strategy $\sigma \in \mathscr{S}$ in EFG Γ is* weak immune *if all players p that follow σ always receive non-negative utility:*

$$\gamma_{wi}(\sigma): \qquad \forall p \in N\ \forall \tau \in \mathscr{S}.\ u_p(\tau[\sigma_p/\tau_p]) \geq 0\ . \tag{γ_{wi}}$$

Strategy τ from Example 3.4 is weak immune, as long as $\alpha \geq \varepsilon$ and $a \geq f$.

In our work, we found weak immunity to be too restrictive (see Section 3.6). We therefore refine Chapter 2 and propose *weaker immunity* to ensure (P1) as follows:

Definition 3.4 (Weaker Immunity)**.** *A joint strategy σ in EFG Γ is* weaker immune *if all players p that follow σ always receive at least a negative infinitesimal:*

$$\gamma_{weri}(\sigma): \qquad \forall p \in N\ \forall \tau \in \mathscr{S}.\ \mathsf{real}(u_p(\tau[\sigma_p/\tau_p])) \geq 0\ . \tag{γ_{weri}}$$

(P2) is ensured by *collusion resilience* and *practicality*. Collusion resilience requires honest behavior to yield the best payoff, even in the presence of collusion.

Definition 3.5 (Collusion Resilience)**.** *A joint strategy σ in EFG Γ is* collusion resilient *if colluding players $S \subset N$ cannot profit from deviation:*

$$\gamma_{cr}(\sigma): \qquad \forall S \subset N\ \forall \tau \in \mathscr{S}.\ \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\sigma[\tau_S/\sigma_S])\ . \tag{γ_{cr}}$$

Strategy τ from Example 3.4 is not collusion resilient, since player A could deviate by choosing C_h initially and obtain α, while by following τ they receive only $\alpha - \epsilon$. Practicality ensures that for all player decisions, the honest behavior is also "greedy": if all players act selfishly (that is, maximizing their own utilities), the honest choice yields the best utility.

Definition 3.6 (Practicality)**.** *A joint strategy σ in EFG Γ is* practical *if it is a subgame perfect equilibrium, i.e., a Nash equilibrium in every subgame:*

$$\begin{aligned} \gamma_{pr}(\sigma): \quad & \forall h \in \mathscr{H}\ \forall p \in N\ \forall \tau \in \mathscr{S}_{|h}. \\ & u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h}[\tau_p/\sigma_{|h_p}])\ . \end{aligned} \tag{γ_{pr}}$$

Strategy τ from Example 3.4 is not practical. In the subgame after history (C_h), the selfish choice for B is to choose action S. Assuming player B acts this way, player A's greedy strategy is to choose action C_h initially with expected utility α instead of $\alpha - \epsilon$. Using (γ_{wi}), (γ_{cr}), and (γ_{pr}), we formalize *security* as in [RAKM23]:

Definition 3.7 (Security)**.** *A terminal history h^* of an EFG Γ is* secure *if there are three strategies extending h^* that satisfy (γ_{wi}), (γ_{cr}) and (γ_{pr}), respectively.*

Our notion of security given in Definition 3.7 ensures that players can defend against every attack. As the defense strategy may vary depending on the attack, different strategies for (γ_{wi}), (γ_{cr}), and (γ_{pr}) are allowed.

3.4 First-Order Arithmetic Theory of Security Properties

We now introduce our first-order formalization of game-theoretic security by exploiting and adjusting the EFG security properties of Section 3.3 within the first-order theory of linear real arithmetic. Our formalization ensures that if a first-order (security) formula is satisfiable, a model for the formula provides a *joint strategy for a given honest history* of an EFG (Section 3.4.1). We present our first-order formalization piecewise, as constraints on such models, imposing, among others, that models must form a joint strategy, the joint strategy should result in a given honest history, and user-supplied assumptions should be considered (Section 3.4.2–Section 3.4.4). We consider the game utilities to be symbolic terms evaluated over pairs of real values, as mentioned in Section 3.3.1. We therefore universally quantify their symbolic variables in our encoding. As before, EFG $\Gamma = (N, G)$ is arbitrarily fixed.

3.4.1 Joint Strategies and Honest Histories

A joint strategy for an EFG $\Gamma = (N, G)$ selects exactly one action for each internal node of the tree. We introduce Boolean *action variables* v_a^h to indicate whether at non-terminal history h a player chooses action a, and constrain these variables v_a^h so that exactly one variable is assigned for each h. We thus have

$$\bigwedge_{h \in \mathscr{H} \setminus \mathscr{T}} \underbrace{\left(\bigvee_{a \in A(h)} v_a^h \right)}_{\text{(ALO)}} \wedge \underbrace{\bigwedge_{a_i, a_j \in A(h),\, a_i \neq a_j} \left(\neg v_{a_i}^h \vee \neg v_{a_j}^h \right)}_{\text{(AMO)}}. \qquad (\phi_{\text{strat}})$$

Constraint (ALO) ensures that each non-terminal history h has at least one action variable set. The at-most-one constraint (AMO) [NM15] ensures that no more than one v_a^h is set. Our next lemma then concludes that a model I of ϕ_{strat} uniquely describes a joint strategy $\sigma \in \mathscr{S}$ and vice versa.

Lemma 3.1 (Model-Strategy Translation)**.** *Consider the EFG Γ with joint strategies $\mathscr{S}$. Let $\mathcal{M}$ be models of the formula ϕ_{strat}. Then, $f : \mathcal{M} \to \mathscr{S}$, where*

$$f(\mathsf{I}) = \sigma \quad \iff \quad \forall h \in \mathscr{H} \setminus \mathscr{T} : \sigma(h) = a \leftrightarrow \mathsf{I}(v_a^h) = \top \qquad (3.1)$$

is a well-defined bijection.

Proof. Recall that

$$\mathscr{S} = \{\sigma |\ \sigma : \mathscr{H} \setminus \mathscr{T} \to \bigcup_{h \in \mathscr{H} \setminus \mathscr{T}} A(h),\ \sigma(h) \in A(h)\} \quad \text{and}$$
$$\mathcal{M} = \{\mathsf{I} |\ \mathsf{I} : (v_a^h)_{a \in A(h)}^{h \in \mathscr{H} \setminus \mathscr{T}} \to \{\texttt{true}, \texttt{false}\},\ \mathsf{I}(\phi_{\text{strat}}) = \texttt{true}\}\,.$$

We start by showing that f is well-defined. Let $\mathsf{I} \in \mathcal{M}$ and define $\sigma := f(\mathsf{I})$. From $\mathsf{I}(\phi_{\text{strat}}) = \texttt{true}$, it follows that for all $h \in \mathscr{H} \setminus \mathscr{T}$, there exists an action $a \in A(h)$ such that $\mathsf{I}(v_a^h) = \texttt{true}$ as well as there cannot be two different $a_i, a_j \in A(h)$ such that $\mathsf{I}(v_{a_i}^h) = \mathsf{I}(v_{a_j}^h) = \texttt{true}$. By definition of f, this says exactly that for every $h \in \mathscr{H} \setminus \mathscr{T}$, there exists precisely one action $a \in A(h)$ such that $\sigma(h) = a$. Therefore, σ is a function from $\mathscr{H} \setminus \mathscr{T}$ to $\bigcup_{h \in \mathscr{H} \setminus \mathscr{T}} A(h)$ with $\sigma(h) \in A(h)$ for all h. Hence, $\sigma \in \mathscr{S}$.

Next, we show the injectivity of f. Let $\mathsf{I}, \mathsf{I}' \in \mathcal{M}$, $\mathsf{I} \neq \mathsf{I}'$. Then, there exists a v_a^h such that $\mathsf{I}(v_a^h) \neq \mathsf{I}'(v_a^h)$. Without loss of generality, we assume $\mathsf{I}(v_a^h) = \texttt{true}$. For $\sigma := f(\mathsf{I})$, $\sigma' := f(\mathsf{I}')$, it follows $\sigma(h) = a \neq \sigma'(h)$. Thus, $\sigma \neq \sigma'$ and hence, f is injective. Lastly, we show the surjectivity of f. We pick an arbitrary $\sigma \in \mathscr{S}$ and consider I with $\mathsf{I}(v_a^h) = \texttt{true}$ iff $\sigma(h) = a$. This I is a function $(v_a^h)_{a \in A(h)}^{h \in \mathscr{H} \setminus \mathscr{T}} \to \{\texttt{true}, \texttt{false}\}$. Since σ is a function with $\sigma(h) \in A(h)$, we know that for all $h \in \mathscr{H} \setminus \mathscr{T}$, there exists exactly one $a \in A(h)$ such that $\mathsf{I}(v_a^h) = \texttt{true}$. Therefore, I is a model of ϕ_{strat} and $\mathsf{I} \in \mathcal{M}$. We conclude f is surjective and thus a well-defined bijection. □

Lemma 3.1 is the crux of our work, *reducing game-theoretic security analysis to satisfiability modulo first-order linear real arithmetic:* game-theoretic security holds iff first-order formulas describing security properties are satisfiable. In what follows, we introduce the first-order formulas capturing game-theoretic security, which then together with Lemma 3.1 enable the automation of (dis)proving game-theoretic security (Section 3.5). To this end, we extend honest histories and hence further constrain EFG joint strategies. We do so by ensuring that all action variables in the honest history are set. That is, for an honest history $h^* = (a_1, \ldots, a_n)$, we obtain

$$v_{a_1}^{\emptyset} \wedge \cdots \wedge v_{a_n}^{(a_1, \ldots, a_{n-1})}. \tag{ϕ_{hist}}$$

Example 3.5. Consider the Simplified Closing Game with honest history (C_h, S). From ϕ_{strat} and ϕ_{hist}, we obtain the following constraints on action variables:

$$\underbrace{\left(v_H^{\emptyset} \vee v_{C_h}^{\emptyset} \vee v_D^{\emptyset}\right) \wedge \left(\neg v_H^{\emptyset} \vee \neg v_{C_h}^{\emptyset}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } A.}$$
$$\wedge \underbrace{\left(\neg v_H^{\emptyset} \vee \neg v_D^{\emptyset}\right) \wedge \left(\neg v_{C_h}^{\emptyset} \wedge v_D^{\emptyset}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } A.}$$

$$\wedge \underbrace{\left(v_I^{(C_h)} \vee v_S^{(C_h)}\right) \wedge \left(\neg v_I^{(C_h)} \vee \neg v_S^{(C_h)}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } B.}$$

$$\wedge \underbrace{\left(v_I^{(D)} \vee v_P^{(D)}\right) \wedge \left(\neg v_I^{(D)} \vee \neg v_P^{(D)}\right)}_{\text{Constraints from } \phi_{\text{strat}} \text{ for } B.} \wedge \quad \underbrace{v_{C_h}^{\emptyset} \wedge v_S^{(C_h)}}_{\text{Constraints from } \phi_{\text{hist}}.}$$

3.4.2 Weak and Weaker Immunity

A joint strategy is weak immune (γ_{wi}) if the utility of each player following the strategy is non-negative, no matter how other players behave. For each possible terminal history, we thus need to ensure that if a player takes the corresponding actions, the resulting utility for the player is greater than or equal to 0. The set of non-terminal histories leading to a terminal history t where it is the turn of player p is $\mathscr{H}_t^p$, as defined in Definition 3.2. We then formalize weak immunity as

$$\bigwedge_{p \in N} \bigwedge_{t \in \mathscr{T}} \left[\bigwedge_{(h,a) \in \mathscr{H}_t^p} v_a^h \right] \rightarrow u_p(t) \geq 0. \tag{ϕ_{wi}}$$

Moreover, we express weaker immunity (γ_{weri}) as

$$\bigwedge_{p \in N} \bigwedge_{t \in \mathscr{T}} \left[\bigwedge_{(h,a) \in \mathscr{H}_t^p} v_a^h \right] \rightarrow \mathsf{real}(u_p(t)) \geq 0. \tag{ϕ_{weri}}$$

To ensure that models of ϕ_{wi} and ϕ_{weri} yield weak(er) immune joint strategies of our EFG Γ, we respectively combine the constraints ϕ_{strat} and ϕ_{hist} with ϕ_{wi} and ϕ_{weri}.

Example 3.6. In order to find a weak immune joint strategy for the Simplified Closing Game, we add the following formula to the constraints from Example 3.5:

$$\left.\begin{aligned} (v_H^{\emptyset} \rightarrow \alpha - \varepsilon \geq 0) \wedge (v_{C_h}^{\emptyset} \rightarrow -a \geq 0) \\ \wedge (v_D^{\emptyset} \rightarrow d_A + \alpha - \varepsilon \geq 0) \\ \wedge (v_D^{\emptyset} \rightarrow -a \geq 0) \wedge (v_{C_h}^{\emptyset} \rightarrow \alpha \geq 0) \end{aligned}\right\} \text{Constraints for } A.$$

$$\left.\begin{aligned} \wedge\, \alpha \geq 0 \wedge (v_I^{(C_h)} \rightarrow -b \geq 0) \\ \wedge (v_S^{(C_h)} \rightarrow \alpha \geq 0) \\ \wedge (v_I^{(D)} \rightarrow -d_A + \alpha \geq 0) \\ \wedge (v_P^{(D)} \rightarrow a - f + \alpha \geq 0) \end{aligned}\right\} \text{Constraints for } B.$$

Note that the first constraint for player B does not contain an implication. This is because player B does not make a choice at terminal history (H) and consequently, the empty antecedent is omitted. If we consider the honest history (C_h, S), we can simplify the formula by ϕ_{hist} and propositional reasoning, but weak immunity does not hold as

the constraint $v^{\emptyset}_{C_h} \to -a \geq 0$ is not satisfied for $a > 0$. When in turn checking for weaker immunity, we can disregard infinitesimal terms, so the constraints from ϕ_{weri} simplify to:

$$\underbrace{(v^{\emptyset}_{C_h} \to -a \geq 0) \wedge (v^{\emptyset}_{D} \to d_A \geq 0) \wedge (v^{\emptyset}_{D} \to -a \geq 0)}_{\text{Constraints for } A.} \quad \wedge$$
$$\underbrace{(v^{(C_h)}_{I} \to -b \geq 0) \wedge (v^{(D)}_{I} \to -d_A \geq 0) \wedge (v^{(D)}_{P} \to a - f \geq 0)}_{\text{Constraints for } B.}$$

3.4.3 Collusion Resilience

Within a collusion resilient joint strategy, no subgroup of players benefits when deviating from the honest behavior (γ_{cr}). We thus need to ensure that all possible deviations of a group of players receive *total* utility less than that obtained by honest behavior. Hence, our formalization in this respect needs to include only the action variables of the players that do *not* belong to the deviating subgroup, as these are the players whose choices are in accordance with the desired joint strategy. For an honest history h^*, we formalize collusion resilience as:

$$\bigwedge_{S \subset N} \bigwedge_{t \in \mathscr{T}} \left[\bigwedge_{(h,a) \in \mathscr{H}_t^{N \setminus S}} v^h_a \right] \to \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(t). \tag{ϕ_{cr}}$$

Example 3.7. Collusion resilience of the Simplified Closing Game is captured by:

$$\left.\begin{aligned} (v^{(C_h)}_{I} \to \alpha \geq -a) \wedge (v^{(C_h)}_{S} \to \alpha \geq \alpha) \\ \wedge (v^{(D)}_{I} \to \alpha \geq d_A + \alpha - \varepsilon) \\ \wedge (v^{(D)}_{P} \to \alpha \geq -a) \end{aligned}\right\} \text{Subgroup } \{A\}.$$
$$\left.\begin{aligned} \wedge (v^{\emptyset}_{H} \to \alpha \geq \alpha) \wedge (v^{\emptyset}_{C_h} \to \alpha \geq -b) \\ \wedge (v^{\emptyset}_{C_h} \to \alpha \geq \alpha) \\ \wedge (v^{\emptyset}_{D} \to \alpha \geq -d_A + \alpha) \\ \wedge (v^{\emptyset}_{D} \to \alpha \geq a - f + \alpha) \end{aligned}\right\} \text{Subgroup } \{B\}.$$

Since the game has only two players, the two singleton sets of players are the only strict subgroups of players. If we consider the honest history (C_h, S), we set the action variables $v^{\emptyset}_{C_h}, v^{(C_h)}_{S}, \neg v^{\emptyset}_{H}$ and $\neg v^{(C_h)}_{I}$. The resulting formula is satisfied for all possible values of a, b, α and ε that satisfy the initial conditions. Hence, the Simplified Closing Game is collusion resilient.

Example 3.8. We also illustrate how our work disproves collusion resilience using the Simplified Routing Game with the honest history $(S_H, L, L, L, L, U, U, U, U)$. For the subgroup $\{P_1, P_3\}$ and the terminal history $(S_H, L, L, L, L, U, S_{SP_1}, U)$, we get the following implication as an instance of ϕ_{cr}:

$$v^{\emptyset}_{S_H} \wedge v^{(S_H)}_{L} \wedge v^{(S_H,L,L)}_{L} \wedge v^{(S_H,L,L,L,L)}_{U} \to 2f \geq 3f - \varepsilon$$

All action variables are set to `true` as they are part of the honest history. As $f > \varepsilon > 0$, the formula is not satisfiable. The Simplified Routing Game is thus not collusion resilient as players P_1 and P_3 can collude profitably.

3.4.4 Practicality

In practical joint strategies, no player has an incentive to deviate in any subgame (γ_{pr}). Thus, we need to inspect deviations from a joint strategy in a subgame starting from some history h, so we write $\mathscr{H}^S_{|h,t}$ to denote $\mathscr{H}^S_t$ in the subgame h.

As already presented in [RAKM23], a practical strategy can be constructed iteratively bottom-up: at every internal node, assuming we have a practical strategy (and thus utility) for its subgames, we can choose the action that yields the best utility for the current player. Using this idea, we formalize practicality as follows:

$$\bigwedge_{h \in \mathscr{H} \setminus \mathscr{T}} \bigwedge_{t,r \in \mathscr{T}_{|h}} \left[\bigwedge_{(h',a) \in \mathscr{H}^N_{|h,t},\, h' \neq \emptyset} v_a^{(h,h')} \wedge \bigwedge_{(\tilde{h},c) \in \mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})} \right] \rightarrow u_{P(h)}(h,r) \geq u_{P(h)}(h,t). \tag{ϕ_{pr}}$$

The formula first quantifies (as a conjunction) over all subgames, represented by non-terminal histories h, and then over terminal histories in the subgames starting at h. We read the implication as follows: the terminal history r is the one that the practical strategy yields, as on the left-hand side of the implication all of r's actions are asserted. On the right-hand side of the implication it is required that the utility at r of the current player (at history h) is better than the utilities from the practical strategies of other children (note that for terminal history t we do not require the first action to be asserted, but only the actions in the child subgame).

Example 3.9. We analyze practicality of the Simplified Closing Game. For the subgame starting at history (C_h), we obtain:

$$\underbrace{(v_I^{(C_h)} \rightarrow -b \geq \alpha)}_{\text{With } t=(S) \text{ and } r=(I).} \wedge \underbrace{(v_S^{(C_h)} \rightarrow \alpha \geq -b)}_{\text{With } t=(I) \text{ and } r=(S).} \tag{3.2}$$

For the honest history (C_h, S), this is satisfiable with $v_S^{(C_h)}$ and $\neg v_I^{(C_h)}$. If (C_h, I) were the honest history, there would be no strategy, as in the subgame starting at (C_h) we should have $-b \geq \alpha$, which contradicts initial conditions.

3.4.5 Equivalence to Game-Theoretic Definitions

The first-order encoding of the game-theoretic security properties established in Sections 3.4.1 to 3.4.4 is correct. That means the SMT formulas Equations (ϕ_{wi}) to (ϕ_{pr}) are equivalent to their game-theoretic versions Equations (γ_{wi}) to (γ_{pr}). To prove this claim, we need the following lemmas.

Lemma 3.2. *Let* I *be a model of* ϕ_{strat} *and* $h \in \mathscr{H} \setminus \mathscr{T}$*. Then, there exists exactly one* $r \in \mathscr{T}_{|h}$ *such that*

$$\mathsf{I}\left(\bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})}\right) = \texttt{true}.$$

Proof. Let I be such that $\mathsf{I}(\phi_{\text{strat}}) = \texttt{true}$ and $\sigma := f(\mathsf{I}) \in \mathscr{S}$ as in Lemma 3.1. Let us now fix an arbitrary $h \in \mathscr{H} \setminus \mathscr{T}$ and consider the restricted strategy $\sigma_{|h}$ after history h and its generated history $H(\sigma_{|h}) \in \mathscr{T}_{|h}$. We show that $r = H(\sigma_{|h})$. Let $(\tilde{h}, c) \in \mathscr{H}^N_{|h,H(\sigma_{|h})}$. This is equivalent to $\sigma(h, \tilde{h}) = c$. By definition of σ, this implies $\mathsf{I}(v_c^{(h,\tilde{h})}) = \texttt{true}$. Therefore, $r = H(\sigma_{|h})$. Assume there exists a different r' that makes the conjunction true. At some history t, the next actions c and c' of r and r' have to differ for the first time. Then, $\mathsf{I}(v_{c'}^t) = \mathsf{I}(v_c^t) = \texttt{true}$. By definition of σ, this implies $\sigma(h,t) = c'$ and $\sigma(h,t) = c$, which contradicts $\sigma \in \mathscr{S}$. □

Lemma 3.3. *Under the assumption* (ϕ_{strat})*, the formula* (ϕ_{pr}) *is equivalent to*

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}} \bigwedge_{p\in N} \bigwedge_{t,r\in\mathscr{T}_{|h}} \left[\bigwedge_{(h',a)\in\mathscr{H}^{N\setminus\{p\}}_{|h,t}} v_a^{(h,h')} \wedge \bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})}\right] \rightarrow u_p(h,r) \geq u_p(h,t). \tag{3.3}$$

Proof. We first prove (3.3) $\implies$ (ϕ_{pr}). Let us fix $h \in \mathscr{H} \setminus \mathscr{T}$, $r, t \in \mathscr{T}_{|h}$ and assume (3.3) and LHS of (ϕ_{pr}). Consider the LHS of (3.3) for $p = P(h)$: since $(\emptyset, a) \notin \mathscr{H}^{N\setminus P(h)}_{|h,t}$, the LHS of (3.3) for $p = P(h)$ holds and therefore its RHS does too (for $p = P(h)$), which is the same as (ϕ_{pr})'s RHS. Hence, the implication is satisfied. To show $(\phi_{pr}) \implies$ (3.3), we prove $(\phi_{pr}) \implies (N'_{pr})$ and $(N'_{pr}) \implies$ (3.3) instead by the following two lemmas, where (N'_{pr}) is as follows:

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}} \bigwedge_{t,r\in\mathscr{T}_{|h}} \left[\bigwedge_{(h',a)\in\mathscr{H}^{N\setminus P(h)}_{|h,t}} v_a^{(h,h')} \wedge \bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})}\right] \rightarrow u_{P(h)}(h,r) \geq u_{P(h)}(h,t). \tag{N'_{pr}}$$

□

Lemma 3.4. *Formula* (ϕ_{pr}) *implies formula* (N'_{pr})*.*

Proof. We transcribe (N'_{pr}) to an equivalent form as follows:

$$\bigwedge_{t\in\mathscr{T}} \bigwedge_{h\in\mathscr{H}_t^N\setminus\{t\}} \bigwedge_{r\in\mathscr{T}_{|h}} \left[\bigwedge_{(h',a)\in\mathscr{H}^{N\setminus P(h)}_{|h,t'}} v_a^{(h,h')} \wedge \bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})} \right] \qquad (N''_{pr})$$
$$\rightarrow u_{P(h)}(h,r) \geq u_{P(h)}(h,t'),$$

where $t=(h,t')$ and we prove (ϕ_{pr}) implies (N''_{pr}). Let I be a model of (ϕ_{pr}) and (ϕ_{strat}), we show I also models (N''_{pr}). Let $t=(a_1,\ldots,a_n)\in\mathscr{T}$. We prove the formula (N''_{pr}) for all $h\in\mathscr{H}_t^N\setminus\{t\}$ by induction on the number of choices $P(h)$ makes in t, so by induction on the structure of t.

Base cases: Let $h\in\mathscr{H}_t^N\setminus\{t\}$ be such that $t=(h,t')$ and player $P(h)$ makes no choices in t'. For example, $h=(a_1,\ldots,a_{n-1})$ and $t'=(a_n)$ is such a case, as t' is a terminal history after h and there are no choices available after a_n. In fact, there is precisely one base case for every player p making a choice in t. Assume the last choice p makes is a_j, then our base case is $h=(a_1,...,a_{j-1})$, $t'=(a_j,...,a_n)$. For such h, it holds that $\mathscr{H}^{N\setminus P(h)}_{|h,t}=\mathscr{H}^N_{|h,t}\setminus\{(\emptyset,a)\}$, since obviously $P(h)$'s only turn is after h and therefore the formulas are identical.

Inductive cases: Assume for $h_n\in\mathscr{H}_t^N\setminus\{t\}$ such that $t=(h_n,t_n)$ and $P(h_n)$ makes n choices in t_n, that the implication in (N''_{pr}) holds. Consider $h\in\mathscr{H}_t^N\setminus\{t\}$ and $r\in\mathscr{T}_{|h}$ such that $t=(h,t')$, the player $P(h)$ makes $n+1$ choices in t' and the LHS of the implication in (N''_{pr}) holds. According to Lemma 3.2, this means that $H(\sigma_{|h})=r$, where $\sigma=f(I)$ is the strategy generated by I. Let now h_p be the shortest non-empty subhistory of t' with $P(h)=P(h,h_p)$, i.e. $t=(h,h_p,t_n)$.

Then, for $(h,h_p)\in\mathscr{H}_t^N\setminus\{t\}$ and $r_n:=H(\sigma_{|(h,h_p)})\in\mathscr{T}_{|(h,h_p)}$, player $P(h)$ makes n choices in t_n. By construction of t_n as subsequence of t', and r_n, the LHS in (N''_{pr}) is true. Applying our hypothesis, we get that

$$u_{P(h,h_p)}(h,h_p,r_n)\geq u_{P(h,h_p)}(h,h_p,t_n).$$

Together with $P(h,h_p)=P(h)$ and $t=(h_p,t_n)$ this yields

$$u_{P(h)}(h,h_p,r_n)\geq u_{P(h)}(t).$$

Finally, we show $u_{P(h)}(h,r)\geq u_{P(h)}(h,h_p,r_n)$, by considering (ϕ_{pr}) with $h,r,(h_p,r_n)$. As $r=H(\sigma_{|h})$, we know

$$\bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})}.$$

We also know

$$\bigwedge_{(h',a)\in\mathscr{H}^N_{|h,(h_p,r_n)},h'\neq\emptyset} v_a^{(h,h')}$$

because the actions of h_p also occur in the conjunction

$$\bigwedge_{(h',a)\in\mathscr{H}_{|h,(h_p,r_n)}^{N\setminus P(h)}} v_c^{(h,\tilde{h})}$$

since $P(h)$ is never the player and the actions of r_n have to hold because $r_n = H(\sigma_{|(h,h_p)})$ and hence the corresponding action variables have to be true by definition of σ. Therefore, the RHS of (ϕ_{pr}) also has to hold, which yields $u_{P(h)}(h,r) \geq u_{P(h)}(h,h_p,r_n)$. Putting the two inequalities together, we reach $u_{P(h)}(h,r) \geq u_{P(h)}(h,t)$. This concludes the induction. □

Lemma 3.5. *Formula* (N'_{pr}) *implies formula* (3.3).

Proof. Let I be a model of (ϕ_{strat}) and (N'_{pr}), we show I is also a model of (3.3). We fix $h \in \mathscr{H} \setminus \mathscr{T}$, $r,t \in \mathscr{T}_{|h}$, $p \in N$ and assume I models the LHS of (3.3) for these h,r,t,p. For $p = P(h)$, trivially also LHS of (N'_{pr}) and thus RHS of (N'_{pr}) and RHS of (3.3) hold. For $p \neq P(h)$, we know $H(\sigma_{|h}) = r$, i.e. the resulting history $\sigma = f(I)$ of the strategy modeled by I restricted to h has to be r, by applying Lemma 3.2. Let now a_i be the first action in r after which it is player p's turn, i.e. $r = (a_1,...,a_i,r_p)$ and $P(h,a_1,..,a_i) = p$. If there is no such action a_i, then $\mathscr{H}_{|h,t}^{N\setminus\{p\}} = \mathscr{H}_{|h,t}^{N}$ and since LHS of (3.3) and (ϕ_{strat}) hold, $r = t$, so the RHS of (3.3) trivially holds. Since

$$\bigwedge_{(h',a)\in\mathscr{H}_{|h,t}^{N\setminus\{p\}}} v_a^{(h,h')},$$

we have $\sigma(h,h') = a$ for h' after which it is not p's turn. But from $H(\sigma_{|h}) = r$ it follows that $t = (a_1,..,a_i,t_p)$ for some t_p. That means r and t start with the same sequence of actions $a_1,..,a_i$, after which it is p's turn.

Now consider (N'_{pr}) with $(h,a_1,..a_i)$, r_p and t_p: The LHS is implied by the LHS of (3.3) with h,r,t, as the considered action variables are a superset of the required ones in (N'_{pr}). Hence, the RHS of (N'_{pr}) holds:

$$u_{P(h,a_1,..,a_i)}(h,a_1,..a_i,r_p) \geq u_{P(h,a_1,..,a_i)}(h,a_1,..,a_i,t_p)\ .$$

Further, $P(h,a_1,..,a_i) = p$, $(a_1,..,a_i,r_p) = r$ and $(a_1,..,a_i,t_p) = t$. Therefore, the RHS of (3.3) holds, which concludes the implication proof. □

We can now state a proof of correctness of the first-order formulas for the security properties. This means that every model of the first-order formulas provides a joint strategy that satisfies the game-theoretic security property.

Theorem 3.1 (Strategy Equivalence)**.** *Let the honest history of an EFG* $\Gamma = (N,G)$ *be* $h^* = (a_1,...,a_n)$*. For every choice of actions* σ *and every Boolean interpretation* I *of the*

action variables $(v_a^h)_{a\in A(h)}^{h\in\mathscr{H}\setminus\mathscr{T}}$ *holds: If* $\sigma(h) = a \iff \mathsf{I}(v_a^h) = \texttt{true}$*, then the following formulas are equivalent:*

$$\sigma \in \mathscr{S} \wedge H(\sigma) = h^* \wedge\ \gamma_{sp}(\sigma) \iff \mathsf{I}(\phi_{strat} \wedge \phi_{hist} \wedge \phi_{sp}) = \texttt{true}\,, \tag{3.4}$$

where $sp \in \{wi, weri, cr, pr\}$*.*

Proof. Let us fix an interpretation I and such a choice of actions σ, with $\sigma(h) = a \iff \mathsf{I}(v_a^h) = \texttt{true}$. We need to show that σ is a well-defined strategy.
Step 1: $\sigma \in \mathscr{S} \iff \mathsf{I}(\phi_{\text{strat}}) = \texttt{true}$. From Lemma 3.1 we know for a $\sigma \in \mathscr{S}$, $f^{-1}(\sigma)$ is a model of ϕ_{strat} and by definition of f, we know $f^{-1}(\sigma) = \mathsf{I}$. The same argument works vice versa since f is a bijection.
Step 2: $H(\sigma) = h^* \iff \mathsf{I}(\phi_{\text{hist}}) = \texttt{true}$. Since $h^* = (a_1, .., a_n)$, we know $\sigma(a_1, .., a_i) = a_{i+1}$ for $i \in \{1, 2, \ldots, n-1\}$. This is equivalent to $\mathsf{I}(v_{a_{i+1}}^{(a_1,..,a_i)}) = \texttt{true}$ for $i \in \{1, 2, \ldots, n-1\}$, which is equivalent to $\mathsf{I}(\phi_{\text{hist}}) = \texttt{true}$.
Step 3: Assuming $\sigma \in \mathscr{S}$ and $H(\sigma) = h^*$, then $\gamma_{\text{sp}}(\sigma) \iff \mathsf{I}(\phi_{\text{sp}}) = \texttt{true}$, $\text{sp} \in \{\text{wi}, \text{weri}, \text{cr}, \text{pr}\}$. We consider each security property separately.

Case $sp = wi$: We have to show that

$$\mathsf{I}\left(\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to u_p(t) \geq 0\right) = \texttt{true}$$

is equivalent to

$$\forall p \in N.\ \forall \tau \in \mathscr{S}.\ u_p(\tau[\sigma_p/\tau_p]) \geq 0.$$

By the rules of first-order logic, we reach the following equivalences. Note that the truth values of $u_p(t) \geq s$ and $u_p(\tau) \geq s$ for some expression s are independent of our interpretation I and our choice of actions σ. We therefore consider them Boolean values that depend on t or τ respectively.

$$\begin{aligned}
&\mathsf{I}\left(\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to u_p(t) \geq 0\right) = \texttt{true}\\
\iff &\forall p \in N.\ \forall t \in \mathscr{T}.\ \mathsf{I}\left(\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \to u_p(t) \geq 0\right) = \texttt{true}\\
\iff &\forall p \in N.\ \forall t \in \mathscr{T}.\ \left(\forall (h,a) \in \mathscr{H}_t^p.\ \mathsf{I}(v_a^h) = \texttt{true}\right) \to u_p(t) \geq 0\\
\iff &\forall p \in N.\ \forall t \in \mathscr{T}.\ (\forall (h,a) \in \mathscr{H}_t^p.\ \sigma(h) = a) \to u_p(t) \geq 0
\end{aligned}$$

The truth value of a formula does not change by adding redundant comparisons. We can therefore consider all strategies $\tau \in \mathscr{S}$ and reason about their terminal histories

$H(\tau) \in \mathscr{T}$ to continue the above equivalent transformation:

$$\begin{aligned}
&\iff \forall p \in N.\, \forall \tau \in \mathscr{S}.\, \left(\forall (h,a) \in \mathscr{H}^p_{H(\tau)}.\, \sigma(h) = a\right) \to u_p(H(\tau)) \geq 0 \\
&\overset{*}{\iff} \forall p \in N.\, \forall \tau \in \mathscr{S}.\, \tau_p = \sigma_p \to u_p(\tau) \geq 0 \\
&\iff \forall p \in N.\, \forall \tau \in \mathscr{S}.\, u_p(\tau[\sigma_p/\tau_p]) \geq 0\ .
\end{aligned}$$

The equivalence (*) holds because a strategy τ which satisfies $\forall (h,a) \in \mathscr{H}^p_{H(\tau)} \sigma(h) = a$ is exactly a strategy that coincides with σ along the resulting history $H(\tau)$ on every action a that player p takes. Since only the resulting history $H(\tau)$ matters for the utility function u, we can instead just ask for τ and σ to coincide on *every* action a player p chooses. Formally, $\tau_p = \sigma_p$. The last equivalence, finally, is only a syntactic transformation, as $\tau[\sigma_p/\tau_p]$ are precisely the strategies that coincide with σ along p's choices.

Case sp = weri: The weak immunity proof works for weaker immunity, too. We only have to replace u_p by $\mathsf{real}(u_p)$. We can do this, as the truth value of $u_p(t) \geq s$ (resp. $u_p(\tau) \geq s$) for an expression s is independent of our interpretation I and our choice of actions σ.

Case sp = cr: We show that

$$\forall S \subset N\ \forall \tau \in \mathscr{S}.\, \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\sigma[\tau_S/\sigma_S])$$

is equivalent to

$$\mathsf{I}\left(\bigwedge_{S \subset N} \bigwedge_{t \in \mathscr{T}} \left[\bigwedge_{(h,a) \in \mathscr{H}_t^{N \setminus S}} v_a^h\right] \to \sum_{p \in S} u_p^{\mathscr{T}}(h^*) \geq \sum_{p \in S} u_p(t)\right) = \texttt{true}.$$

By performing the analogous equivalence transformations as for the weak immunity proof, we get

$$\mathsf{I}\left(\bigwedge_{S \subset N} \bigwedge_{t \in \mathscr{T}} \left[\bigwedge_{(h,a) \in \mathscr{H}_t^{N \setminus S}} v_a^h\right] \to \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(t)\right) = \texttt{true}$$

equivalent to

$$\forall S \subset N\ \forall \tau \in \mathscr{S}.\, \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(\tau[\sigma_{N \setminus S}/\tau_{N \setminus S}])\ .$$

Since $H(\sigma) = h^*$, it follows that $u_p(h^*) = u_p(\sigma)$. Finally, the fact that $\tau[\sigma_{N \setminus S}/\tau_{N \setminus S}] = \sigma[\tau_S/\sigma_S]$ (for $p \in S$ it is τ_p and for $p \in N \setminus S$ it is σ_p) concludes the proof of equivalence.

Case $sp = pr$: We know that ϕ_{pr} and (3.3) are equivalent under our assumptions ϕ_{strat} from Lemma 3.3. It therefore suffices to prove $\mathsf{I}((3.3)) = \texttt{true}$ iff $\gamma_{\text{pr}}(\sigma)$, that is, proving

$$\forall h \in \mathscr{H}\, \forall p \in N\, \forall \tau \in \mathscr{S}_{|h}.\ u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h}[\tau_p/\sigma_{|h,p}])$$

equivalent to

$$\mathsf{I}\left(\begin{array}{c} \bigwedge_{h \in \mathscr{H} \setminus \mathscr{T}} \bigwedge_{p \in N} \bigwedge_{t,r \in \mathscr{T}_{|h}} \\ \left[\bigwedge_{(h',a) \in \mathscr{H}_{|h,t}^{N \setminus \{p\}}} v_a^{(h,h')} \ \wedge \bigwedge_{(\tilde{h},c) \in \mathscr{H}_{|h,r}^{N}} v_c^{(h,\tilde{h})}\right] \\ \rightarrow u_p(h,r) \geq u_p(h,t) \end{array}\right) = \texttt{true}.$$

Note that in $\gamma_{\text{pr}}(\sigma)$ we can equivalently only consider $h \in \mathscr{H} \setminus \mathscr{T}$ in $\gamma_{\text{pr}}(\sigma)$, since for $h \in \mathscr{T}$ the strategy is empty and thus the inequality is trivially satisfied. We can now proceed with equivalently rewriting (3.3) analogous to the other security properties. We rewrite $\mathsf{I}(v_c^{(h,\tilde{h})}) = \texttt{true}$ as $\sigma(h,\tilde{h}) = c$ for all $(\tilde{h}, c)$ such that $r(\tilde{h}) = c$, which is equivalent to $H(\sigma_{|h}) = r$. We obtain that

$$\begin{aligned} \forall h \in \mathscr{H} \setminus \mathscr{T}\, \forall p \in N\, \forall t, r \in \mathscr{T}_{|h}.& \\ &\left(\forall (h',a) \in \mathscr{H}_{|h,t}^{N \setminus \{p\}}.\, \sigma_{|h}(h') = a \wedge H(\sigma_{|h}) = r\right) \\ &\rightarrow u_p(h,r) \geq u_p(h,t) \end{aligned}$$

is equivalent to

$$\begin{aligned} \forall h \in \mathscr{H} \setminus \mathscr{T}\, \forall p \in N\, \forall t \in \mathscr{T}^h.& \\ &\left(\forall (h',a) \in \mathscr{H}_{|h,t}^{N \setminus \{p\}}.\, \sigma_{|h}(h') = a\right) \\ &\rightarrow u_p(h, H(\sigma_{|h}))) \geq u_p(h,t)\ . \end{aligned}$$

This equivalence holds because for every $h \in \mathscr{H} \setminus \mathscr{T}$, there exists precisely one r that satisfies $H(\sigma_{|h}) = r$, namely $H(\sigma_{|h})$ itself. Therefore, replacing r by $H(\sigma_{|h})$ in the utility function u_p does not change the truth value of the formula. From there, we can again proceed as we did for the other security properties and reach

$$\forall h \in \mathscr{H} \setminus \mathscr{T}\, \forall p \in N\, \forall \tau \in \mathscr{S}_{|h}.\, u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h}[\tau_p/\sigma_{|h,p}]),$$

which is precisely (γ_{pr}).

Finally, we put the three steps together, which yields $\sigma \in \mathscr{S} \wedge H(\sigma) = h^* \wedge \gamma_{\text{sp}}(\sigma)$ iff $\sigma \in \mathscr{S} \wedge H(\sigma) = h^* \wedge \mathsf{I}(\phi_{\text{sp}}) = \texttt{true}$ iff $\mathsf{I}(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}) = \texttt{true}$. □

3.5 Automated Reasoning of Game-Theoretic Security

The first-order formulas of Section 3.4, combined with Lemma 3.1, provide us with the theoretical foundations for automating security analysis of blockchain protocols

presented as EFG trees. We now present our algorithmic advancement on top of satisfiability checking modulo linear real arithmetic (Algorithm 3.1), allowing us to (dis)prove formulas from Section 3.4 and hence provide game-theoretic security guarantees. Further, our work yields natural extensions for generating concrete counterexamples whenever security properties are violated (Section 3.5.2) and infers preconditions to enforce security (Algorithm 3.3).

For proving the security formulas of Section 3.4, we focus on automating reasoning about a tuple

$$\Pi = (\Gamma, \mathscr{O}, \mathit{inf}, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}}), \text{ where}$$

- Γ is an EFG modeling the protocol of interest;
- $\mathscr{O} \subseteq \mathscr{T}$ is a set of honest histories representing expected behavior;
- inf is a set of infinitesimals occurring in the players' utilities;
- C is the set of initial constraints on variables occurring in player utilities;
- C_{wi}, C_{weri}, C_{cr}, and C_{pr} are sets of constraints on variables to hold when checking formulas of Section 3.4, namely weak immunity (C_{wi}), weaker immunity (C_{weri}), collusion resilience (C_{cr}) and practicality (C_{pr}), respectively.

Note that the sets C, C_{wi}, C_{weri}, C_{cr} and C_{pr} may possibly be empty. For every honest history in $\mathscr{O}$, our work constructs the respective first-order security formula of Section 3.4 and uses SMT solving for establishing satisfiability of the respective formula: if a model is found, we construct a joint strategy as described in Lemma 3.1. Since security properties must hold for all possible values of the utility variables that adhere to the initial conditions C, we universally quantify over all variables and add an implication to account for preconditions. We hence translate game-theoretic analysis from Section 3.4 into the satisfiability solving of

$$\forall \vec{x}. \left[\bigwedge_{c \in C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow (\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]) \right], \tag{3.5}$$

where $\vec{x} = (x_1, x_2, \ldots, x_\ell)$ are all variables appearing in utilities of players and $C_{\text{sp}} \in \{C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}}\}$. By Lemma 3.1, a model $\mathsf{I} \in \mathcal{I}$ of (3.5) is a joint strategy satisfying the respective security property of Section 3.4.

We next detail our solution towards solving (3.5), yielding our automated reasoning approach to prove game-theoretic security in Algorithm 3.1.

3.5.1 Security Reasoning with Case Splitting

We note that formula (3.5) is too restrictive, as next illustrated.

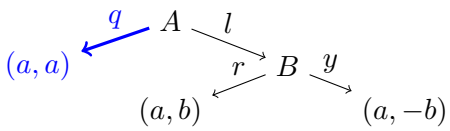


Figure 3.3: EFG $\text{Splits}_{\text{wi}}$ Necessitating Case Splits During Reasoning. We Assume $a > 0$.

Example 3.10 ($\text{Splits}_{\text{wi}}$)**.** Consider the EFG of Figure 3.3 with $N = \{A, B\}$ and honest history (q), where $a > 0$. We aim to find a weak immune strategy for this game. Clearly, A must take action q, but if A deviates, B must have non-negative utility. The action of B depends on b: if $b > 0$, B should choose r; if $b < 0$, y should be chosen; and otherwise, either choice is possible. The game is therefore weak immune for history (q), but requires different strategies for different cases.

Example 3.10 shows that during security analysis, we may need to consider several different orderings of linear terms within utilities. Such *case splits* turn out to be also necessary for real-world protocols, such as the Closing Game [RAKM23]. In order to account for possible case splits, we modify (3.5) and introduce preconditions to order terms. To this end, we compute the set T_u of linear terms appearing in the constructed formulas; for example, for collusion resilience we have:

$$T_u = \left\{ \sum_{p \in S} u_p(t) \mid S \subset N, t \in \mathscr{T} \right\}. \tag{3.6}$$

We then consider all consistent *total orders* $\preceq$ over T_u. As we must find models for all such orders $\preceq$, we reduce solving (3.5) to solving

$$\boxed{\begin{array}{l} \forall\ (\preceq, T_u) \text{ total order. } \exists\ \mathsf{I} \in \mathcal{I}. \\ \mathsf{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow (\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]) \right) = \top, \end{array}} \tag{3.7}$$

where quantification over total orders $\preceq$ happens outside SMT solving (see Remark 5) and $\preceq$ is interpreted as the set of ordering constraints over elements of T_u, representing the total order it yields. This is, in fact, equivalent to the desired all-quantified formula $\forall \vec{x}\ \exists \mathsf{I}$, as shown next.

Theorem 3.2. *For an EFG $\Gamma = (N, G)$, the formula* (3.7) *for $sp \in \{wi, weri, cr, pr\}$ is equivalent to the eagerly all-quantified formula:*

$$\forall \vec{x}.\ \exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}\left(\bigwedge_{c \in C \cup C_{sp}} c[\vec{x}] \rightarrow \phi_{strat} \wedge \phi_{hist} \wedge \phi_{sp}[\vec{x}] \right) = \texttt{true}, \qquad (\forall \vec{x}\ \exists \mathsf{I})$$

where $\vec{x}$ are the variables appearing in the utilities of the game tree.

Proof. *Case* $(\forall \vec{x}\ \exists \mathsf{I}) \Leftarrow (3.7)$: Fix $\vec{x} = (x_1, \ldots, x_\ell) \in \mathbb{R}^\ell$. This defines precisely one total ordering on T_u, call it $\preceq$. We use (3.7) with $\preceq$ to obtain a model I such that

$$\mathsf{I}\left(\forall \vec{z}. \bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{z}] \to \phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{z}]\right) = \texttt{true}. \tag{3.8}$$

We take the same model (the same assignment of the action variables) as a candidate for our model. We need to prove

$$\mathsf{I}\left(\bigwedge_{c \in C \cup C_{\text{sp}}} c[\vec{x}] \to \phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]\right) = \texttt{true}.$$

We set $\vec{z} = \vec{x}$ in the formula (3.8). Now we know that $\bigwedge_{c \in \preceq} c[\vec{x}]$ holds by construction, so the formula simplifies to the desired one.

Case $(\forall \vec{x}\ \exists \mathsf{I}) \Rightarrow (3.7)$: Suppose $(\forall \vec{x}\ \exists \mathsf{I})$ holds. Let $\preceq$ be a total order on T_u. If[1]

$$\left(\bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c\right) = \texttt{false},$$

then the formula

$$\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{x}] \to \phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]$$

is a tautology (as the left-hand side of the implication is false), so every assignment of the action variables is a suitable model for $\mathsf{I} \in \mathcal{I}$. So, suppose $\vec{r} = (r_1, \ldots, r_\ell) \in \mathbb{R}^\ell$ is such that

$$\bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{r}] = \texttt{true}. \tag{3.9}$$

We use $(\forall \vec{x}\ \exists \mathsf{I})$ with $\vec{r}$ to obtain a model $\mathsf{I} \in \mathcal{I}$ and we claim I is also a model for (3.7) with the ordering $\preceq$. From (3.9) we can conclude that

$$\mathsf{I}(\phi_{\text{strat}}) \wedge \mathsf{I}(\phi_{\text{hist}}) \wedge \mathsf{I}(\phi_{\text{sp}}[\vec{r}]) = \texttt{true}. \tag{3.10}$$

Now let $\vec{z} = (z_1, z_2, \ldots, z_\ell) \in \mathbb{R}^\ell$ be such that $\bigwedge_{c \in \preceq \cup C \cup C_{\text{sp}}} c[\vec{z}]$ holds. We need to prove that

$$\mathsf{I}(\phi_{\text{strat}}) \wedge \mathsf{I}(\phi_{\text{hist}}) \wedge \mathsf{I}(\phi_{\text{sp}}[\vec{z}]) = \texttt{true}.$$

From (3.10) we know that $\mathsf{I}(\phi_{\text{strat}}) \wedge \mathsf{I}(\phi_{\text{hist}}) = \texttt{true}$, so we just need to prove that $\mathsf{I}(\phi_{\text{sp}}[\vec{z}]) = \texttt{true}$.

We make the following observation: since $\vec{r}$ and $\vec{z}$ both satisfy the total order $\preceq$ on $T_u[\vec{r}]$ (respectively $T_u[\vec{z}]$), for every $e, e' \in T_u[\vec{x}]$ we have that

$$e(\vec{r}) \preceq e'(\vec{r}) \quad \text{iff} \quad e(\vec{z}) \preceq e'(\vec{z}). \tag{3.11}$$

[1] This is decidable because it is just linear arithmetic.

To prove $\mathsf{I}(\phi_{\text{sp}}[\vec{z}]) = \texttt{true}$, we consider security properties separately.

Case sp = wi: The equation $\mathsf{I}(\phi_{\text{wi}}[\vec{z}]) = \texttt{true}$ corresponds to

$$\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p}\mathsf{I}(v_a^h)\right] \to u_p(t)[\vec{z}] \geq 0.$$

Suppose $p \in N$ and $t \in \mathscr{T}$ such that

$$\left(\bigwedge_{(h,a)\in\mathscr{H}_t^p}\mathsf{I}(v_a^h)\right) = \texttt{true}.$$

From (3.10) we know that $u_p(t)[\vec{r}] \geq 0$ and since $u_p(t)[\vec{x}] \in T_u[\vec{x}]$ and $0 \in T_u[\vec{x}]$, we can from (3.11) conclude the desired inequality $u_p(t)[\vec{z}] \geq 0$.

Case sp = weri: The proof is similar to the case sp = wi.

Case sp = cr: The equation $\mathsf{I}(\phi_{\text{cr}}[\vec{z}]) = \texttt{true}$ corresponds to

$$\bigwedge_{S\subset N}\bigwedge_{t\in\mathscr{T}}\left[\bigwedge_{(h,a)\in\mathscr{H}_t^{N\setminus S}}\mathsf{I}(v_a^h)\right] \to \sum_{p\in S}u_p(h^*)[\vec{z}] \geq \sum_{p\in S}u_p(t)[\vec{z}].$$

Suppose $S \subset N$ and $t \in \mathscr{T}$ such that

$$\bigwedge_{(h,a)\in\mathscr{H}_t^{N\setminus S}}\mathsf{I}(v_a^h).$$

From (3.10) we know that

$$\sum_{p\in S}u_p(h^*)[\vec{r}] \geq \sum_{p\in S}u_p(t)[\vec{r}]$$

and since $\sum_{p\in S}u_p(h^*)[\vec{x}] \in T_u[\vec{x}]$ and $\sum_{p\in S}u_p(t)[\vec{x}] \in T_u[\vec{x}]$, we can conclude the desired inequality from (3.11)

$$\sum_{p\in S}u_p(h^*)[\vec{z}] \geq \sum_{p\in S}u_p(t)[\vec{z}].$$

Case sp = pr: We first use Lemma 3.3 with model I and thus the goal to prove becomes the following:

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}}\bigwedge_{p\in N}\bigwedge_{t,t'\in\mathscr{T}_{|h}} \left[\bigwedge_{(h',a)\in\mathscr{H}_{|h,t}^{N\setminus\{p\}}}\mathsf{I}(v_a^{(h,h')}) \wedge \bigwedge_{(\tilde{t},c)\in\mathscr{H}_{|h,t'}^{N}}\mathsf{I}(v_c^{(h,\tilde{t})})\right] \to u_p(h,t')[\vec{z}] \geq u_p(h,t)[\vec{z}].$$

Suppose $h \in \mathscr{H} \setminus \mathscr{T}$, $p \in N$ and $t, t' \in \mathscr{T}_{|h}$ are such that

$$\left[\bigwedge_{(h',a) \in \mathscr{H}^{N \setminus \{p\}}_{|h,t}} \mathsf{I}(v_a^{(h,h')}) \ \wedge \bigwedge_{(\tilde{t},c) \in \mathscr{H}^{N}_{|h,t'}} \mathsf{I}(v_c^{(h,\tilde{t})}) \right] = \texttt{true}.$$

From (3.10) we know that

$$u_p(h,t')[\vec{r}] \geq u_p(h,t)[\vec{r}]$$

and since $u_p(h,t')[\vec{x}] \in T_u[\vec{x}]$ and $u_p(h,t)[\vec{x}] \in T_u[\vec{x}]$, we can from (3.11) conclude the desired inequality

$$u_p(h,t')[\vec{z}] \geq u_p(h,t)[\vec{z}].$$

□

We conclude the following result.

Theorem 3.3 (Game-Theoretic Security)**.** *Let $\Gamma = (N, G)$ be an EFG with honest history h^*, the formula* (3.7) *for $sp \in \{wi, weri, cr, pr\}$ is equivalent to its game-theoretic analog:*

$$\forall \vec{x}.\ \forall c \in C \cup C_{sp}.\ c[\vec{x}] \rightarrow \exists\, \sigma \in \mathscr{S}.\ H(\sigma) = h^* \wedge\ \gamma_{sp}(\sigma)[\vec{x}], \tag{3.12}$$

where $\vec{x} = (x_1, \ldots, x_\ell)$ are the variables occurring in the utility terms (interpreted over reals), and C and C_{sp} are finite sets of linear preconditions on $\vec{x}$.

Proof. If we can show that for a game $\Gamma = (N, G)$ with honest history h^*, formula (3.12) is equivalent to formula $(\forall \vec{x}\ \exists \mathsf{I})$, then the theorem follows from Theorem 3.2, which shows the equivalence of $(\forall \vec{x}\ \exists \mathsf{I})$ and (3.7).

We, therefore, prove the claimed equivalence. In the following, we write $p[\vec{x}]$ for $\bigwedge_{c \in C \cup C_{\mathrm{sp}}} c[\vec{x}]$ in $(\forall \vec{x}\ \exists \mathsf{I})$. For fixed σ and I we know that $\sigma \in \mathscr{S} \wedge H(\sigma) = h^* \wedge \gamma_{\mathrm{sp}}(\sigma)[\vec{x}]$ iff $\mathsf{I}(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]) = \texttt{true}$, where $\mathsf{I}(v_a^h) = \texttt{true}$ iff $\sigma(h) = a$, from Theorem 3.1. Thus,

$$\begin{aligned} & \exists\, \sigma \in \mathscr{S}.\ H(\sigma) = h^* \wedge\ \gamma_{\mathrm{sp}}(\sigma)[\vec{x}] \\ \iff\ & \exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]) = \texttt{true}\ . \end{aligned}$$

Let us now fix arbitrary $\vec{x} = (x_1, \ldots, x_\ell) \in \mathbb{R}^\ell$.

Case 1: $p[\vec{x}] = \texttt{true}$. Then,

$$p[\vec{x}] \rightarrow \exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]) = \texttt{true}$$

is equivalent to $\exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}(\vec{x})) = \texttt{true}$, which is equivalent to $\exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}(p[\vec{x}] \rightarrow \phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]) = \texttt{true}$.

Case 2: $p[\vec{x}] = \texttt{false}$. Then, $\mathsf{I}(p[\vec{x}] \rightarrow \phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}])$ is $\texttt{true}$ for all I. Also, $p[\vec{x}] \rightarrow \exists \mathsf{I} \in \mathcal{I}.\ \mathsf{I}(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]) = \texttt{true}$ holds.

Hence, the claimed equivalence holds for all values of $\vec{x}$. □

Note that every model I of (3.7) translates to a strategy σ in (3.12) and vice versa, by letting $\sigma(h) = a$ iff $\mathsf{I}(v_a^h) = \texttt{true}$. The above theorem yields the following result.

Corollary 3.1 (Soundness and Completeness of Encoding)**.** *The encoding of the game-theoretic properties weak immunity, weaker immunity, collusion resilience, and practicality using the formulas ϕ_{strat}, ϕ_{hist} together with ϕ_{wi}, ϕ_{weri}, ϕ_{cr}, and ϕ_{pr}, respectively, including case splits* (3.12)*, is sound and complete.*

Proof. The corollary follows directly from Theorem 3.3 □

Note that our first-order formulas encoding game-theoretic security properties are expressed in the decidable theory of first-order linear real arithmetic [Col75], [JDM12]. Since the number of total orders in (3.7) is finite, our formalization is *decidable*.

Remark 5. For efficient handling of (3.7), we compute *unsatisfiable (unsat) cores* to detect which linear terms require case splitting. We introduce labels for inequalities appearing in the formula ϕ_{sp} — one of ϕ_{wi}, ϕ_{weri}, ϕ_{cr} or ϕ_{pr}. For example, if ϕ_{sp} contains $x < y$, we introduce a label $\ell_{(x,y)}$ as a new Boolean variable[2], replace the inequality with the implication $\ell_{(x,y)} \rightarrow x < y$, and add $\ell_{(x,y)}$ to our encoding. The weak immunity formula ϕ_{wi} then becomes

$$\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left(\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \rightarrow \ell_{(u_p(t),0)} \rightarrow u_p(t) \geq 0\right) \wedge \ell_{(u_p(t),0)}$$

The adjusted formula for weaker immunity, ϕ_{weri}, reads as follows:

$$\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left(\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right] \rightarrow \ell_{(\mathsf{real}(u_p(t)),0)} \rightarrow \mathsf{real}(u_p(t)) \geq 0\right) \wedge \ell_{\mathsf{real}(u_p(t)),0)}.$$

The constraint for collusion resilience, ϕ_{cr}, is rewritten as:

$$\bigwedge_{S\subset N}\bigwedge_{t\in\mathscr{T}}\left(\left[\bigwedge_{(h,a)\in\mathscr{H}_t^{N\setminus S}} v_a^h\right] \rightarrow \ell_{(\sum_{p\in S} u_p(h^*),\sum_{p\in S} u_p(t))} \rightarrow \sum_{p\in S} u_p(h^*) \geq \sum_{p\in S} u_p(t)\right) \wedge \ell_{(\sum_{p\in S} u_p(h^*),\sum_{p\in S} u_p(t))}.$$

The formula for practicality, ϕ_{pr}, is amended as follows:

$$\bigwedge_{h\in\mathscr{H}\setminus\mathscr{T}}\bigwedge_{t,r\in\mathscr{T}_{|h}}\left(\left[\bigwedge_{(h',a)\in\mathscr{H}^N_{|h,t},\, h'\neq\emptyset} v_a^{(h,h')} \wedge \bigwedge_{(\tilde{h},c)\in\mathscr{H}^N_{|h,r}} v_c^{(h,\tilde{h})}\right] \rightarrow \ell_{(u_{P(h)}(h,r),u_{P(h)}(h,t))} \rightarrow u_{P(h)}(h,r) \geq u_{P(h)}(h,t)\right) \wedge \ell_{(u_{P(h)}(h,r),u_{P(h)}(h,t))}.$$

[2]We actually use *two* labeling Boolean variables in our implementation, one each for the infinitesimal and real components of the inequality.

If unsatisfiability of the formula is established, we extract an unsat core composed of our labels and retrieve candidate linear terms for case splits. Example 3.10 yields an unsat core composed of labels $\ell_{0,-b}$ and $\ell_{b,0}$, from which we conclude that the following three cases need to be analyzed: $b > 0$, $b = 0$ and $b < 0$. Yet, even for a small number of linear terms, there might be a very large number of possible orderings $\preceq$ between terms to be considered. The number of $\preceq$ can, however, be significantly reduced by making two observations: (i) orderings $\preceq$ must be compatible with C and C_{sp}; and (ii) orderings must be consistent with real arithmetic. For example, with the set of linear terms $a, -a, 0$, we should consider $a > 0 > -a$, $-a > 0 > a$, and $a = -a = 0$, but not e.g. $a > -a > 0$ or $a = 0 > -a$.

Based on the above, Algorithm 3.1 summarizes our reasoning approach for proving game-theoretic security properties based on our first-order formalization with case splits. Given an input Π, Algorithm 3.1 establishes satisfiability of a game-theoretic property (cf. Lemma 3.1), by using an auxiliary SMT solver A to track (remaining) possible terms orderings. If at any point A reports unsatisfiability, there are no more possible orderings $\preceq$. Constraints C and C_{sp} are added immediately to A. To get an ordering, we ask A for a model — which maps real variables to numeric values — from which we infer an ordering $\preceq$ on linear terms over variables (`EvaluateModel`). For example, if the model is $a = 1, b = 2, c = 3$, we obtain the ordering on linear terms $2c - b > a + b = c > 0 > a - c$. If we find a strategy for a case split, we add a *conflict clause* to ensure we do not consider the same ordering twice.

We note that in Algorithm 3.1 the only properties considered are weak(er) immunity, collusion resilience, and practicality; these properties are enough to enforce game-theoretic security (Definition 3.7). However, Algorithm 3.1 can be applied to any game-theoretic formula about an EFG as long as this formula can be expressed as a first-order linear inequality over two computable functions f, g, with utility function u, set of players N and set of strategies $\mathscr{S}$, that is $f(u, N, \mathscr{S}) \leq g(u, N, \mathscr{S})$. Our work yields a generic way to translate such game-theoretic formulas to SMT formulas.

3.5.2 Generating Counterexamples to Security

In case there is no joint strategy fulfilling the security property ϕ_{sp} within Algorithm 3.1, our work provides automated formal analysis explaining why the conditions of ϕ_{sp} are violated and derives concrete *counterexamples* to security.

Weak(er) Immunity. For the weak(er) immunity property ϕ_{wi}, a counterexample is a harmed player p together with a partial strategy of the other players $N - p$ such that — while following the honest history — no matter how p acts, they cannot avoid receiving a real-valued negative utility. We say that a strategy for a player p follows the honest history h^* if at every node appearing in h^*, where p is making a choice, the strategy will choose the action in h^*.

Algorithm 3.1: Game-Theoretic Security Reasoning

input : an input instance $\Pi = (\Gamma, \mathscr{O}, \mathit{inf}, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}})$, an honest history $h_o \in \mathscr{O}$ and the name of a security property $\text{sp} \in \{\text{wi}, \text{weri}, \text{cr}, \text{pr}\}$

output : `true` if ϕ_{sp} is satisifiable in Π, `false` otherwise

```
1  S ← Solver()
2  A ← Solver()
3  AddConstraints (S, ComputeStrategyConstraints (Γ, h_o))
4  AddConstraints (A, C ∪ C_sp)
5  T ← ∅
6  while Solve (A) = sat do
7  |  M ← GetModel (A)
8  |  O ← {EvaluateModel (M, x, y) : (x, y) ∈ Combinations (T)}
9  |  if Check (S, sp, C ∪ C_sp ∪ O) = sat then
10 |  |  // found strategy for current ordering, add conflict clause
   |  |  AddConstraints (A, {⋁_{c∈O} ¬c})
11 |  end
12 |  else
13 |  |  T' ← ∅
14 |  |  foreach ℓ_{x,y} ∈ GetUnsatCore(S) do
15 |  |  |  T' ← T' ∪ {t : t ∈ {x, y}, t ∉ T }
16 |  |  if T' = ∅ then // no new expressions, considered every case
17 |  |  |  return false
18 |  |  end
19 |  |  T ← T ∪ T'
20 |  end
21 end
22 return true
```

Definition 3.8 (Counterexamples of Weak(er) Immunity)**.** *Let* Γ *be an EFG and* h^* *the considered honest history. A* counterexample to h^* being weak(er) immune *is a player* p *together with a partial strategy* $s_{N-p} \subseteq \tau_{N-p} \in \mathscr{S}_{N-p}$ *of the other players such that* s_{N-p} *extended by any strategy* σ_p *of player* p*, which follows the honest history* h^**, yields a terminal history* t *and it is* minimal *with that property. Further, we have*

$$\forall \sigma_p \in \mathscr{S}_p \left(\forall (h, a) \in \mathscr{H}^p_{h^*}.\ \sigma_p(h) = a\right) \rightarrow u_p(t) < 0 \tag{3.13}$$

for weak immunity (wi), and respectively

$$\forall \sigma_p \in \mathscr{S}_p \left(\forall (h, a) \in \mathscr{H}^p_{h^*}.\ \sigma_p(h) = a\right) \rightarrow \mathsf{real}(u_p(t)) < 0 \tag{3.14}$$

for weaker immunity (weri).

Minimality of the partial strategy s_{N-p} states that, if any information point $s_{N-p}(h) = a$ is removed, there exists a strategy σ_p of player p such that the tuple (σ_p, s'_{N-p}) does not yield a terminal history, where the s'_{N-p} is s_{N-p} without action a. That means, when following only actions of (σ_p, s'_{N-p}), we get stuck at an internal node of the tree.

Example 3.11. The Simplified Closing Game of Figure 3.1 with honest history (C_h, S) is not weak immune. The counterexample is the player A and the partial strategy for player B taking action I after C_h (and therefore making the utility of A negative). The partial strategy is minimal, because if we remove the choice of action I for player B, after A chooses action C_h, we get stuck at the internal node of the game (where B is making a choice), rather than ending in a leaf node, where the utilities of players are known. Note that the partial strategy says nothing about the tree in the subgame after D, as we only consider strategies where player A chooses C_h at the root node, to follow the honest history.

Collusion Resilience. A counterexample to collusion resilience ϕ_{cr} consists of a group of deviating players S and their partial strategy $s_S \in \mathscr{S}$ leading to better-than-honest joint utility for S, no matter how the other players $N - S$ react – while following the honest history.

Definition 3.9 (Counterexamples Collusion Resilience)**.** *Let Γ be an EFG and h^* the considered honest history. A* counterexample to h^* being collusion resilient (cr) *is a set of deviating players S together with their partial strategy $s_S \subseteq \tau_S \in \mathscr{S}_S$ such that s_S extended by any strategy σ_{N-S} of players $N - S$, which follows the honest history h^*, yields a terminal history t and it is minimal with that property. Further,*

$$\begin{aligned} &\forall \sigma_{N-S} \in \mathscr{S}_{N-S} \,. \\ &\left(\forall (h, a) \in \mathscr{H}_{h^*}^{N-S}.\, \sigma_{N-S}(h) = a\right) \rightarrow \sum_{p \in S} u_p(t) > \sum_{p \in S} u_p(h^*) \,. \end{aligned} \tag{3.15}$$

The minimality of s_S is similar to the minimality of the partial strategy for weak(er) immunity.

Example 3.12. In the Simplified Routing Game of Figure 3.2, the terminal history $(S_H, L, L,\ L, L, U, S_{SP_1}, U)$ results in a strictly better outcome for the subgroup $\{P_1, P_3, B\}$ than the honest history, modeling the Wormhole attack (see Section 3.2). While choosing the honest actions L and L, the other players A and P_2 are powerless in this scenario.

Practicality. Intuitively, a counterexample to practicality of h^* has to provide a reason why a rational player would not follow h^*. That is, somewhere along h^*, assume after a prefix h, there exists an action a which promises the current player $P(h)$ a strictly better

utility than h^*. However, a "promised" utility has to be one that results from a practical history t in the subgame after (h, a), otherwise it would not be an actual counterexample to the practicality of h^*. Therefore, we define a counterexample to practicality as follows.

Definition 3.10 (Counterexamples to Practicality)**.** *For an EFG Γ and honest history h^*, a* counterexample to practicality of h^* *is a prefix h of h^* together with an action $a \in A(h)$, such that for all practical terminal histories t in the subgame $\Gamma_{|(h,a)}$ it holds that $u_{P(h)}(h^*) < u_{P(h)}((h, a, t))$.*

Example 3.13. The Simplified Closing Game of Figure 3.1 with honest history (H) is not practical. The counterexample is action C_h at the root. That is, the prefix h of the honest history (H) is the empty history, and the deviating action is C_h. The only practical terminal history in the subgame Γ_{C_h} is (S), since it yields a strictly better utility for B than action I. Finally, the utility for $P(h) = A$ after the honest history (H) is $\alpha - \varepsilon$, while A's utility after (C_h, S) is α. Action C_h at the root is, therefore, a valid counterexample to the practicality of honest history (H).

Correctness of Counterexamples. The various counterexamples to security properties, as introduced above, are evidence for a violated security property, as stated below.

Theorem 3.4 (Counterexamples to Security)**.** *For an EFG Γ and an honest history h^*, there exists a counterexample to wi, weri, cr or pr of h^* according to Definition 3.8–Definition 3.10 iff h^* is not weak(er) immune, collusion resilient, or practical, respectively.*

Proof. We start with weak immunity. We show first that the existence of a counterexample implies that h^* is not weak immune.

Let $p \in N$ and $s_{N-p} \subseteq \tau_{N-p}$ be a counterexample to the weak immunity of h^* in Γ according to Definition 3.8. We first extend the partial strategy s_{N-p} to an arbitrary strategy $\tau_{N-p} \in \mathscr{S}_{N-p}$, $s_{N-p} \subseteq \tau_{N-p}$ and consider the formula

$$\forall \sigma_p \in \mathscr{S}_p \, (\forall (h, a) \in \mathscr{H}^p_{h^*}.\ \sigma_p(h) = a) \rightarrow u_p(\sigma_p, \tau_{N-p}) < 0 \, . \tag{3.16}$$

The utility remains unchanged as the extension does not impact the generated terminal history. By definition, the history h^* is not weak immune if

$$\begin{aligned} &\forall \sigma \in \mathscr{S}\ H(\sigma) = h^* \rightarrow \\ &\quad \exists p \in N\ \exists \tau_{N-p} \in \mathscr{S}_{N-p}.\ u_p(\sigma_p, \tau_{N-p}) < 0 \, . \end{aligned} \tag{3.17}$$

Let us now pick an arbitrary $\sigma \in \mathscr{S}$ with $H(\sigma) = h^*$ in (3.17). the restriction of this σ to p's single strategy σ_p satisfies the LHS of (3.16). Therefore, $u_p(\sigma_p, \tau_{N-p}) < 0$, which concludes this direction of the proof.

For the other implication, that whenever h^* is not weak immune there exists a counterexample according to Definition 3.8, we consider the SMT formula for weak immunity,

which was proven equivalent (modulo the labels $\ell_{p,t}$) in Theorem 3.3:

$$\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \bigwedge_{p \in N} \bigwedge_{t \in \mathscr{T}} (\ell_{p,t} \rightarrow ((\bigwedge_{(h,a) \in \mathscr{H}_t^p} v_a^h) \rightarrow u_p(t) \geq 0)) \wedge \ell_{p,t}. \tag{3.18}$$

We assume h^* is not weak immune, therefore, the above formula is unsat. By construction of the formula, we know that we can satisfy $\phi_{\text{strat}} \wedge \phi_{\text{hist}}$. We therefore consider a minimal unsat core of the labels $\ell_{p,t}$. Since within each player p the relevant action variables v_a^h are independent of the other players (we only consider actions made by p in $\mathscr{H}_t^p$), a minimal unsat core only contains labels of one player p. Thus, a minimal unsat core L consists of finitely many labels $\ell_{p,t}$, representing terminal histories t that yield a negative utility for player p, if p chose actions according to these histories. As L is a minimal unsat core, at least one of those terminal histories had to be chosen by p to find a strategy, but none of them can be. If we now remove p's actions a from the terminal histories t of the unsat core, and collect the remaining choices of actions, we receive a partial strategy s_{N-p} of the other players $N - p$. The fact that L is a minimal core ensures that there is at most one action at each internal node. It also makes the constructed partial strategy s_{N-p} minimal in the number of asserted actions. Finally, by adding the actions of a single strategy σ_p to s_{N-p}, we have a path of actions from root to leaf, since the unsat core has to contain a problem (which is label, which is terminal history) for every possible behavior of p.

The proofs for the correctness of weaker immunity and collusion resilience are similar.

To show the correctness of the practicality counterexamples according to Definition 3.10, we start with implication $\Rightarrow$. We prove the existence of a practicality counterexample implies the game Γ with honest history h^* is not practical, by assuming the negation (exists counterexample and Γ is practical) and leading it to a contradiction:

First, we show the following *observation* for any history $h' \in \mathscr{H}$ and practical strategy $\sigma \in \mathscr{S}$, also $\sigma_{|h'} \in \mathscr{S}_{|h'}$ is practical. The strategy $\sigma_{|h'} \in \mathscr{S}_{|h'}$ is practical, iff

$$\forall h \in \mathscr{H}_{|h'}\ \forall p \in N\ \forall \tau \in \mathscr{S}_{|(h',h)}. \\ u_{|(h',h),p}(\sigma_{|(h',h)}) \geq u_{|(h',h),p}(\sigma_{|(h',h)}[\tau_p / \sigma_{|(h',h),p}]) \ .$$

By fixing h, p and τ the inequality follows from the practicality of σ, by considering the history $(h', h) \in \mathscr{H}$ (player p and strategy τ remain unchanged).

Assuming now there exists a counterexample to practicality, we fix the prefix h of h^*, and the action $a \in A(h)$ of said counterexample. We also assume the game is practical and fix an honest practical strategy $\sigma \in \mathscr{S}$. Then, we construct a strategy $\tau \in \mathscr{S}_{|h}$ as follows: let $\tau(h) := a$, $\tau_{|(h,a)} := \sigma_{|(h,a)}$, and the rest be arbitrary. For these values, the inequality

$$u_{|h,P(h)}(\sigma_{|h}) < u_{|h,P(h)}(\sigma_{|h}[\tau_{P(h)} / \sigma_{|h,P(h)}]) \tag{3.19}$$

holds (as detailed next), which is a contradiction to the practicality of σ.

It remains to show that the above inequality (3.19) is correct. Since σ is an honest strategy, it follows $u_{|h,P(h)}(\sigma_{|h}) = u_{P(h)}(h^*)$. Also

$$H_{|h}(\sigma_{|h}[\tau_{P(h)}/\sigma_{|h,P(h)}]) = (a, H_{|(h,a)}(\sigma_{|(h,a)})) \tag{3.20}$$

by the definition of τ and the fact that at history h it is the turn of player $P(h)$. Consider now that $H_{|(h,a)}(\sigma_{|(h,a)})$ is a practical history in $\Gamma_{|(h,a)}$, since $\sigma_{|(h,a)}$ is practical according to our *observation* above. From (3.20) and the definition of counterexample, it follows

$$u_{|h,P(h)}(\sigma_{|h}) = u_{P(h)}(h^*) < u_{P(h)}(h, a, H_{|(h,a)}(\sigma_{|(h,a)})) = u_{|h,P(h)}(\sigma_{|h}[\tau_{P(h)}/\sigma_{|h,P(h)}]) \; .$$

This concludes the proof of (3.19) and therefore implication $\Rightarrow$.

For the other implication $\Leftarrow$, we proceed by contraposition, which means we assume there exists no counterexample, and show that the game is practical. If there is no counterexample to the practicality of a game, then for all prefixes h of h^* and all actions $a \in A(h)$:

$$\exists t \in \mathscr{T}_{|(h,a)}.\; t \text{ is practical } \wedge\, u_{P(h)}(h^*) \geq u_{P(h)}(h, a, t) \; . \tag{3.21}$$

A terminal history $t \in \mathscr{T}_{|(h,a)}$ is practical iff there exists a practical strategy $\sigma^{(h,a)} \in \mathscr{S}_{|(h,a)}$ that extends t, i.e. $H(\sigma^{(h,a)}) = t$. Hence (3.21) is equivalent to

$$\exists \sigma^{(h,a)} \in \mathscr{S}_{|(h,a)}.\; \sigma^{(h,a)} \text{ is practical } \wedge\, u_{P(h)}(h^*) \geq u_{|(h,a),P(h)}(\sigma^{(h,a)}) \; . \tag{3.22}$$

Based on this, we can now construct a strategy $\sigma \in \mathscr{S}$. For all prefixes h of h^*, let σ pick the honest action a^*, i.e. $\sigma(h) := a^*$. For all other choices $a \in A(h)$ after h, we define $\sigma_{|(h,a)} := \sigma^{(h,a)}$, where $\sigma^{(h,a)}$ is the practical strategy from (3.22).

Strategy $\sigma \in \mathscr{S}$ is fully defined since every history in the game tree is either along the honest history or has a joint prefix with the honest history (possibly empty), at which point a dishonest action $a \neq a^*$ was followed. It further yields the honest history h^* by definition. Only the strategy's practicality remains to be proven. To do so, we fix an arbitrary history $h' \in \mathscr{H}$, player $p \in N$ and strategy $\tau \in \mathscr{S}_{|h'}$.

Case 1. History h' is not along the honest history. Then the required inequality $u_{|h',p}(\sigma_{|h'}) \geq u_{|h',p}(\sigma_{|h'}[\tau_p/\sigma_{|h',p}])$ follows from the practicality of the respective $\sigma^{(h,a)}$. Where h is the common prefix of h^* and h', and a is the deviation choice away from h^*. Hence $h' = (h, a, k)$, where $k \in \mathscr{H}_{|(h,a)}$.

Case 2. History h' is along the honest history and $p = P(h')$. We further distinguish whether τ deviates from σ at h'.

Case 2.1. $a' := \tau(h') \neq \sigma(h')$. Then, by the fact that it is the turn of player $P(h')$ at h' and the definition of σ it follows:

$$\begin{aligned} u_{|h',P(h')}(\sigma_{|h'}[\tau_{P(h')}/\sigma_{|h',P(h')}]) \;&= \\ u_{|(h',a'),P(h')}(\sigma_{|(h',a')}[\tau_{|(a'),P(h')}/\sigma_{|(h',a'),P(h')}]) \;&= \\ u_{|(h',a'),P(h')}(\sigma^{(h',a')}[\tau_{|(a'),P(h')}/\sigma^{(h',a')}_{P(h')}]) \; . \end{aligned}$$

Further, by the practicality of $\sigma^{(h',a')}$ we proceed with

$$u_{|(h',a'),P(h')}(\sigma^{(h',a')}[\tau_{|(a'),P(h')}/\sigma^{(h',a')}_{P(h')}]) \leq u_{|(h',a'),P(h')}(\sigma^{(h',a')}) .$$

Finally, applying (3.22) to the right expression we conclude

$$u_{|(h',a'),P(h')}(\sigma^{(h',a')}) \leq u_{P(h')}(h^*) = u_{|h',P(h')}(\sigma_{|h'}) .$$

Hence, the required inequality is shown.

Case 2.2. $\tau(h') = \sigma(h')$. Consider the first deviation point d of $\sigma_{|h'}[\tau_{P(h')}/\sigma_{|h',P(h')}]$ from h^* (i.e $\tau(h',d) \neq \sigma(h',d)$) if it exists and the corresponding deviating action a_d. Note that $P(h') = P(h',d)$ as only $P(h')$ deviates from σ. Then, by these facts

$$u_{|h',P(h')}(\sigma_{|h'}[\tau_{P(h')}/\sigma_{|h',P(h')}]) = u_{|(h',d),P(h',d)}(\sigma_{|(h',d)}[\tau_{|d,P(h',d)}/\sigma_{|(h',d),P(h',d)}]) .$$

From there, the same reasoning steps as in Case 2.1 apply and yield the required inequality. In case no such deviation point d exists, $\sigma_{|h'}[\tau_{P(h')}/\sigma_{|h',P(h')}]$ and $\sigma_{|h'}$ yield the same history (the honest history). Hence, the inequality trivially holds.

Case 3. History h' is along the honest history and $p \neq P(h)$. If it is never player p's turn after h' along h^*, then $u_{|h',p}(\sigma_{|h'}) = u_{|h',p}(\sigma_{|h'}[\tau_p/\sigma_{|h',p}])$. Thus, the inequality is trivially satisfied.

Otherwise, consider the first time after h' along h^* where it is player p's turn. We call the respective history k. Then $u_{|h',p}(\sigma_{|h'}) = u_{|(h',k),p}(\sigma_{|(h',k)})$, since h' and (h',k) are along h^* and $H(\sigma) = h^*$. Also

$$u_{|h',p}(\sigma_{|h'}[\tau_p/\sigma_{|h',p}]) = u_{|(h',k),p}(\sigma_{|(h',k)}[\tau_{|k,p}/\sigma_{|(h',k),p}]) ,$$

for the same reasons and because player p has no turn in history k. By definition of k, we know that $P(h',k) = p$. Hence, Case 3 reduces to Case 2 with history (h',k) along h^* and $P(h',k) = p$, which was already proven. We, therefore, showed the necessary inequality in all cases, which implies the constructed strategy σ is practical. □

Computing Counterexamples to ϕ_{wi}, ϕ_{weri} and ϕ_{cr}. Within our work, we find counterexamples to (violated) ϕ_{wi}, ϕ_{weri} and ϕ_{cr} using an approach similar to case splitting over term orderings. We first amend formulas ϕ_{wi}, ϕ_{weri} and ϕ_{cr} with suitable labels. This allows us to detect at which histories the checked property is violated and for which player(s), by inspecting labels in the reasoning core. Each history reflects one choice in the counterexample strategy. For the purpose of counterexample generation, the formula for weak immunity ϕ_{wi} is then rewritten as follows:

$$\bigwedge_{p \in N} \bigwedge_{t \in \mathscr{T}} \left(\ell_{p,t} \rightarrow \left[\bigwedge_{(h,a) \in \mathscr{H}^p_t} v^h_a \right] \rightarrow u_p(t) \geq 0 \right) \wedge \ell_{p,t} .$$

The constraint for weaker immunity, ϕ_{weri}, is amended in the following way:

$$\bigwedge_{p\in N}\bigwedge_{t\in\mathscr{T}}\left(\ell_{p,t}\rightarrow\left[\bigwedge_{(h,a)\in\mathscr{H}_t^p} v_a^h\right]\rightarrow \mathsf{real}(u_p(t))\geq 0\right)\ \wedge\ \ell_{p,t}\ .$$

And finally for collusion resilience, ϕ_{cr}, is adjusted as follows:

$$\bigwedge_{S\subset N}\bigwedge_{t\in\mathscr{T}}\left(\ell_{S,t}\rightarrow\left[\bigwedge_{(h,a)\in\mathscr{H}_t^{N\setminus S}} v_a^h\right]\rightarrow \sum_{p\in S}u_p(h^*)\geq\sum_{p\in S}u_p(t)\right)\ \wedge\ \ell_{S,t}\ .$$

We generate counterexamples to ϕ_{wi}, ϕ_{weri} and ϕ_{cr} by enumerating minimal unsat cores [LPMMS16]. From these unsat cores, (groups of) players and partial strategies are identified, yielding counterexamples to the respective security property. Each unsat core represents a counterexample; thus, by listing and interpreting all unsat cores, we generate all counterexamples to ϕ_{wi}, ϕ_{weri} and ϕ_{cr}.

Computing Counterexamples to ϕ_{pr}. Generating counterexamples to ϕ_{pr} is summarized in Algorithm 3.2[3], requiring a different approach than for ϕ_{wi}, ϕ_{weri} and ϕ_{cr}.

Algorithm 3.2 takes as input a game Γ, an honest history h_o and the *case* of term orderings for which practicality analysis failed. Algorithm 3.2 returns a set of counterexamples (CE) to ϕ_{pr}, that is, a set of terminal histories, and a subcase of the initial *case*. Note that the set CE contains all the counterexamples to ϕ_{pr} in the given refined problematic case. In each iteration of the while loop of Algorithm 3.2 (lines 6–25), all practical histories that yield a strictly better utility for the deviator are listed, then the game tree is cut such that new counterexamples can be revealed (line 7). The variables GT (current game tree), h (guidance on what to cut) and *subgame* (subgame to be considered) keep track of the cutting and ensure that the correct terminal histories are added to CE. Additionally, it might be necessary to further specify the ordering of the utility terms, which is considered in line 16 of Algorithm 3.2.

Based on the above considerations, all counterexamples to practicality for a specific term ordering are thus generated via Algorithm 3.2, as proven next.

Theorem 3.5 (Correctness of Algorithm 3.2)**.** *Consider the output* $(\mathsf{CE}, \mathsf{subcase})$ *of Algorithm 3.2. Then, the set* CE *computed by Algorithm 3.2 contains exactly all counterexamples to* h^* *being practical in* $\mathsf{subcase}$[4]*.*

Proof. (i) We first prove that $c\in\mathsf{CE}$ implies c is a counterexample in $\mathsf{subcase}$. Let the honest history be h^* and $c\in\mathsf{CE}$. Assuming $\mathsf{subcase}$, we show that c is of the form

[3]Algorithm 3.2 is based on the assumption made in [BKK+23a] that for a counterexample with history h and action a, there exists only one practical terminal history t in $\Gamma_{|(h,a)}$ and it is summarized as $c:=(h,a,t)$.

[4]This result is also based on the only-one-practical-terminal-history assumption of [BKK+23a].

Algorithm 3.2: Counterexamples to Practicality

input : an input instance $\Pi = (\Gamma, \mathscr{O}, \mathit{inf}, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}})$, an honest history $h_o \in \mathscr{O}$ and the case *case* for which practicality failed

output : the set of all counterexamples CE to the practicality of h_o in a subcase of *case*, together with that subcase *subcase*

```
1  CE ← ∅
2  GT ← Γ
3  ho ← h_o
4  subgame ← ()
5  subcase ← case
6  while ho ¬pr ∧ GT ≠ leaf do
7  |  prStrat ← {h'|h'pr in GT ∧ u_{P(h)}(ho) < u_{P(h)}(h')}
8  |  // h is the maximal common prefix of h' with ho
9  |  if prStrat = ∅ then
10 |  |  // in subgame no more counterexamples
11 |  |  h ← first action of ho
12 |  end
13 |  else
14 |  |  h ← shortest prefix h in prStrat
15 |  end
16 |  subcase.RefineSubcase ()
17 |  // if further case split was done in line 7
18 |  foreach c ∈ prStrat do
19 |  |  CE.Add ( (subgame, c) )
20 |  end
21 |  ho ← ho − h
22 |  subgame ← subgame + h
23 |  GT.Remove (direct subtrees of h that occur in prStrat)
24 |  GT ← GT_{|h}
25 end
26 return CE, subcase
```

$(\mathsf{prefix}, a, h'')$, where prefix is the maximal common prefix of c with h^*, a an action such that $u_{P(\mathsf{prefix})}(h^*) < u_{P(\mathsf{prefix})}(c)$ and h'' practical in $\Gamma_{|(\mathsf{prefix},a)}$.

By Algorithm 3.2 (line 7), when $c = (\mathsf{subgame}, h')$, we have that h' is practical for some ho, *subcase* and GT. Additionally, $u^{\mathsf{GT}}_{P(h)}(\mathsf{ho}) < u^{\mathsf{GT}}_{P(h)}(h')$, where h is the maximal common prefix of h' with ho. Note that in every iteration of the loop ho is a suffix of h^*, as only prefixes h of h^* are cut off (line 11, 14, and 21). Further, $\mathsf{subcase}$ is a subcase of *subcase*, since *subcase* is only refined in the algorithm (line 16). The game GT is the subgame of

Γ after subgame, possibly without some already removed direct subtrees. This holds as we always consider subgames after h (line 24), which are pieces of h^* and are collected in *subgame* (line 22). The direct subtree removal happens in line 23 of Algorithm 3.2. We hence conclude that $u_{P(\mathsf{subgame},h)}(h^*) = u^{\mathsf{GT}}_{P(h)}(\mathsf{ho}) < u^{\mathsf{GT}}_{P(h)}(h') = u_{P(\mathsf{subgame},h)}(c)$ in *subcase*. Additionally, c is a terminal history of the input game Γ. The property left to show is that $h' = (a, h'')$, such that h'' is practical in $\Gamma_{|(\mathsf{subgame},a)}$. Note that only direct subtrees of $\Gamma_{|\mathsf{subgame}}$ have been removed in GT, hence $\mathsf{GT}_{|(a)} = \Gamma_{|(\mathsf{subgame},a)}$. Since $h' = (a, h'')$ is practical in GT, h'' has to be practical in $\mathsf{GT}_{|a} = \Gamma_{|(\mathsf{subgame},a)}$ in *subcase*. Therefore, c is a counterexample in *subcase* and thus also in subcase, concluding direction (i) of the proof.

(ii) We next show the reverse direction of (i); that is, c is a counterexample in subcase implies $c \in \mathsf{CE}$. Let C be the set of all counterexamples in the case subcase. As subcase does not necessarily yield a total order on the utility terms, we fix an arbitrary total order $\leq_s$ on the utility terms which is compatible with subcase. We now order elements $c \in C$ according to

1. length of the common prefix h of c with the honest history h^* (shortest first) and within this group into
2. subgroups according to the value $u_{P(h)}(c)$ (decreasingly). Within these subgroups, the order does not matter.

Towards a contradiction, we assume $c \in C$ is the first (according to the ordering above) that is not in CE. Let $c = (h, a, t)$ and $u_{P(h)}(c) =: u$, such that t is practical in $\Gamma_{|(h,a)}$, $u > u_{P(h)}(h^*)$ and h the maximal common prefix with h^*. We distinguish between the following possible cases.

Case 1: (a, t) is not practical in $\Gamma_{|h}$ in total order $\leq_s$. Thus, there is a $(a', t') \in \mathscr{H}_{|h}$ such that $u' := u_{P(h)}(h, a', t') >_s u$, and t' practical in $\Gamma_{|(h,a')}$ which is a counterexample to (a, t) being practical in $\leq_s$. This also yields a counterexample $c' = (h, a', t')$ to c being practical in $\leq_s$. It also is another counterexample to h^* being practical ($u' >_s u > u_{(P(h))}(h^*)$) in $\leq_s$ and thus subcase. Therefore, $c' \in C$ and it occurs in the same group but in a strictly earlier subgroup than c. Thus, by our assumption that c is the first to not appear in CE, we get $c' \in \mathsf{CE}$.

In Algorithm 3.2, every $c \in \mathsf{CE}$ occurs once in prStrat[5] as the element with the shortest common prefix with h^*. This holds as we always remove at least one direct subtree or move further along h^*. Let us now consider the last occurrence of c' in prStrat. In this iteration, the tree GT will be cut to at most $\Gamma_{|h} - \{a' : c' = (h, a', t')\}$. Since all counterexamples c'' with the same prefix h as c and with greater utilities $u_{P(h)}(c'') >_s u_{P(h)}(c)$ have been found in CE, the corresponding branches are removed from GT in the following iterations of Algorithm 3.2 as well.

[5] only the respective suffix of c occurs since we add the subgame prefix only later to CE.

We consider now under which condition (a,t) from $c=(h,a,t)$ becomes practical in a partial tree of $\Gamma_{|h}$ in subcase: since t is practical in $\Gamma_{|(h,a)}$ it suffices if $u_{P(h)}(h,a,t) \geq u_{P(h)}(h,a'',t'')$ in subcase, for all $a'' \in A(h)$, t'' practical in $\Gamma_{|(h,a)}$. Note that each such (a'',t'') that violates this property yields a counterexample $c''=(h,a'',t'')$ to h^*'s practicality in subcase with a utility strictly better or incomparable to c. We observe that incomparability at this step in the algorithm could not have happened, as it would have caused a further case split (line 16) which contradicts the fact that subcase is the output value of the case split. Let us thus consider the case of strictly better utility: Since $\leq_s$ is compatible with subcase, we know $u_{P(h)}(c'') >_s u_{P(h)}(c)$ implies $u_{P(h)}(c'') > u_{P(h)}(c)$ in subcase. Thus, all such (a'',t'') have at this point already been removed from GT.

Therefore, (a,t) of $c=(h,a,t)$ is practical in this tree, implying that (a,t) has to appear in prStrat in the next iteration and hence also $c \in CE$. This contradicts the assumption of *Case 1.*

Case 2: (a,t) is practical in $\Gamma_{|h}$ in $\leq_s$, yielding two further cases.

Case 2.1: $c=(h,a,t)$ is in the first group C, that is, there is no counterexample with a shorter prefix. In this case, $c=(h,a,t)$ has to be practical in Γ in $\leq_s$. By assuming the contrary, there exists a counterexample $c'=(h',a',t')$, where h' is a prefix of c, t' is practical in $\Gamma_{|(h',a')}$ in $\leq_s$, and $u_{P(h')}(c) <_s u_{P(h')}(c')$. Then, h' cannot be a prefix of h, as this contradicts c being in the first group of C. Further, h can not be a prefix of h', since this contradicts the fact that (a,t) is practical in $\Gamma_{|h}$ in $\leq_s$. Therefore, c is practical in Γ in $\leq_s$, implying however that c has to occur in the first iteration of the loop in prStrat (being practical in subcase). Another case is not possible (similar to Case 1) as a further case split of subcase did not happen.

Case 2.2: $c=(h,a,t)$ is not in the first group in C. Let then h' be the maximal common prefix with h^* in the group directly before c. All the counterexamples c' of this group have $u_{P(h')}(c') > u_{P(h')}(h^*)$. Similar to *Case 1*, as they all appear in CE, we reach the following tree to be considered: $\Gamma_{|h'}$ minus all the immediate dishonest branches that contain counterexamples. In the subsequent iteration of the algorithm, prStrat has to be empty as there are no more branches that make the condition $u_{P(h')}(c') > u_{P(h')}(h^*)$ true. Therefore, we go to the if-branch in line 9 and consider $\Gamma_{|(h',a^*)}$ in the following iteration. The action a^* is the next action in h^* after h'. Note that for the tree $\Gamma_{|(h',a^*)}$, c is in the first group of counterexamples by construction. Therefore, *Case 2.1* applies for the game $\Gamma_{|h',a^*}$, thus counterexample c has to be practical in $\Gamma_{|(h',a^*)}$ and hence has to occur in prStrat in this iteration. This contradicts the assumption that $c \notin$ CE and overall concludes direction (ii) of the proof. □

3.5.3 Inferring Preconditions for Security

In addition to identifying concrete counterexamples (Section 3.5.2) in case a security property ϕ_{sp} is not satisfied in Algorithm 3.1, our work can also determine additional assumptions necessary to ensure that property ϕ_{sp} holds. We support such an extended security analysis by iteratively computing preconditions for the input EFG.

Definition 3.11 (Preconditions to Security)**.** *Given a game* Γ*, an honest history* h^**, a security property* sp *and initial conditions* C *and* C_{sp} *such that* h^* *violates* sp*, we say* π *is a* precondition *if*

$$\forall\, (\preceq, T_u) \text{ total order. } \exists\, \mathsf{I} \in \mathcal{I}.$$
$$\mathsf{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{sp} \cup \{\pi\}} c[\vec{x}] \rightarrow (\phi_{strat} \wedge \phi_{hist} \wedge \phi_{sp}[\vec{x}])\right) = \top.$$

We note that in Definition 3.11, the precondition π strengthens the initial constraints by making the left-hand side of the above implication satisfiable for fewer total orders, thus ensuring validity of the above implication for more of them.

In Algorithm 3.3, we show our adjustment of Algorithm 3.1, allowing us to generate the weakest precondition under which a security property sp is satisfied. If precondition generation is enabled in our work, the solving routine at line 16 in Algorithm 3.1 does not terminate; instead, it adds the negation of the ordering in the unsolved case O to preconditions in C (the variable pre in line 20 of Algorithm 3.3) and restarts the solving routine with the new set of initial constraints. With such an adjustment of Algorithm 3.1, Algorithm 3.3 allows us to generate *weakest preconditions* to sp, in the following sense: if π and ψ are preconditions, then π is *weaker* than ψ if

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}]$$

is satisfiable for more total orders $\preceq$ than

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\psi\}} c[\vec{x}].$$

For proving correctness of Algorithm 3.3, we first prove the following helping lemma.

Lemma 3.6 (Unique Weakest Precondition)**.** *Given a game* Γ*, an honest history* h^**, a security property* sp *and finite sets of initial constraints* C *and* C_{sp}*, there exists a unique (modulo equivalence) weakest precondition* π *to make history* h^* *satisfy* sp*.*

Proof. Let π be a precondition, by Definition 3.11, it holds that

$$\forall\, (\preceq, T_u) \text{ total order. } \exists\, \mathsf{I} \in \mathcal{I}.$$
$$\mathsf{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}] \rightarrow (\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}])\right) = \top.$$

As noted after Corollary 3.1, the satisfiability of the formula depends only on the finitely many total orders of terms in T_u. Thus, any precondition can be weakened to a list of term orderings to be avoided, which are finitely many. The weakest of these is the one

Algorithm 3.3: Generating Weakest Preconditions

input : an input instance $\Pi = (\Gamma, \mathscr{O}, \mathit{inf}, C, C_{\text{wi}}, C_{\text{weri}}, C_{\text{cr}}, C_{\text{pr}})$, an honest history $h_o \in \mathscr{O}$ and the name of a security property $\text{sp} \in \{\text{wi}, \text{weri}, \text{cr}, \text{pr}\}$

output : weakest precondition under which ϕ_{sp} is satisifiable in Π

```
1  S ← Solver()
2  A ← Solver()
3  pre ← true // the precondition to be constructed
4  φ ← true // the conflict clauses to reach the next case
5  AddConstraints (S, ComputeStrategyConstraints (Γ, h_o))
6  AddConstraints (A, C ∪ C_sp)
7  T ← ∅
8  while Check (A, φ) = sat do
9      M ← GetModel (A, φ)
10     O ← {EvaluateModel (M, x, y) : (x, y) ∈ Combinations (T)}
11     if Check (S, sp, C ∪ C_sp ∪ {pre} ∪ O) = sat then
12         // found strategy for current ordering, add conflict clause
13         φ ← φ ∧ (⋁_{c∈O} ¬c)
14     end
15     else
16         T' ← ∅
17         foreach ℓ_{x,y} ∈ GetUnsatCore(S) do
18             T' ← T' ∪ {t : t ∈ {x, y}, t ∉ T }
19         if T' = ∅ then // no new expressions, case is unsat
20             pre ← pre ∧ (⋁_{c∈O} ¬c)
21             φ ← φ ∧ (⋁_{c∈O} ¬c)
22         end
23         T ← T ∪ T'
24     end
25 end
26 return pre
```

that allows precisely all total orderings of terms in T_u that are satisfiable and forbids all others, and is thus unique up to equivalence (as a quantifier-free first-order formula can be expressed in many equivalent ways). □

We next proceed with proving correctness of Algorithm 3.3.

Theorem 3.6 (Weakest Preconditions – Correctness of Algorithm 3.3)**.** *Given a game* Γ*,*

an honest history h^* and a security property sp *that h^* violates, Algorithm 3.3 generates the weakest precondition π which makes h^* satisfy* sp.

Proof. As Algorithm 3.3 adjusts Algorithm 3.1, we only prove that whenever Algorithm 3.3 terminates (which it does, because we only need to consider finitely many total orders of T_u), the generated precondition π (i) is indeed a precondition and (ii) is the weakest.

(i) By construction, π is a conjunction of negated term orderings:

$$\pi = \left(\bigvee_{c \in \mathsf{O}_1} \neg c\right) \wedge \cdots \wedge \left(\bigvee_{c \in \mathsf{O}_n} \neg c\right).$$

Let us write $\neg O_i$ for $\left(\bigvee_{c \in \mathsf{O}_i} \neg c\right)$. By construction, we leave out only the term orderings for which the security property holds (line 11 in the Algorithm 3.3), and hence π is a precondition.

(ii) Let ψ be the weakest precondition given by Lemma 3.6. Towards a contradiction, assume that ψ is strictly weaker than π; that is, there exists a total order $\preceq$ such that

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\psi\}} c[\vec{x}] \tag{3.23}$$

is satisfiable and

$$\bigwedge_{c \in \preceq \cup C \cup C_{\mathrm{sp}} \cup \{\pi\}} c[\vec{x}] \tag{3.24}$$

is unsatisfiable. Let M be a model for $\vec{x}$ that satisfies (3.23). Since M does not satisfy (3.24), we know that $M(\pi) = \texttt{false}$. Let O_i be the first ordering in the conjunction π such that $M(\neg O_i) = \texttt{false}$. Consider line 20 of Algorithm 3.3 at the point when $\neg O_i$ was added to the precondition π: the else case of the condition of line 11 implies that the security property is not satisfied in the case O_i; further, line 19 implies that there are no more comparisons to be added. As such, for every total order $\preceq_i$ that implies the case O_i, we have

$$\forall\, \mathsf{I} \in \mathcal{I}.\, \mathsf{I}\left(\forall \vec{x}. \bigwedge_{c \in \preceq_i \cup C \cup C_{\mathrm{sp}}} c[\vec{x}] \rightarrow (\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}])\right) = \texttt{false},$$

which is equivalent to

$$\forall\, \mathsf{I} \in \mathcal{I}.\, \exists \vec{x}. \bigwedge_{c \in \preceq_i \cup C \cup C_{\mathrm{sp}}} c[\vec{x}] \wedge \neg\, \mathsf{I}\left(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]\right). \tag{3.25}$$

The inequalities in $\phi_{\mathrm{sp}}[\vec{x}]$ only depend on the total order $\preceq_i$ and not on the actual values of $\vec{x}$. Thus,

$$\exists \vec{x}. \bigwedge_{c \in \preceq_i \cup C \cup C_{\mathrm{sp}}} c[\vec{x}] \wedge \neg\, \mathsf{I}\left(\phi_{\mathrm{strat}} \wedge \phi_{\mathrm{hist}} \wedge \phi_{\mathrm{sp}}[\vec{x}]\right)$$

implies

$$\forall \vec{x}. \bigwedge_{c \in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}])\,.$$

Using (3.25), we get

$$\forall\, \mathsf{I} \in \mathcal{I}.\, \forall \vec{x}. \bigwedge_{c \in \preceq_i \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}])\,. \tag{3.26}$$

Consider now $\preceq_M$ the total order on T_u induced by the model M. Since $M(\neg O_i) = \texttt{false}$, we know that M satisfies O_i, so $\preceq_M$ implies O_i as well and (3.26) holds for $\preceq_M$ too. As ψ is a precondition, by using Definition 3.11 with $\preceq_M$, we obtain $\mathsf{I}' \in \mathcal{I}$ such that

$$\forall \vec{x}. \bigwedge_{c \in \preceq_M \cup C \cup C_{\text{sp}} \cup \{\psi\}} c[\vec{x}] \rightarrow \mathsf{I}'\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}]) \tag{3.27}$$

holds. Using (3.26) with total order $\preceq_M$ and $\mathsf{I}' \in \mathcal{I}$, we obtain

$$\forall \vec{x}. \bigwedge_{c \in \preceq_M \cup C \cup C_{\text{sp}}} c[\vec{x}] \rightarrow \neg\, \mathsf{I}'\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[\vec{x}])\,. \tag{3.28}$$

With the model M for $\vec{x}$, the antecedent of the implication in (3.27) holds as M satisfies (3.23) (and thus satisfies the constraints in C, C_{sp} and ψ). Therefore, the formula $\mathsf{I}'\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[M(\vec{x})])$ holds. With the model M for $\vec{x}$, the antecedent of the implication in (3.28) is also valid, yielding $\neg\, \mathsf{I}'\,(\phi_{\text{strat}} \wedge \phi_{\text{hist}} \wedge \phi_{\text{sp}}[M(\vec{x})])$. We thus obtained a contradiction, concluding the proof. □

Example 3.14. The weak immunity property ϕ_{wi} of the game from Example 3.10 for honest history (l, r) is not satisfied if $b < 0$. Our work identifies this case and reports that there is no weak immune joint strategy given that $b < 0$. The solving routine of Algorithm 3.1 is restarted with $b \geq 0$ added to C in Algorithm 3.3, triggering the satisfiability of the such revised weak immunity property ϕ_{wi}. In this case, Algorithm 3.3 returns one additional precondition, namely $b \geq 0$.

We conclude this section by noting that finding sufficient (nontrivial) preconditions to satisfy a security property ϕ_{sp} is not always possible with Algorithm 3.3: if the existing preconditions C imply the ordering O for which there is no adequate joint strategy, the security property ϕ_{sp} cannot be satisfied by adding more assumptions about the EFG utilities. In this case, the weakest precondition returned by Algorithm 3.3 is `false`.

3.6 Implementation and Experimental Evaluation

Implementation. We implemented our game-theoretic security analysis in the new CheckMate tool[6], written in Python and using Z3 [DMB08] as the underlying SMT

[6]Our tool is available at `https://github.com/apre-group/checkmate/tree/ccs23`.

Game	Nodes	Players	History	Security	Calls	Time
$\text{Splits}_{\text{wi}}$	5	2	(q)	✓	8	0.38
$\text{Splits}_{\text{cr}}$	5	2	(n)	✓	10	0.76
Market Entry	5	2	(e, i)	✗ (wi,weri)	6	0.32
Pirate	52	4	(y, n, n, n, y, y)	✗ (wi,weri,cr,pr)	11	1.88
Simplified Closing	8	2	(H)	✗ (pr)	5	0.36
			(C_h, S)	✗ (wi,weri)	6	0.36
Simplified Routing	17	5	$(S_H, L, L, L, L, U, U, U, U)$	✗ (wi,cr)	6	0.57
Closing	221	2	(H)	✗ (pr)	5	16.4
			(C_h, S)	✓	6	17.8
3-player Routing	21,688	3	(S_H, L, L, U, U)	✗ (wi,cr,pr)	149	1222

Table 3.1: CheckMate results, time in seconds. ✓means that all security properties are satisfied. ✗(sp) indicates that the history is not secure as property "sp" failed.

solver of Algorithm 3.1. The input for CheckMate is a JSON [ISO17] encoding of an input instance Π of Algorithm 3.1, which is also the current representation language for the EFG. The instance Π is parsed into an internal representation, which is used to construct SMT constraints for (any subset of) the security properties presented in Section 3.4. CheckMate then executes Algorithm 3.1 for each property, logging intermediate results such as case splits. CheckMate outputs a joint strategy if a property is satisfied, or a list of counterexamples otherwise; in the latter case, CheckMate also produces preconditions (Section 3.5.3). In addition, CheckMate supports a *verification* mode to check whether a joint strategy satisfies a security property.

CheckMate only supports EFGs with finite game trees. However, EFGs are known to have the capacity to model many protocols. Further, every step in the game can model an arbitrary long time in the protocol (e.g., the "ignore" actions model unlimited time). An advantage of the trees being finite is that the game-theoretic properties will only quantify over finite sets, implying that the respective security properties can be translated to first-order linear real arithmetic (as shown in Section 3.4).

Benchmarks. We evaluated CheckMate using the examples of Section 3.2, Examples 3.15 to 3.17, which are briefly described below, as well as real-world blockchain protocols from Bitcoin's Lightning Network [PD16]. These examples are listed as the

last four entries of Table 3.1. Namely, we used the complete model of the Closing Game of [RAKM23] and the full model of the routing phase corresponding to the Routing Game of [RAKM23] with three players. The honest behavior of the full routing game is similar to the Simplified Routing Game, but players have many more actions available, such as sharing the secrets or varying the way they lock funds. The full Routing Game for five (and more) players is a task for future work.

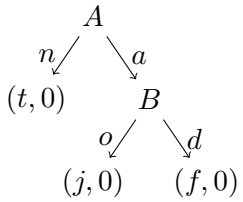


Figure 3.4: $\text{Splits}_{\text{cr}}$ Game with $t > 0$ and $j > t > f \vee f > t > j$.

Example 3.15 ($\text{Splits}_{\text{cr}}$)**.** In the game depicted in Figure 3.4, player A starts by choosing between action n and action a. Picking action n results in a utility of t for player A, with $t > 0$. If they take action a, player B has to choose either action o or action d. Action o yields the utility j for player A, while action d results in the utility f, with $j > t > f \vee f > t > j$. Player B always receives a payoff of 0 in this game.

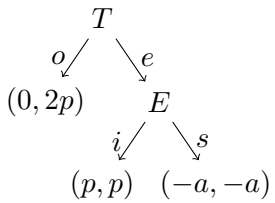


Figure 3.5: Market Entry Game with $a, p > 0$.

Example 3.16 (Market Entry Game)**.** The Market Entry Game displayed in Figure 3.5 is an adaptation of the Chain-Store Game defined in [OR94]. Player T starts by either choosing action o or action e. If they pick action o, the game ends, yielding the utility 0 for T and $2p$ for E, with $p > 0$. Otherwise, it is player E's turn: E either takes action i, resulting in utility p for both players, or action s, which yields the utility $-a$ for both players, with $a > 0$.

Example 3.17 (Pirate Game)**.** The Pirate Game displayed in Figure 3.6 (and Figure 3.7 and Figure 3.8, respectively, for the subgames S_B and S_C) is an adapted version of the "puzzle for pirates" introduced by Stewart [Ste99]. It models a voting scenario: Each player proposes a joint utility whose sum is g. First, the players decide if they want to accept A's proposed distribution (in alphabetical order). If the majority of players are in favor of the proposal (indicated by taking action y when it is their turn), the game

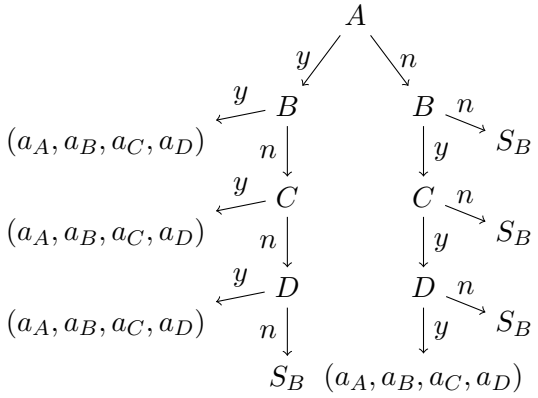


Figure 3.6: Pirate Game with $d, g, a_p, b_p, c_p \geq 0$ for $p \in N$ and $g = a_A + a_B + a_C + a_D$, $g = b_B + b_C + b_D = c_C + c_D$. The subgame S_B is displayed in Figure 3.7.

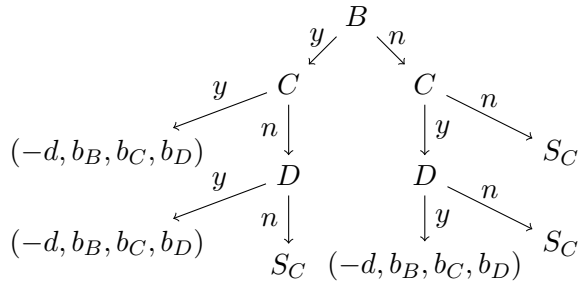


Figure 3.7: Subgame S_B of the Pirate Game (Figure 3.6).

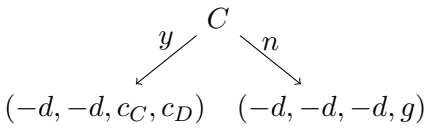


Figure 3.8: Subgame S_C of the Pirate Game (Figures 3.6 and 3.7).

ends with the utility a_A for player A, a_B for player B, and so on. Otherwise, i.e., if the majority picks action n, player A is eliminated from the game, which results in the utility $-d$ for player A when the game ends, with $d > 0$. The process repeats with the joint utility proposed by player B, where player B gets the utility b_B, C receives b_C, etc. If B's proposal is rejected, B is eliminated as well, and the remaining players vote on player C's proposed distribution. In case of a tie, the decision of the proposing player is the casting vote.

As the modeling process of real-world protocols as EFGs is an intricate and time-consuming process, representing a challenge on its own, we so far have a restricted but representative benchmark set, showcasing the logical expressivity provided by CheckMate to efficiently

Game	Property	CE	PC
Market Entry	wi	1	`false`
(e, i)	weri	1	`false`
Pirate	wi	141	`false`
(y, n, n, n, y, y)	weri	141	`false`
	cr	>80	`false`
	pr	1	π
Simplified Closing (H)	pr	1	`false`
Simplified Closing (C_h, S)	wi	1	`false`
	weri	1	`false`
Simplified Routing	wi	9	`false`
$(S_H, L, L, L, L, U, U, U, U)$	cr	16	`false`
Closing (H)	pr	5	`false`
3-player Routing	wi	41	`false`
(S_H, L, L, U, U)	cr	2	`false`
	pr	>0*	TO

Table 3.2: Counterexample (CE) and Precondition (PC) analysis provided by CheckMate. $>N$ indicates that we found N counterexamples in 10 minutes, but generation has not terminated yet. The practicality result (*) for 3-player Routing is computationally demanding, but begins to produce results after the time limit. At around 60 minutes, we have over 100 counterexamples. TO indicates a timeout after 2 hours.

(dis)prove blockchain security. We are not aware of other efforts providing practical challenges for evaluating formal methods in support of blockchain security. We believe our EFG examples provide an initial set of examples to be further used in verifying blockchain security.

Experimental Results. Table 3.1 summarizes our experimental results, using an Apple M1 Pro CPU with 10 cores and 32 GB of RAM. The sizes of our EFG examples in terms of nodes and players are listed in columns 2–3. For each honest history (column 4), Table 3.1 displays CheckMate's results for game-theoretic security. We also report execution times for complete analysis of all properties, without counterexample generation. In column 6 (Calls), the number of SMT calls within CheckMate is listed. Both protocol benchmarks (the Closing Game and the 3-Player Routing Game) hint that even for bigger game trees, the number of case splits is relatively low.

Table 3.2 displays the number of counterexamples found for each violated property, as well as a precondition strong enough to make the respective property true. We write `false` to indicate that no precondition (except `false`) is strong enough to satisfy the property. Thus, the problem is inherent to the game and is not a matter of the variable values. Due to its size, we do not state the nontrivial precondition π of the Pirate Game, generated by CheckMate in 37 seconds.

Experimental Analysis. CheckMate successfully analyzes the games of Section 3.2 and correctly identifies the Wormhole attack as one of the 16 counterexamples to collusion resilience in the Simplified Routing Game. Applied to the Closing Game, CheckMate terminates after about 17 seconds for each honest history and correctly identifies required case splits for history (C_h, S). The Routing Game is not weak immune, but *weaker* immune: honest players will not lose "real" resources but may suffer opportunity cost. The 3-player Routing Game has 21,688 nodes, compared to 221 nodes for the Closing Game.

Our experiments demonstrate the usability and scalability of CheckMate. For example, for analyzing real-world applications (closing and routing phases in the last two lines of Table 3.1), CheckMate enabled the automation of reasoning about trillions of game strategies and thousands of game nodes. We are not aware of other automated reasoning approaches handling such and similar EFGs.

The mostly trivial preconditions in Table 3.2 are not surprising, since we asserted constraints we were aware of as initial conditions. For the Closing Game and history (C_h, S), for example, we inherited initial constraints $a, b \geq f$ for weak immunity and $c \neq p_A$ for practicality from [RAKM23]. When removing those constraints, CheckMate finds a precondition equivalent to $a, b \geq f$ for weak immunity and one equivalent to $c \neq p_A \vee b - p_A + d_B = f$ for practicality. Thus, CheckMate provides a less restrictive but still sufficient precondition than [RAKM23].

3.7 Related Work

Game theory opens up new venues in security and privacy analysis [DTH+17], particularly within blockchain technologies [LNW+19]. Game-theoretic modeling approaches in support of blockchain security have recently emerged [ZBPBS21, RAKM23], complemented by specific analysis of several attack vectors, such as the griefing attack [MBS+22]. Our work complements these efforts as we express game-theoretic security in first-order linear real arithmetic and turn protocol security into an SMT-solving problem. Extended with game-theoretic case splitting and counterexample generation, we scale game-theoretic security to large protocols and fully automate it. Our work is thus orthogonal to [RAKM23].

The work of [CGP19] focuses on game-theoretic security specific to the Ethereum blockchain [Woo14], but does not reason about punishment mechanisms. While these works provide rigorous game-theoretic models, they lack support for automated reasoning, hindering their scalability and computer-aided certification.

Focusing on formal verification for security, the Tamarin [MSCB13], Verifpal [KNT20], and Proverif [Bla14] frameworks study general protocols, while the Verisol approach targets the Ethereum blockchain [WLC+19]. These methods operate on cryptographic security properties, proving whether certain actions are cryptographically possible. Our

work complements these techniques by analyzing and establishing whether punishment mechanisms work as intended.

A similar line of research is analysis of cryptographic security properties in rational cryptography [GLR13, AL09, Gra10, IML05, LT06]. Rational cryptography does not assume players to be inherently honest or malicious, but instead *rational*. However, rational cryptography verifies cryptographic alignment of rationality and honesty, whereas our work in CheckMate focuses on incentive/punishment mechanisms modeled via games.

Another game-theoretic security reasoning framework was introduced [KNPS20, KNPS21]. Here, concurrent stochastic game structures (CGSs) are modeled within probabilistic asynchronous-time temporal logic with rewards (rPATL). The respective CGSs are verified via model-checking of PATL formulas. Unlike this framework, our work does not reason with uncertainty. Instead, we provide a decidable first-order logic fragment for proving game-theoretic security. Thanks to our logical precision, we can avoid the computational burden of handling probability (reward) operators. Our EFGs also avoid concurrent game strategies and provide a deterministic game structure.

When it comes to automated game-theoretic analysis, it is worth noting the modeling and verification support offered by Gambit [MMT05], PRISM-games [KNPS20] and Open Games [GHWZ18]. These frameworks, however, are restricted to constant numeric utilities and cannot process symbolic utilities, limiting their applicability to reason about all game actions/strategies. We are not aware of other game-theoretic frameworks that natively support symbolic utilities, which is a key feature of CheckMate. Compositional reasoning in Open Games [GHWZ18] does enable modular analysis, which we aim to further investigate in order to split large EFGs (such as routing games) into subgames.

3.8 Conclusion

Game-theoretic security analysis provides new ways to derive security guarantees and identify attack vectors. Yet, automation in this area was so far limited, if at all, hindering the applicability and scalability of game-theoretic security analysis. Our work addresses these obstacles and implements novel methods for deciding security properties in games with symbolic utilities. We reduce security analysis to satisfiability solving over game strategies expressed in first-order theory of linear arithmetic. We provide full automation within the new CheckMate framework, scaling security analysis to very large game trees, by automatically identifying necessary case splits for efficient reasoning. In addition, CheckMate supports the generation of counterexamples and necessary preconditions to security, enabling model repair and synthesis of game properties.

Limitations and Challenges for Future Work. Modeling protocols requires considerable expertise, and the precision of our analysis depends on the accuracy of the underlying model. To maximize application of the CheckMate framework, we must

provide ways to accurately and efficiently model protocols as EFGs. Partial automation of the EFG modeling process is therefore a valuable task to be addressed further.

Currently, CheckMate only supports linear utilities. However, protocols exist that naturally involve nonlinear terms over utilities. For example, ratios in market settings produce terms containing multiplication of two variables. While CheckMate internally already supports nonlinear arithmetic formulas over utilities, it comes with the caveat that the resulting SMT queries are also nonlinear and therefore significantly harder. Improving the performance of SMT solving over nonlinear arithmetic in the setting of CheckMate is its own challenge, which we aim to address in the future.

Finally, extending CheckMate with compositional game analysis is another direction we are already investigating. We believe compositionality may enable us to support randomized game aspects, for example, model behavior that is not controllable by any player. In addition, compositional game analysis might ease modeling further real-world protocols as EFGs.

CHAPTER 4

The Game-Theoretic Security Tool CheckMate

This chapter is based on the conference paper [RBK+24]:

Sophie Rain, Lea Salome Brugger, Anja Petković Komel, Laura Kovács, and Michael Rawson. Scaling Checkmate for Game-Theoretic Security. In Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, pages 222–231, Stockport, UK, 2024.

4.1 Problem Statement

Ensuring the security of decentralized protocols becomes even more critical in the context of decentralized finance. Once deployed on the blockchain, vulnerabilities cannot be corrected and have the potential for significant monetary loss. Various existing approaches for the analysis and verification of blockchain protocols [Bla14, Cer, HBS23, MSCB13, OMA+23, TMSS23, WLC+19] focus on cryptographic and algorithmic correctness or, in other words, whether it is possible to steal assets or gain secret information. However, *economic* aspects must also be considered: whether it is possible for a group of users to profit from unintended behavior within the protocol itself, leading to vulnerabilities [MMSS+19]. Algorithmic game theory [Hal08, OR94] precisely captures such economic aspects.

This chapter describes our open-source tool CheckMate[1] for the automation of game-theoretic protocol analysis. To the best of our knowledge, CheckMate is the first fully automated tool that enforces game-theoretic security. CheckMate constructs and proves game-theoretic *security properties* in the *first-order theory of real arithmetic* while ensuring that game-theoretic security is precisely captured via Byzantine fault tolerance

[1] available at `https://github.com/apre-group/checkmate/tree/lpar25`

and incentive compatibility of the analyzed protocol. As introduced in our previous work [BKK+23a], Byzantine fault tolerance of a protocol guarantees that as long as users follow protocol instructions, they cannot be harmed, independently of how other users behave. Incentive compatibility ensures that the intended course of action is also the most profitable to the users, implying that no user has an economic incentive to deviate. We refer to this intended course of action as *honest behavior*, captured by an *honest history* in game theory.

Following our previous work [RAKM23], inputs to CheckMate are *extensive form games* (EFGs). CheckMate translates Byzantine fault tolerance into the EFG property *weak(er) immunity*, whereas incentive compatibility is expressed in CheckMate via the EFG properties *collusion resilience* and *practicality*. As such, protocol verification in CheckMate becomes the task of proving weak(er) immunity, collusion resilience, and practicality, for which CheckMate implements novel reasoning engines in first-order arithmetic.

The purpose of this tool chapter is to describe what CheckMate *can do (Section 4.2) and how it can be used (Section 4.3).* Theoretical details are covered in our previous work [BKK+23a], but we also improve the algorithmic setting here. CheckMate is no longer restricted to linear input constraints, improves case splitting over arithmetic formulas, and revises counterexamples to practicality, as well as weakest precondition generation and strengthening. For efficiency reasons, CheckMate has an entirely new implementation in C++, using about 2,800 lines of code tightly integrated with the satisfiability modulo theory (SMT) solver Z3 [DMB08]. Our experimental results show the practical gains made over our previous work and also add 7 new benchmarks to the landscape of game-theoretic security analysis. Overall, we used CheckMate to decide the security of 15 benchmarks, including five based on real-world protocols.

4.2 Structure and Components

CheckMate analyzes game-theoretic security of game models. Given an EFG G, CheckMate decides whether G satisfies the security properties of (i) weak(er) immunity – denoted *wi* respectively *weri* in Figure 4.1, (ii) collusion resilience – *cr*, and (iii) practicality – *pr*. Properties (i)-(iii) imply game-theoretic security of G [BKK+23a].

Pipeline. Figure 4.1 summarizes the CheckMate pipeline. After parsing and preprocessing an input EFG G, CheckMate processes one honest history and security property (i)–(iii) at a time. For this history and property, CheckMate constructs *an SMT formula ϕ such that ϕ is satisfiable iff the EFG satisfies the security property* – without the need for further case analysis – with respect to the history. To this end, the SMT formula ϕ consists of three constraints, listed as (a)–(c) in the following. The constraints (a)–(c) are based on the central concept of encoding each action in the game as a Boolean variable that is set to true iff the corresponding action is taken in the game. In more detail, (a) the joint strategy constraint ensures that exactly one action

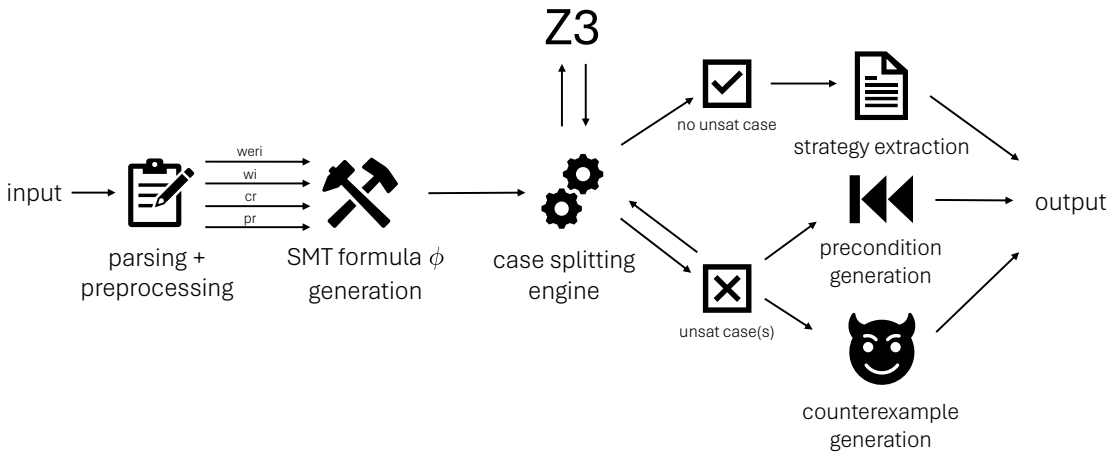


Figure 4.1: The CheckMate Pipeline.

is chosen at each turn. Further, (b) the honest history constraint guarantees that the chosen set of actions yields the honest history. Finally, (c) the property constraint uses a universally-quantified formula to enforce that all variables occurring in the player's pay-offs satisfying a set of preconditions also satisfy the respective security property (i)–(iii) of the EFG.

The formula ϕ is passed to the case splitting engine of CheckMate, which calls Z3 iteratively to decide whether ϕ is satisfiable. If not, case analysis is applied and the resulting constraints are added in turn to the preconditions of constraint (c) of ϕ. This iterative process is terminating as CheckMate is both sound and complete [BKK+23a]. If ϕ is satisfiable, we extract a model – if required by the user – and output the result. If ϕ is unsatisfiable and no further case splits apply, CheckMate implements various actions controlled by command-line options: listing cases that violate ϕ; producing counterexamples witnessing *why* ϕ was violated; and/or computing the weakest precondition that, if added to G as an additional constraint, satisfies ϕ. We describe the main components of CheckMate using Figure 4.2.

Illustrative Example. The EFG in Figure 4.2 has two players, A and B. Nodes represent the player whose turn it is and edges their choices. On reaching a leaf, the game ends, and the pay-off *utility* for each player is given. Here, player A starts and chooses between actions l_A and r_A. If l_A is chosen, the game ends, and A receives utility $a-1$, whereas player B receives a. Otherwise, r_A is chosen, and player B continues in a further subgame. We assume $a > 0$ and specify the *honest history* of G to be (r_A, l_B): that is, we fix the "honest" choice of player A to be r_A and of B to be l_B.
When analyzing whether G satisfies the security property of weak immunity (*wi*), CheckMate constructs the SMT formula ϕ with the following three components: (a) the joint strategy constraint given by $(l_A \vee r_A) \wedge \neg(l_A \wedge r_A) \wedge (l_B \vee r_B) \wedge \neg(l_B \wedge r_B)$; (b) the honest history constraint captured by $r_A \wedge l_B$; and (c) the property constraint of

$\forall a, b.\ a > 0 \rightarrow wi(G)$.

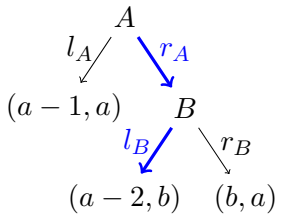


Figure 4.2: Game G with $a > 0$, honest history (r_A, l_B).

4.2.1 CheckMate Input

CheckMate takes as input a JSON file [ISO17] with a specific structure containing the EFG to be analyzed together with its honest histories. Figure 4.3 shows the encoding of the EFG from Figure 4.2, conforming to the JSON schema of CheckMate[2]. The schema defines the structure of the input as an object with the following keys:

`players` A list of all players, represented as strings.

`actions` A list of all possible actions throughout the game, represented as strings.

`constants` Symbolic constants occurring in the players' utilities.

`infinitesimals` Symbolic constants occurring in the players' utilities that are treated as infinitely closer to 0 than the constants in `constants`. Symbolic values in utilities must be included either in `infinitesimals` or `constants`.

`initial_constraints` Initial constraints to be enforced on the otherwise unconstrained symbolic values in utilities.

`property_constraints` Further initial constraints specifically for each security property of weak(er) immunity, collusion resilience, or practicality. This key lets the user specify the weakest possible assumptions for each security property.

`honest_histories` A list of honest histories, i.e., each history is one of the desired courses of EFG actions. Each history is the game-theoretic behavior that is (dis-)proved secure by CheckMate sequentially. An honest history is a list of actions; therefore, this key expects a list of lists of strings.

`tree` The structure of the EFG. Each node in the game tree is either a branch or a leaf. Each branch is represented by an object with the following keys:

`player` The name of the player whose turn it is.

[2] `input.schema.json` in the repository (`https://github.com/apre-group/checkmate/tree/lpar25`)

```
{ "players": ["A","B"],
 "actions": ["l_A","r_A","l_B","r_B"],
 "constants": ["a","b"],
"infinitesimals": [],
"initial_constraints": ["a > 0"],
"property_constraints": {"weak_immunity": [],
                         "weaker_immunity": [],
                        "collusion_resilience": [],
                         "practicality": []},
"honest_histories": [["r_A","l_B"]],
 "tree" : {
    "player": "A",
    "children": [
     {"action": "l_A",
      "child": {"utility": [{"player": "A","value": "a-1"},
                            {"player": "B","value": "a"  }]}},
     {"action": "r_A",
      "child": {"player": "B",
                "children": [
                 {"action": "l_B",
                  "child": {"utility": [{"player": "A","value": "a-2"},
                                        {"player": "B","value": "b" }]}},
                 {"action": "r_B",
                  "child": {"utility": [{"player": "A","value": "b"},
                                 {"player": "B","value": "a"}]}}]}}]}}
```

Figure 4.3: CheckMate Input Encoding Figure 4.2.

children A list of branches the player can choose from. Each branch is encoded as another object with keys `action` and `child`. The `action` key provides the action that the player takes to reach `child`, another tree.

Each leaf of `tree` is encoded as an object with a single key `utility`. As leaves represent one way of finishing the game, it contains the pay-off information for each player in this scenario. `utility` contains the players' utilities, using these keys:

player The name of the player.

value The player's utility. This can be any term over infinitesimals, constants, and reals provided as strings.

CheckMate Formulas. CheckMate uses infix notation in arithmetic and Boolean expressions over real numbers, constants, and infinitesimals declared in the input. It supports +, −, and $\star$ in arithmetic expressions with the usual meanings, but multiplication is allowed only if at least one of the multiplicands is not an infinitesimal. The Boolean expressions =, !=, <, <=, >, and >= have their usual meanings. Booleans can be combined only with disjunction spelled |, but this is not a limitation in practice.

Example (EFG in JSON format). In the JSON encoding of Figure 4.3 corresponding to the game G of Figure 4.2, we have the following keys. The players are `A` and `B`, the actions of G are `l_A`, `r_A`, `l_B`, and `r_B`. The only symbolic values of G are `a` and `b`. None of them are supposed to be infinitesimals; thus, they are both listed under `constants`. The only initial constraint we enforce is that `a` is strictly positive, that is, `a > 0`, as specified in the caption of Figure 4.2. We do not assume any property constraints, so the corresponding lists in Figure 4.3 are empty. As defined in Figure 4.2, we consider (`r_A`, `l_B`) the only honest history. In G, it is player `A`'s turn at the first internal node, which has two children. The first child, which is led to through action `l_A`, is an internal node containing utilities `a-1` for player `A` and `a` for player `B`. The other child, accessible via action `r_A`, leads to another internal node, where it is the turn of player `B`.

4.2.2 CheckMate Output

Given an input as detailed in Section 4.2.1, CheckMate analyzes each specified security property (`<current property>`) for each honest history (`<current honest history>`). To this end, CheckMate answers the following question in its output:

`Is history <current honest history> <current property>?` (Q)

Figure 4.4 shows the CheckMate output for the input of Figure 4.3, when considering the security property of weak immunity.

By answering the above question (Q), CheckMate outputs intermediate logs about necessary case splits, term comparisons used during splitting, and partial results during CheckMate reasoning. Intermediate logs are displayed via indentation in the CheckMate output (see lines 4–6 in Figure 4.4). A partial CheckMate result indicates the satisfiability of the considered security property in the currently analyzed case (line 6 in Figure 4.4). Once an answer to (Q) is derived (line 8 of Figure 4.4), CheckMate reports – without indentation – either

- `NO, it is not <current property>`, in which scenario the EFG does not satisfy the analyzed security property and is, therefore, *not game-theoretically secure*;
- `YES, it is <current property>`, in which case the EFG with the considered honest history has the analyzed property and *may be game-theoretically secure*. If a game and a history satisfy *each security property*, that is, not only the one currently analyzed but each of the three properties of weak(er) immunity, collusion resilience, and practicality, the EFG is *game-theoretically secure*.

In addition, CheckMate can be instrumented by the user to also report on strategies (Section 4.2.4), counterexamples (Section 4.2.5), and weakest preconditions (Section 4.2.6) produced while answering question (Q).

```
1   WEAK IMMUNITY
2
3   Is history [r_A, l_B] weak immune?
4           Require case split on (>= b 0.0)
5           Require case split on (>= (- a 2.0) 0.0)
6           Case [(>= b 0.0), (>= (- a 2.0) 0.0)] satisfies property.
7           Case [(>= b 0.0), (< (- a 2.0) 0.0)] violates property.
8   NO, it is not weak immune.
9
10  Counterexample for [(>= b 0.0), (< (- a 2.0) 0.0)]:
11          Player A can be harmed if:
12          Player B takes action l_B after history [r_A]
13
14  Weakest Precondition:
15          (and (>= a 2.0) (>= b 0.0))
```

Figure 4.4: CheckMate Output for Analyzing the Weak Immunity of the EFG of Figure 4.3, with Counterexample and Weakest Precondition Generation.

4.2.3 Case Splitting in CheckMate

The case splitting engine takes as input the generated SMT formula ϕ corresponding to the analyzed security property. CheckMate uses Z3 to determine satisfiability of ϕ. If ϕ is satisfiable, CheckMate uses the model satisfying ϕ, which is provided by Z3 and proceeds to the next reasoning engine. Otherwise, Z3 reports an unsat core, a set of constraints that are a sufficient reason why ϕ is unsatisfiable. The case splitting engine uses this unsat core to decide whether unsatisfiability is due to (i) a necessary case split on the utilities' values that has not yet been considered; or (ii) the EFG structure.

If (i), CheckMate creates two new Z3 queries: one where we add the new utility constraint to ϕ, and one with its negation. The property is satisfied only if both queries are satisfiable, which might require case splitting recursively. We record models for each case, again recursively if necessary. If (ii), CheckMate records the current case split as an unsat case together with its unsat core: these are used later for counterexample and precondition generation. If requested by the user, there is also a feature to keep exploring all cases, even after encountering unsatisfiability. This allows us to provide counterexamples to unsat cases or compute weakest preconditions to be further used in redesigning protocols without unintended behavior.

4.2.4 Strategy Extraction

If requested by the user and if the property was satisfied by the current honest history, CheckMate produces explicit strategies as follows. We take as input the list of cases that we divided into in Section 4.2.3, together with their models, and infer the corresponding game-theoretic strategy per case. These strategies provide a witness for the game and its honest history, satisfying the security property. The list of cases with their witness strategies is subsequently provided in the CheckMate output.

4.2.5 Counterexamples

If requested by the user (Section 4.3) and if the honest history violated the security property, CheckMate additionally computes counterexamples as to why the security property was violated. Depending on further flags (Section 4.3), one or all counterexamples for one or all unsat cases are produced. Accordingly, the counterexamples engine receives one or all unsat cases and their unsat cores. For all received cases, we study the unsat core to extract counterexamples.

To compute all counterexamples, we forbid the game choices that led to the found counterexample by adding their negation to the SMT formula ϕ's constraints and checking satisfiability. We then iterate until the extended ϕ satisfies the property. As in previous work [BKK+23a], counterexamples to *practicality* are computed differently, i.e., without the use of unsat core, while following the same iterative procedure to produce all counterexamples.

Lines 10–12 of Figure 4.4 list a counterexample to the *weak immunity* of Figure 4.3. Within a counterexample to weak immunity, CheckMate reports on the harmed player p (line 11 of Figure 4.4) and lists the actions the other players can take to attack p. These actions cannot be prevented by the honest player p, which leads to p being harmed (line 12 of Figure 4.4). In a counterexample to *collusion resilience*, CheckMate provides the group of players that profits from an attack and also lists the attack. An attack is a set of deviating actions the malicious group takes to profit, while the honest players cannot prevent these actions. In a counterexample to *practicality*, CheckMate lists a player p, a rational subhistory r different from the honest one, and the part of the honest history after which p profits from deviating to r.

4.2.6 Weakest Preconditions

To extract the weakest precondition, that – if added to the set of initial constraints – makes the honest history satisfy the security property, all unsat cases have to be computed in the case splitting engine (see Section 4.2.3). This list of unsat cases operates as the input to the weakest preconditions routine. We apply tailored simplification steps to reduce the list of unsat cases to an equivalent and readable formula.

This only applies if the user sets the weakest precondition flag of CheckMate (see Section 4.3) and the analyzed honest history violated the respective security property. In this case, the computed weakest precondition is provided as output. Lines 14–15 of Figure 4.4 list the weakest precondition for the weak immunity of Figure 4.3 and history (r_A, l_B).

4.3 Usage

CheckMate invocations are of the form `checkmate GAME FLAGS`, where `GAME` is an input file as specified in Section 4.2.1 and `FLAGS` are described below. CheckMate accepts the following options to modify its behavior:

-preconditions If a security property is not satisfied, CheckMate computes the weakest precondition, which, if enforced additionally, would satisfy the security property.

-counterexamples If a security property is not satisfied, CheckMate provides a counterexample showing why the property does not hold, i.e., an attack vector. The number of considered scenarios is controlled by the `all_cases` flag.

-all_counterexamples If a security property is not satisfied, CheckMate provides *all* counterexamples for the violated case(s).

-all_cases If a security property is not satisfied, CheckMate computes *all* violated cases.

-strategies If a security property is satisfied, CheckMate provides evidence in the form of a strategy that satisfies it.

Additionally, the user can choose which security properties to analyze with the options `-weak_immunity`, `-weaker_immunity`, `-collusion_resilience`, and also `-practicality`. If no property is specified, then all four of them will be analyzed by default. For instance, to generate the output shown in Figure 4.4, we execute

```
checkmate GAME -weak_immunity -counterexamples -preconditions,
```

where `GAME` is an input file containing the JSON encoding of the game in Figure 4.3.

4.4 Evaluation

We evaluated our tool on 15 benchmarks. Table 4.1 surveys our examples, with its last 7 lines listing new benchmarks compared to [BKK+23a]. Out of our 15 examples, 5 describe blockchain protocols with 2, 3, or 5 players – these are the **Simplified Closing**, **Simplified Routing**, **Closing**, **3-Player Routing** and **Unlocking Routing**. The **Auction** example of Table 4.1 models the economic behaviors of an auction; **Tic Tac Toe** as well as **Tic Tac Toe Concise** a game of tic-tac-toe; whereas the 7 other examples of Table 4.1 are game-theoretic problems with 2 to 4 players. Table 4.1 summarizes our experimental results. CheckMate was run in default mode; that is, none of the flags described in Section 4.3 were set, and all four security properties were analyzed. In each of the terminating benchmarks, the current CheckMate version (version v1), presented in this work, is significantly faster than its initial prototype (version v0) [BKK+23a]. As mentioned in Section 4.1, this is due to our optimized and reshaped C++ implementation, as well as thanks to an improved case splitting algorithm.

Experimental Analysis. Our tool improvements make even the 5-player game **Unlocking Routing**, with 36,113 nodes, feasible to analyze. Our biggest game **Tic Tac**

Toe, on which CheckMate does not terminate within one hour, is modeled in an unnecessarily huge way on purpose to show CheckMate's limitations: the majority of EFG branches could be removed for symmetry reasons. A game-theoretically equivalent pruned game tree is analyzed in benchmark **Tic Tac Toe Concise**, for which CheckMate terminates in 107.84 seconds. Scaling CheckMate further is an interesting challenge for future work.

Of the 7 new benchmarks, only **Tic Tac Toe Concise** was secure for the honest history incorporating the known tie-yielding behavior. For the game-theoretic problems G, **Centipede**, and **EBOS**, none of the security properties hold. This is not surprising as game-theoretic problems often model dilemmas, which by their nature are not secure. Surprisingly, many of the necessary preconditions were not `false`, but rather readable and short. For example, the weakest precondition to make the **EBOS** benchmark satisfy *practicality* is $2d + f \geq p$, where d, f, and p are variables occurring in the utilities of **EBOS**.

Auction violates weak immunity and collusion resilience. The model assigns the auctioneer a negative value in case the item is not being sold, while the bidders get negative values if they do not receive the item; hence, **Auction** cannot be weak immune and is reported as such by CheckMate. Ignoring the inconvenience of not selling the item, respectively not receiving it, **Auction** then becomes weak immune. This ignoring of small negative values is what weaker immunity incorporates [BKK+23a]. Further, the auctioneer and one of the bidders can collude to ensure this one bidder gets the item, which contradicts collusion resilience and is also reported by CheckMate. The weakest precondition to imply security is `false`.

Finally, **Unlocking Routing** violates weak immunity for similar reasons as **Auction**, and thus also satisfies weaker immunity. It is practical, but is not collusion resilient, as it is vulnerable to the known Wormhole attack [RAKM23]. For this benchmark too, the weakest precondition to make it secure is `false`. That means the structure of the protocol has to be changed to enable security, a mere restriction of values is not enough.

4.5 Related Work and Conclusion

We describe the CheckMate tool for automating the security analysis of blockchain protocols. CheckMate complements the state of the art in protocol verification with game-theoretic security analysis, providing economic security guarantees in addition to algorithmic correctness. CheckMate differs from existing static analyzers [Cer, HBS23, OMA+23] of Ethereum smart contracts, as these techniques merely consider cryptographic security and formally verify the Solidity [Aut25] implementation of smart contracts. Formal methods are also used in the cryptographic verification of more general protocols [Bla14, KNT20, MSCB13], yet without game-theoretic considerations. On the other hand, existing game-based analyzers [GHWZ18, KNPS20, MMT05] exhibit stochastic concurrent games and provide probabilistic results about likely behaviors [KNPS20] or apply compositional techniques for simulating game behavior.

Game	Nodes	Players	Histories	Time (v1)	Time (v0)
Splits$_{\text{wi}}$	5	2	3	0.03	0.35
Splits$_{\text{cr}}$	5	2	3	0.03	0.35
Market Entry	5	2	3	0.02	0.28
Simplified Closing	8	2	2	0.02	0.26
Simplified Routing	17	5	1	0.02	0.31
Pirate	52	4	40	1.07	27.08
Closing	221	2	2	0.34	9.60
3-Player Routing	21,688	3	1	6.83	242.54
G **(Figure 4.2)**	5	2	1	0.02	0.18
Centipede	19	3	1	0.07	0.48
EBOS	31	4	1	0.02	0.53
Auction	92	4	1	0.11	1.72
Unlocking Routing	36,113	5	1	10.85	478.58
Tic Tac Toe Concise	58,748	2	1	107.84	254.87
Tic Tac Toe	549,946	2	1	TO	TO

Table 4.1: Results of the current CheckMate (v1) versus its initial prototype (v0) from [BKK+23a]. Runtimes in seconds; timeout (TO) after one hour; using a 12-core AMD Ryzen 9 7900X processor running at 4.7 GHz and 128 GB of DDR5 memory clocked at 4800 MHz.

Unlike [GHWZ18, KNPS20, MMT05], CheckMate supports SMT-based precise reasoning over symbolic utilities without predicting/simulating its EFG properties. Security analysis in CheckMate becomes a theorem-proving task in first-order real arithmetic, for which CheckMate implements novel, SMT-based techniques. With its various features and modes, CheckMate helps blockchain developers not only to analyze their protocols but also to "debug" and revise their protocol modeling and verification tasks. In particular, the counterexamples generated by CheckMate capture attack vectors to be mitigated, whereas the weakest preconditions computed by CheckMate provide constraints to be enforced in the protocols. Our experimental results demonstrate the real-world scalability of CheckMate, verifying, for example, the closing and routing phases of Bitcoin's Lightning Network [PD16].

CHAPTER 5

Compositional Game-Theoretic Security

This chapter is based on article [BKK+25]:

Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. Divide and Conquer: A Compositional Approach to Game-Theoretic Security. EasyChair Preprint no. 15785, 2025.

This article is an extended version of a paper that is currently under review.

5.1 Problem Statement

Decentralized systems based on blockchain technology, such as cryptocurrencies [Nak08] and decentralized finance [Woo14], are in need of security guarantees. Establishing such guarantees is usually approached by formal analysis of the underlying cryptographic protocols [MSCB13, Bla14, WLC+19, KNT20]; or by game-theoretic security analysis [ZBPBS21, RAKM23] to ensure economic incentives in a protocol align with intended outcomes and capture malicious actions preventable by punishment mechanisms within blockchain analysis. This work focuses on game-theoretic security.

Recent work shows that automatic analysis of game trees is tractable via satisfiability modulo theory (SMT) solving in first-order real arithmetic. In particular, game-theoretic analysis is reduced to solving a single large SMT instance [BKK+23a], and this approach scales to mid-size games [RBK+24]. However, this style of automated analysis has inherent limitations. A major problem is *scalability*: full game trees of large protocols are huge, yielding enormous SMT instances that cannot be solved in reasonable time. Another challenge is *game-theoretic modeling*: it is much more convenient to reason about subgames in a modular, independent manner and compose subgames' results into

results over the entire protocol. Such convenience becomes even more pronounced in the presence of repeated subgames.

This work addresses the aforementioned challenges and introduces a *compositional approach to game-theoretic security* (Section 5.5). Given a protocol, we model parts of a protocol independently as subgames, analyze the security of the resulting subgames, and combine subgame securities to enforce security of the game modeling the entire protocol. In other words, we perform a *divide-and-conquer* approach for game-theoretic security analysis, whose automation is feasible via SMT solving (Section 5.6). As the same subgame might occur multiple times in the game tree, the reasoning effort involved is dramatically reduced by compositional game design, thus scaling SMT-based reasoning to large game trees (Section 5.7).

Compositional reasoning is, however, not trivial, as illustrated by Example 5.11, which shows that SMT queries may not be naïvely split into subgames, as constraints in one subgame may interact with constraints in other subgames (Section 5.4). Further, a security result of a subgame cannot be just simply propagated upwards, as in our experiments we have encountered all four possible scenarios: a subgame is not secure, but the entire game is; a subgame is secure, but the entire game is not; both subgame and entire game are secure; and both not secure. Our divide-and-conquer approach provides a theoretically *sound and complete way* to decompose reasoning into fine-grained SMT queries over subgames (Theorem 5.8).

To the best of our knowledge, our approach is the first compositional method for game-theoretic security. We implement our work as the next generation of CheckMate called CheckMate2.0. Our experiments demonstrate that divide-and-conquer reasoning enables game-theoretic modeling and analysis of complex real-world protocols with millions of nodes.

Contributions. We bring the following contributions.

- We introduce a *compositional framework for game-theoretic security analysis* (Section 5.5). Our framework defines player-dependent notions of security properties, which in turn enable divide-and-conquer reasoning over game trees. We divide games into subgames while ensuring that the resulting reasoning is both sound and complete.
- We advocate divide-and-conquer algorithmic reasoning to automate compositional modeling and security analysis (Section 5.6). We interleave subgame and supergame (parent game) reasoning, by using the security result of a subgame within leaves of their respective supergames.
- Our compositional framework naturally supports the *generation of counterexamples* if security properties are violated. Moreover, we *revise game preconditions* in order to strengthen and enforce security. When security is established, we *extract a game strategy* as a proven security certificate (Sections 5.6.3 and 5.6.4).

- We implement compositional game reasoning in the CheckMate2.0 tool. Our experiments show that compositionality significantly improves runtime and supports efficient case-splitting over symbolic game utilities (Section 5.7).

5.2 Preliminaries

We assume familiarity with standard first-order logic [Smu95] and real arithmetic in the context of SMT solving [BT18, BN24]. We next introduce game-theoretic concepts relevant to our work. For readability and to ease reasoning about parts of games, the definitions of extensive form games and their properties from Chapter 3 are slightly reformulated.

A *game* is a static finite object with finitely many *players*. Players choose from a finite set of *actions* until the game ends, whereupon they receive a *utility*. The focus is on perfect information *Extensive Form Games* (EFGs) [OR94] in which the actions are chosen sequentially with full knowledge of all previous actions. Games may yield collective benefit or loss, i.e., they are not necessarily *zero-sum*.

Definition 5.1 (Extensive Form Game — EFG)**.** *An* extensive form game $\Gamma = (N, G)$ *is determined by a finite non-empty set of players* N *together with a finite tree* $G = (V, E)$*. A game path* $h = (e_1, ..., e_n)$*, with* $e_i \in E$*, that starts from the root of* G *is called a* history*. We denote the set of histories* $\mathscr{H}$*. There is a bijection between nodes* $v \in V$ *and histories* $h \in \mathscr{H}$ *that lead to these nodes.*

- *A history that leads to a leaf is called* terminal *and belongs to the set of* terminal histories $\mathscr{T} \subseteq \mathscr{H}$*. Terminal histories* t *are associated with a* utility *for each player.*
- Non-terminal histories *are those histories that are not terminal. Non-terminal histories* h *have assigned a next player denoted as* $P(h) \in N$*. Player* $P(h)$ *chooses from the set* $A(h)$ *of possible actions following* h*.*

In an EFG Γ, we call a terminal history h^* *honest* if it represents the expected behavior in Γ. An EFG Γ can have many honest histories; security analysis over Γ is always performed relative to a chosen and fixed honest history (Section 5.3.1).

Example 5.1 (Market Entry Game)**.** Consider the Market Entry game Γ_{me} of Figure 5.1. In this game, there are two players: M representing a new company and E an established company. At the root, it is the turn of player $P(\emptyset) = M$ to choose from actions $A(\emptyset) = \{n, e\}$. Action n represents *not* entering the market, producing a terminal history (n) where M gets 0 utility and E gets all of the profits $p > 0$. Action e represents entering the market, in which case E can respond by either ignoring this move and thus splitting profits equally, or by entering a price war that damages both players.

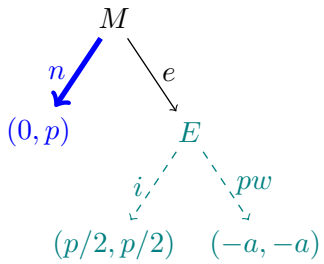


Figure 5.1: Market Entry Game Γ_{me}, with $a, p > 0$. Utility tuples state M's utility first, E's second.

Utilities in game theory are usually numeric constants. We generalize utilities to *symbolic* terms in real arithmetic and thus encode all possible values within given constraints. Variables and numeric constants are evaluated over the real numbers extended by a finite set of *infinitesimals*, closer to zero than any real number. Infinitesimals model subjective (in)conveniences that do not relate directly to funds, such as opportunity cost. We model infinitesimals with terms over $\mathbb{R} \times \mathbb{R}$, ordered lexicographically: the first component represents the real part, the second the infinitesimal. We write real for the first projection and avoid writing pairs, using $a, b, c \ldots$ for real variables, and $\alpha, \beta, \gamma, \ldots$ for infinitesimals. The utility term $a + \alpha - \varepsilon$ is therefore represented as $(a, 0) + (0, \alpha) - (0, \varepsilon) = (a, \alpha - \varepsilon)$.

Example 5.2. We could modify the Market Entry game from Example 5.1 by adding an infinitesimal $\alpha > 0$ to the utility of player M at (e, i). The utility $\frac{p}{2} + \alpha$ represents half of the profit p and the additional benefit of entering the market α, as M is motivated to establish a new entity on the market.

We use the Market Entry game as a running example for its simplicity. Our approach is also evaluated on real-world models in Section 5.7, such as the Closing game, the 3-Player Routing game, or the Auction game. To formulate game-theoretic security properties, we need the following definitions for EFGs.

Definition 5.2 (EFG Properties)**.** *Let $\Gamma = (N, G)$ be an EFG.*

Strategy *A* strategy *σ for a group of players $S \subseteq N$ is a function mapping non-terminal histories $h \in \mathscr{H} \setminus \mathscr{T}$, where one of the players in group S has a turn $P(h) \in S$, to the possible actions $A(h)$. We write $\mathscr{S}_S$ for the set of strategies for group S, and $\mathscr{S}$ for $\mathscr{S}_N$, which we call* joint strategies*. We refer to the union of strategies with disjoint domains as a* combined strategy *and denote it as a tuple. To combine e.g. $\sigma_S \in \mathscr{S}_S$ and $\tau_{N-S} \in \mathscr{S}_{N-S}$, we write $(\sigma_S, \tau_{N-S}) \in \mathscr{S}$*[1]*.*

Resulting History *The* resulting terminal history *$H(\sigma)$ of a joint strategy $\sigma \in \mathscr{S}$ is the unique history obtained by following chosen actions in σ from root to leaf.*

[1]This tuple notation is consistent with the substitution notation in the previous chapters, e.g. $(\sigma_S, \tau_{N-S}) = \tau[\sigma_S/\tau_S]$ but more convenient when it comes to reasoning about parts of games.

Following Honest History *A strategy for a player p follows the honest history h^* if, at every node along the honest history, where p is making a choice, the strategy chooses the action in h^*. For every other node, there is no constraint.*

Utility Function *The* utility function *$u_p(\sigma)$ assigns to player $p \in N$ their utility at the resulting terminal history of the joint strategy $\sigma \in \mathscr{S}$, that is $u_p(\sigma) := u_p(H(\sigma))$. We sometimes write all player utilities for a joint strategy $\sigma \in \mathscr{S}$ as $u(\sigma)$, denoting a tuple of size $|N|$.*

Subgame Subgames $\Gamma_{|h}$ *of Γ are formed from the same set N of players and a subtree of G, and are therefore identified by the history h leading to the subtree $G_{|h}$. Histories $\mathscr{H}_{|h}$ of $\Gamma_{|h}$ are histories in $\mathscr{H}$ with prefix h, strategies $\sigma_{|h} \in \mathscr{S}_{|h}$ of $\Gamma_{|h}$ are strategies restricted to the nodes in $\mathsf{G}_{|h}$ and the utility function $u_{|h}$ of $\Gamma_{|h}$ assigns each joint strategy $\sigma_{|h} \in \mathscr{S}_{|h}$ the utility of the yielded leaf $u_{|h}(\sigma_{|h}) := u(h, H(\sigma_{|h}))$. This includes trivial subgames: leaves or the entire tree Γ at the empty history.*

Supergame *If h' is a prefix of h, $\Gamma_{|h'}$ is a supergame of $\Gamma_{|h}$.*

Subtree along/off Honest History *Let h^* be the honest history. A subgame $\Gamma_{|h}$ is* along *the honest history iff h is a prefix of h^*; that is, there is a history $g \in \mathscr{H}_{|h}$ in the subtree such that $(h, g) = h^*$. Otherwise, $\Gamma_{|h}$ is* off *the honest history.*

Intuitively, a *subgame* is the part of the game that is still to be played after some actions have been taken already. A *supergame* of a subgame is any game tree that embeds the subgame as the subtree. For the sake of readability, we use the following simplifications. We use *subgame/subtree* and *supergame/supertree* interchangeably. We write $u(\sigma_S, \tau_{N-S}) := u((\sigma_S, \tau_{N-S}))$ for the combined strategy $(\sigma_S, \tau_{N-S}) \in \mathscr{S}$. For history $k \in \mathscr{H}$, we write $k_{|h}$ to express the suffix of k after h, that is $(h, k_{|h}) = k$.

Example 5.3. Consider again the Market Entry game Γ_{me} in Figure 5.1. A joint strategy τ could have M taking action n initially, and player E taking i after (e). M's strategy $\tau_M \in \mathscr{S}_M$ takes action n initially. E receives $u_E(\tau) = p$. The history resulting from τ is (n), and τ is a strategy extending history (n).

The subgame for history (e) has players $\{M, E\}$ and a tree where E must choose between action i with utility $(\frac{p}{2}, \frac{p}{2})$ and action pw with utility $(-a, -a)$. For honest history (e, i) the subtree $\Gamma_{me|(e)}$ after action e is *along* the honest history (e, i), whereas the trivial subtree $\Gamma_{me|(n)}$ after action n is *off* the honest history (e, i).

The Market Entry game has $2 \times 2 = 4$ joint strategies as M chooses from two possible actions, and independently E picks one action out of two in the subtree $\Gamma_{me|(e)}$.

5.3 Game-Theoretic Security Properties

Our work models real-life protocols as extensive form games (EFGs). Subsequently, we reduce the security analysis of a protocol to the game-theoretic security analysis of its

corresponding EFG. According to [ZBPBS21] an adversary could execute an attack in a protocol for personal gain or harming somebody. Therefore, we consider a protocol to be *game-theoretically secure* if the following properties hold:

(P1) **Byzantine Fault-Tolerance.** Even in the presence of adversaries, honest players do not suffer loss. That is, in a secure protocol an honest player will not receive negative utility, independent of others' behavior. Therefore, there are no "attacks" where somebody is harmed.

(P2) **Incentive Compatibility.** Rational agents do not deviate from the honest behavior, as it yields the best payoff. Hence, in a secure protocol, a rational "attacker" is behaving honestly and no adversary gets personal gain by deviation.

5.3.1 Security Properties for Subgames

To accommodate a compositional game-theoretic approach (Section 5.5), we define the security properties *weak immunity*, *collusion resilience*, and *practicality* for any subtree of Γ, soundly generalizing their definitions in the previous chapters. Property (P1) is ensured by weak immunity and (P2) by the combination of collusion resilience and practicality. We assume a total order on symbolic utility terms, lifted in Section 5.3.2.

Definition 5.3 (Weak Immunity)**.** *A subtree $\Gamma_{|h}$ of game Γ with honest history h^* is* weak immune*, if a strategy $\sigma \in \mathscr{S}_{|h}$ exists such that all players p following σ always receive non-negative utility:*

$$\exists \sigma \in \mathscr{S}_{|h}. \forall p \in N \; \forall \tau \in \mathscr{S}_{|h}.\; u_p(\sigma_p, \tau_{N-p}) \geq 0 \; . \qquad (\mathrm{wi}(\Gamma_{|h}))$$

If h is along h^*, additionally σ has to follow $h^*_{|h}$, i.e. $H_{|h}(\sigma) = h^*_{|h}$.

Example 5.4. The Market Entry game from Example 5.1 with honest history (n) is weak immune: if M behaves honestly both players get a nonnegative utility; if M deviates via e, player E can choose action i and obtains a positive utility $\frac{p}{2}$.

Sometimes, weak immunity is too restrictive and we take *weaker immunity* to ensure (P1).

Definition 5.4 (Weaker Immunity)**.** *A subtree $\Gamma_{|h}$ of game Γ with honest history h^* is* weaker immune*, if there exists a strategy $\sigma \in \mathscr{S}_{|h}$, such that all players p that follow σ always receive at least a negative infinitesimal:*

$$\exists \sigma \in \mathscr{S}_{|h}. \forall p \in N \; \forall \tau \in \mathscr{S}_{|h}.\; \mathsf{real}(u_p(\sigma_p, \tau_{N-p})) \geq 0 \; . \qquad (\mathrm{weri}(\Gamma_{|h}))$$

If h is along h^*, additionally $H_{|h}(\sigma) = h^*_{|h}$.

Next, the property of collusion resilience requires the honest behavior to yield the best payoff, even in the presence of collusion.

Definition 5.5 (Collusion Resilience)**.** *A subtree $\Gamma_{|h}$ of the game Γ with honest history h^* is* collusion resilient *if there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that no strict subgroup of players can deviate to receive a joint utility greater than their joint honest utility:*

$$\exists \sigma \in \mathscr{S}_{|h}. \forall S \subset N \; \forall \tau \in \mathscr{S}_{|h}. \; \sum_{p \in S} u_p(h^*) \geq \sum_{p \in S} u_p(\sigma_{N-S}, \tau_S) \; . \qquad (\mathrm{cr}(\Gamma_{|h}))$$

If h is along h^*, also $H_{|h}(\sigma) = h^*_{|h}$ has to hold.

Note that the collusion resilience of a subtree according to the above definition depends on the *honest utility*, the utility resulting from the honest history in the entire game Γ. The node containing the honest utility is not necessarily part of the considered subtree.

Example 5.5. Consider again the Market Entry game from Example 5.1 with the honest history (n). This is collusion resilient: we can take actions n for player M and pw for player E. Since it is a two-player game, the colluding group of players can only be a singleton. If M deviates from the honest behavior, they get utility $-a$, which is less than 0 in the honest case. If E deviates, the history is not affected, since M chooses action n. Thus, the game is collusion resilient.

The next property of practicality ensures that, for all player decisions, the honest behavior is also "greedy": if all players act selfishly, that is they maximize their own utilities, the honest choice yields the best utility.

Definition 5.6 (Practicality)**.** *A subtree $\Gamma_{|h}$ of the game Γ with honest history h^* is* practical*, if there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that no player can deviate in any subtree to receive a strictly greater utility in the subtree:*

$$\exists \sigma \in \mathscr{S}_{|h} \; \forall g \in \mathscr{H}_{|h} \; \forall p \in N \; \forall \tau \in \mathscr{S}_{|(h,g)}. \qquad (\mathrm{pr}(\Gamma_{|h}))$$
$$u_{|g,p}(\sigma_{|g}) \geq u_{|g,p}(\tau_p, \sigma_{|g,N-p}) \; .$$

If h is along h^*, also $H_{|h}(\sigma) = h^*_{|h}$ has to hold.

Example 5.6. The Market Entry game from Example 5.1 with the honest history (n) is not practical. Player E should choose i, as it yields a better utility. It is then not practical for M to choose n, as it yields utility 0, whereas action e yields the better utility $\frac{p}{2}$.

We finally note that every subtree $\Gamma_{|h}$ of a game Γ that is off the honest history is always practical.

Lemma 5.1 (Subtrees off Honest History are Practical)**.** *Every subtree $\Gamma_{|h}$ of a game Γ that is off the honest history is* practical*.*

Proof. We fix an arbitrary subtree $\Gamma_{|h}$ not along the honest history for which we prove practicality. We construct a strategy $\sigma \in \mathscr{S}_{|h}$ in the following inductive way bottom-up: We consider histories k in $\Gamma_{|h}$, such that for all possible actions $a \in A_{|h}(k)$ after k the strategy σ has been defined already for the subtree after (k, a). Initially, this is only the case for histories k, where all actions $a \in A_{|h}(k)$ lead to leaves.

At each such history k we compare the utilities for the current player $P(k)$ after the possible choices $a \in A_{|h}(k)$ according to σ: $u_{|(h,k,a),P(k)}(\sigma_{|(k,a)})$. We define strategy σ to pick an action $a' = \sigma(k) \in A_{|h}(k)$ which maximizes $P(k)$'s utility. If this is done iteratively, one eventually reaches the root of $\Gamma_{|h}$, at which point the strategy $\sigma \in \mathscr{S}_{|h}$ is fully defined.

We now show σ is practical for the fixed case split. According to Definition 5.6, we pick an arbitrary history $g \in \mathscr{H}_{|h}$, a player $p \in N$ and a strategy $\tau \in \mathscr{S}_{|(h,g)}$ in the subgame after g, and prove:

$$u_{|(h,g),p}(\sigma_{|g}) \geq u_{|(h,g),p}(\tau_p, \sigma_{|g,N-p}) \,. \tag{5.1}$$

If τ_p and $\sigma_{|g,p}$ are identical, the inequality holds trivially, hence we assume them to be distinct. The strategy $\sigma_{|g}$ generates a history which we call $h_\sigma = H(\sigma_{|g})$ and $H(\tau_p, \sigma_{|g,N-p}) = h_\tau$. Note that the strategies differ only in choices player p made. We will now use induction on the deviation points ℓ_n of σ along h_τ to show Equation (5.1). Our induction hypothesis is

$$u_{|(h,g,\ell_n),p}(\sigma_{|(g,\ell_n)}) \geq u_{|(h,g,\ell_n),p}(\tau_{|\ell_n,p}, \sigma_{|(g,\ell_n),N-p}) \,. \tag{5.2}$$

For the base case, we consider the last point along the generated history h_τ, where the choice taken in strategy $(\tau_p, \sigma_{|g,N-p})$ differs from the one taken in $\sigma_{|g}$. We call the history leading to this point $\ell_1 \in \mathscr{H}_{|(h,g)}$. The player at ℓ_1 has to be p. Revisiting now the construction of σ, at ℓ_1 we chose $\sigma_{|g}(\ell_1) =: a_1$ to maximize the utility of $P(\ell_1) = p$. Therefore, we know for $c_1 := \tau(\ell_1)$

$$\begin{aligned} u_{|(h,g,\ell_1,a_1),p}(\sigma_{|(g,\ell_1,a_1)}) &\geq u_{|(h,g,\ell_1,c_1),p}(\sigma_{|(g,\ell_1,c_1)}) \\ &= u_{|(h,g,\ell_1,c_1),p}(\tau_{|(\ell_1,c_1),p}, \sigma_{|(g,\ell_1,c_1),N-p}). \end{aligned}$$

The equality holds, because σ and (τ_p, σ_{N-p}) are identical on $\Gamma_{|(h,g,\ell_1,c_1)}$. By definition of a_1 and c_1 we also know

$$u_{|(h,g,\ell_1),p}(\sigma_{|(g,\ell_1)}) \geq u_{|(h,g,\ell_1),p}(\tau_{|\ell_1,p}, \sigma_{|(g,\ell_1),N-p}) \,, \tag{5.3}$$

which concludes the base case.

For the inductive case, we assume the induction hypothesis Equation (5.2), for the previous deviation point ℓ_{n-1} and consider deviation point ℓ_n along h_τ. We define $c_n := \tau(\ell_n)$ an $a_n := \sigma_{|g}(\ell_n)$. By the definition of σ, we have

$$u_{|(h,g,\ell_n,a_n),p}(\sigma_{|(g,\ell_n,a_n)}) \geq u_{|(h,g,\ell_n,c_n),p}(\sigma_{|(g,\ell_n,c_n)}) \,. \tag{5.4}$$

We further know that

$$u_{|(h,g,\ell_n,c_n),p}(\sigma_{|(g,\ell_n,c_n)}) = u_{|(h,g,\ell_{n-1}),p}(\sigma_{|(g,\ell_{n-1})}) \ . \tag{5.5}$$

This is the case, since by definition of ℓ_n and ℓ_{n-1} as subsequent deviation points, the history of $\sigma_{|g}$ between (ℓ_n, c_n) and ℓ_{n-1} is identical to the one of $(\tau_p, \sigma)_{|g,N-p}$. Combining now Equation (5.4), Equation (5.5) and the inductive hypothesis we derive

$$u_{|(h,g,\ell_n,a_n),p}(\sigma_{|(g,\ell_n,a_n)}) \geq u_{|(h,g,\ell_{n-1}),p}(\tau_{|l_{n-1},p}, \sigma_{|(g,\ell_{n-1}),N-p}) \ . \tag{5.6}$$

By the definition of a_n (for the left-hand side of the equation) and the fact that ℓ_n and ℓ_{n-1} are along h_τ (for the right-hand side), we conclude that Equation (5.2) holds also for ℓ_n.

To finalize the proof, we consider the first point ℓ_M along h_τ, where σ_g and $(\tau_p, \sigma_{|g,N-p})$ differ. This is exactly the splitting point of h_σ and h_τ, thus

$$u_{|(h,g),p}(\sigma_{|g}) = u_{|(h,g,\ell_M),p}(\sigma_{|(g,\ell_M)}) \text{ , and} \tag{5.7}$$

$$u_{|(h,g),p}(\tau_p, \sigma_{|g,N-p}) = u_{|(h,g,\ell_M),p}(\tau_{|l_M,p}, \sigma_{|(g,\ell_M),N-p}) \ . \tag{5.8}$$

From this, together with Equation (5.2), Equation (5.1) follows, which shows that σ is indeed practical. Hence, we proved that any subtree not along the honest history is practical. □

Example 5.7. Consider the Market Entry subgame after the non-terminal history (e), marked by teal dashed lines in Figure 5.1. We can always choose the action that yields the best utility for the current player E. The only way we can violate practicality is by having the best choice conflicting with the honest choice, which cannot happen when the subtree is off honest history.

The terms weak(er) immunity, collusion resilience, and practicality can be extended to strategies and terminal histories in the following way.

Definition 5.7 (Security Properties of Strategies and Histories)**.** *A* strategy $\sigma \in \mathscr{S}$ *of a game* Γ *weak(er) immune, collusion resilient, or practical, iff it can serve as the witness strategy in* $(\mathrm{wi}(\Gamma_{|h}))$, $(\mathrm{weri}(\Gamma_{|h}))$, $(\mathrm{cr}(\Gamma_{|h}))$, *or* $(\mathrm{pr}(\Gamma_{|h}))$, *respectively.*

A terminal history $t \in \mathscr{T}$ *of the game* Γ *is weak(er) immune, collusion resilient, or practical, iff there exists a strategy* $\sigma \in \mathscr{S}$ *that has the respective property and extends history* t*, that is* $H(\sigma) = t$*.*

5.3.2 Total Orders

Similarly to [BKK+23a], in order to lift the assumption that we know how all utility terms relate, we make the security analysis relative to a finite set C of *initial constraints*

on the symbolic variables appearing in the utility terms and explicitly universally quantify over the variables, as follows

$$\forall \vec{x}. \left(\bigwedge_{c \in C} c[\vec{x}] \right) \rightarrow \exists \sigma \in \mathscr{S}. H(\sigma) = h^* \wedge sp(\sigma)[\vec{x}] \,, \tag{5.9}$$

where $\vec{x} = (x_1, \ldots, x_\ell)$ are the real variables occurring in the utility terms T_u and $sp(\sigma)$ is a formula pertaining to the security property $sp \in \{wi, weri, cr, pr\}$ after existential quantification of the strategy. For weak immunity $wi(\sigma) = \forall p \in N \; \forall \tau \in \mathscr{S}_{|h}. \, u_p(\sigma_p, \tau_{N-p}) \geq 0$, and similarly for the other properties.

Furthermore, to efficiently handle the comparison of symbolic utilities in an SMT solver, we implement an equivalent version of the above formula by considering all consistent *total orders* $\preceq$ over the set T_u of utility terms appearing in the game tree Γ.

Theorem 5.1 (Game-Theoretic Security with Total Orders)**.** *For an EFG Γ with honest history h^* and a finite set of initial constraints C, property* (5.9) *is equivalent to*

$$\forall (\preceq, T_u) \; \exists \sigma \in \mathscr{S}. H(\sigma) = h^* \; \wedge \; \forall \vec{x}. \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \rightarrow sp(\sigma)[\vec{x}] \,. \tag{5.10}$$

Proof. To show implication "$\Leftarrow$", we pick arbitrary values for the variables $\vec{x}$ that satisfy all initial constraints in C. We then consider the total order $\preceq$ on T_u that is consistent with the choice of values for variables $\vec{x}$. By Equation (5.10), there has to exist a strategy σ yielding the honest history such that for all $\vec{x}$ consistent with $\preceq$ and satisfying the initial constraints C the security property $sp(\sigma)[\vec{x}]$ holds. Since the picked $\vec{x}$ satisfies $\preceq$ and C, the strategy σ works. Hence, 5.9 is implied.

For implication "$\Rightarrow$", let $\preceq$ be an arbitrary total order over the symbolic terms T_u and $\vec{x}$ values that satisfy $\preceq$ as well as C. Then, from (5.9) there exists an honest strategy σ such that $sp(\sigma)[\vec{x}]$. Note that whether $sp(\sigma)[\vec{x}]$ holds depends only on the relation of terms in T_u and not on the actual values of $\vec{x}$. Therefore, $sp(\sigma)[\vec{x}]$ is true for one $\vec{x}$ iff it is true for all $\vec{x}$ consistent with the same term order $\preceq$. Thus, 5.10 holds. □

5.3.3 Counterexamples

If there is no joint strategy satisfying a security property (wi, weri, cr, or pr), we can investigate why *not*. Counterexamples serve the important purpose of providing attack vectors and thus pinpointing weaknesses of a protocol underlying the game model.

Counterexamples to Weak(er) Immunity. For the weak(er) immunity property, a counterexample is a harmed honest player p and a partial strategy of the other players $N - p$ such that no matter what honest actions p chooses, the other players cannot avoid receiving a real-valued negative utility.

Definition 5.8 (Counterexamples to Weak(er) Immunity)**.** *Let* Γ *be an EFG and* h^* *the considered honest history. A* counterexample to h^* being weak(er) immune *is a player* p *together with a partial strategy* s_{N-p} *such that* s_{N-p} *extended by any strategy* σ_p *of player* p *who follows the honest history* h^**, yields a terminal history* $H(s_{N-p}, \sigma_p) = t_{\sigma_p}$ *with* $u_p(t_{\sigma_p}) < 0$ *(resp. for weaker immunity* $\mathsf{real}(u_p(t_{\sigma_p})) < 0$*) and it is* minimal *with that property.*

Minimality of the partial strategy s_{N-p} states that, if any information point $s_{N-p}(h) = a$ is removed, there exists a strategy σ_p of player p such that (σ_p, s'_{N-p}) does not yield a terminal history, where s'_{N-p} is s_{N-p} without action a. That is, when following only actions of (σ_p, s'_{N-p}), we get stuck at an internal node of the tree.

Example 5.8. A counterexample to the weak immunity of the Market Entry game of Example 5.1 with the honest history (e, i) would be player M and a partial strategy for E, where they choose action pw. If M behaves honestly and chooses action e, they end up with the negative utility of $-a$ after the terminal history (e, pw).

Counterexample to Collusion Resilience. A counterexample to collusion resilience (Definition 5.9) consists of a group of deviating players S and their partial strategy $s_S \in \mathscr{S}$, such that the joint utility of S is better than the honest utility, no matter how the other players $N - S$ react, while still following the honest history.

Definition 5.9 (Counterexamples to Collusion Resilience)**.** *Let* Γ *be an EFG and* h^* *the considered honest history. A* counterexample to h^* being collusion resilient *is a set of deviating players* S *together with their strategy* s_S *such that* s_S *extended by any strategy* σ_{N-S} *of players* $N - S$*, which follows the honest history* h^**, yields a terminal history* $H(\sigma_{N-S}, s_S) = t_{\sigma_{N-S}}$ *with*

$$\sum_{p \in S} u_p(t_{\sigma_{N-S}}) > \sum_{p \in S} u_p(h^*)$$

and it is minimal with that property. The minimality of s_S *is similar to the minimality of the partial strategy for weak(er) immunity.*

Example 5.9. In the Market Entry game of Example 5.1, a counterexample to the honest history (e, pw) being collusion resilient is a deviating group $\{E\}$ with a partial strategy that takes action i. Since the honest player M can only take action e, the deviating utility for E is $\frac{p}{2}$, which is greater than the honest one $-a$.

Counterexamples to Practicality. Intuitively, a counterexample to practicality of the honest history h^* has to provide a reason why a rational player would not follow h^*. At some point along h^* after a prefix h, there is an action a promising the current player $P(h)$ a strictly better utility than h^*. Further, in the subgame $\Gamma_{|(h,a)}$ after (h, a) all practical utilities have to be better for $P(h)$, otherwise other players could choose actions in $\Gamma_{|(h,a)}$ that would disincentivize $P(h)$ to deviate from h^*.

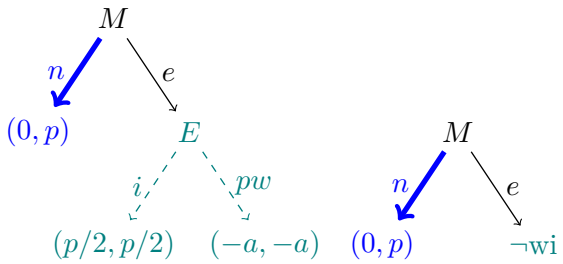


Figure 5.2: Naive Compositionality of Weak Immunity for Market Entry Game, $a, p > 0$.

Definition 5.10 (Counterexamples to Practicality)**.** *For an EFG* Γ *and honest history* h^**, a* counterexample to practicality of h^* *is a prefix* h *of* h^* *together with an action* $a \in A(h)$*, such that for all practical terminal histories* t *in the subgame* $\Gamma_{|(h,a)}$ *it holds that* $u_{P(h)}(h^*) < u_{P(h)}((h, a, t))$.

Example 5.10. Recall that the Market Entry game from Example 5.1 with the honest history (n) is not practical. A counterexample to practicality is the empty prefix $h = \emptyset$ and the action e, as the practical utility in the subgame after history (e) yields $\frac{p}{2}$ for player M, which is strictly better than the 0 in the honest case.

5.4 Unsound Naïve Approach to Compositionality

For a divide-and-conquer style of compositional game-theoretic security analysis, we would like to analyze a game tree by propagating security results of subtrees upwards to the parent/ancestor nodes of the supertree. However, naïvely propagating the yes/no security result of the subtree does not suffice, as shown in Example 5.11.

Example 5.11. Consider the Market Entry game (Example 5.1) reproduced on the left-hand side of Figure 5.2, with honest history (n). Example 5.6 shows this game is weak immune. Now consider a naïve compositional approach looking at the subgame after non-terminal history (e), marked by teal dashed lines. Since player E can take action pw — leading to negative utility for M — this subtree is not weak immune. To mimic a naïve compositionality approach, we replace the subtree after (e) by ¬wi, shown on the right. Asked whether this *supertree* is weak immune, one would say no, as M could deviate from the honest history via e, which leads to a subtree that is not weak immune. This is an incorrect conclusion since the Market Entry game is weak immune for the honest history (n).

The main reason why the naïve approach above fails is that we need more information to be able to propagate a result from a subtree to its parent, namely that the subtree is not weak immune *only for player* M. In the parent, player M can achieve weak immunity by behaving honestly and choosing action n, ensuring weak immunity of the entire game tree. Similar additional information (see Theorem 5.3) is needed for the

other security properties: collusion resilience requires which colluding groups the subtree is secure against; practicality requires the practical utilities resulting from the subtree, and whether the utility of the honest history is practical. We now show that propagating this information yields a sound and complete compositional approach to game-theoretic security.

5.5 Compositional Game-Theoretic Security

Our compositional framework for game-theoretic security analyses is materialized via two crucial components:

1. Stratified analysis of security properties over players, capturing player-wise security properties (Section 5.5.1)
2. Splitting player-wise security properties into subgames, enabling us to propagate subgame reasoning for deriving supergame security (Section 5.5.2).

For simplicity, we assume a total order $\preceq$ on the occurring utility terms T_u in order to relate symbolic game utilities. As before, this assumption is relaxed in Section 5.6, generalizing our approach.

5.5.1 Security Properties Stratified over Players

We start with the following observation. While Example 5.11 shows that there are no implications of subtree and supertree results in general, subtrees *along the honest history* can, in fact, soundly pass negative (not secure) results up to their parents.

Theorem 5.2 (Equivalence of Non-Secure Games)**.** *A game Γ with honest history h^* violates one of the security properties of weak(er) immunity, collusion resilience, or practicality iff there exists a history h along the honest history h^* such that $\Gamma_{|h}$ violates the respective security property.*

Proof. The direction "Γ violates security property $\Rightarrow \exists h \in \mathscr{H}.\ h$ along $h^* \wedge \Gamma_{|h}$ violates security property" is easy to show. Just pick the trivial history $h = \emptyset$, which is always along the honest history.

For the other direction, we consider the security properties individually. We start with *weak immunity* and assume that $\Gamma_{|h}$ is along the honest history h^* and is not weak immune. We fix an arbitrary strategy $\sigma \in \mathscr{S}$ in the entire game Γ that yields the honest history $H(\sigma) = h^*$ and show it is not weak immune. To do so, we consider $\sigma_{|h} \in \mathscr{S}_{|h}$ which still yields the honest history in the subgame: $H_{|h}(\sigma_{|h}) = h^*_{|h}$. Since $\Gamma_{|h}$ is not weak immune, there exists a player $p \in N$ and a strategy $\tau^h \in \mathscr{S}_{|h}$ such that $u_{|h,p}(\tau^h_{N-p}, \sigma_{|h,p}) < 0$. We now construct a strategy $\tau \in \mathscr{S}$ extending τ^h. I.e. $\tau_{|h} = \tau^h$. Everywhere else, we set τ to be identical to σ.

Since σ yields the honest history, $H(\sigma) = h^*$, and $\Gamma_{|h}$ lies along the honest history, we get $H(\tau_{N-p}, \sigma_p) = (h, H_{|h}(\tau_{|h,N-p}, \sigma_{|h,p}))$. By construction of τ also $H(\tau_{N-p}, \sigma_p) = (h, H_{|h}(\tau^h_{N-p}, \sigma_{|h,p}))$ holds. Thus, their utilities have to be identical $u_p(\tau_{N-p}, \sigma_p) = u_{|h,p}(\tau^h_{N-p}, \sigma_{|h,p}) < 0$. Hence, σ is not weak immune and as it was chosen arbitrarily, it follows that Γ with honest history h^* is not weak immune. The proof for *weaker immunity* is analog.

The proof of the second direction for *collusion resilience* follows the same idea. We pick an arbitrary $\sigma \in \mathscr{S}$ yielding the honest history h^* and show it is not collusion resilient, assuming that $\Gamma_{|h}$ is along the honest history and is not collusion resilient. Hence there exists a deviating group $S \subset N$ and a strategy $\tau^h \in \mathscr{S}_{|h}$ such that $\sum_{p\in S} u_{|h,p}(\sigma_{|h}) < \sum_{p\in S} u_{|h,p}(\sigma_{|h,N-S}, \tau^h_S)$. We again construct a strategy $\tau \in \mathscr{S}$ extending τ^h: $\tau_{|h} = \tau^h$. Everywhere else τ is identical to σ. Applying the same reasoning as before, we know that $H(\sigma_{N-S}, \tau_S) = (h, H_{|h}(\sigma_{|h,N-S}, \tau^h_S))$ as well as $H(\sigma) = (h, H(\sigma_{|h}))$, which implies

$$\begin{aligned} \sum_{p\in S} u_p(\sigma) &= \sum_{p\in S} u_{|h,p}(\sigma_{|h}) \\ < \sum_{p\in S} u_{|h,p}(\sigma_{|h,N-S}, \tau^h_S) &= \sum_{p\in S} u_p(\sigma_{N-S}, \tau_S)\ . \end{aligned} \tag{5.11}$$

Therefore, σ is not collusion resilient, and since it was chosen arbitrarily, it follows that Γ with honest history h^* is not collusion resilient.

Proving this for *practicality* is straightforward: we again pick an arbitrary honest strategy $\sigma \in \mathscr{H}$ and assume $h \in \mathscr{H}$ is along the honest history and $\Gamma_{|h}$ is not practical. I.e. there exists a history $k \in \mathscr{H}_{|h}$, a player $p \in N$ and a strategy $\tau \in \mathscr{S}_{|(h,k)}$ such that $u_{|(h,k),p}(\sigma_{|(h,k)}) < u_{|(h,k),p}(\tau_p, \sigma_{|(h,k),N-p})$. Using now $\ell := (h, k) \in \mathscr{H}$, the player p and τ, one can see easily that

$$u_{|\ell,p}(\sigma_{|\ell}) < u_{|\ell,p}(\tau_p, \sigma_{|\ell,N-p})$$

and therefore σ is not practical and neither is Γ. □

Intuitively, the honest history h^* "enforces" a path down the tree Γ: when a non-secure subtree $\Gamma_{|h}$ is encountered along this path, there is no way to compensate for it. Theorem 5.2, however, only propagates non-secure properties along the honest history. To allow for analysis results propagating from subgames to supergames, we *stratify game-theoretic security analysis over individual players*. This means we can analyze the security properties for a player (weak immunity: Definition 5.11, practicality: Definition 5.13), or player group (collusion resilience: Definition 5.12) at a time, without interfering with results of other players or groups (Theorem 5.3).

Definition 5.11 (Weak Immunity for a Player)**.** *A subgame $\Gamma_{|h}$ with honest history h^* is* weak immune *for player $p \in N$, if there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that no matter to which strategy $\tau \in \mathscr{S}_{|h}$ other players deviate, p's utility will be non-negative and, if h*

is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:

$$\exists \sigma \in \mathscr{S}_{|h}.\ (h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \wedge \qquad (\mathrm{wi}_p(\Gamma_{|h}))$$
$$\forall \tau \in \mathscr{S}_{|h}.\ u_{|h,p}(\sigma_p, \tau_{N-p}) \geq 0\ .$$

An analogous definition applies to *weaker immunity.*

Example 5.12 (Player-Wise Weak Immunity)**.** Let us revisit the Market Entry game Γ_{me} with honest history (n) from Example 5.1, considering one player at a time.

The first player is M. The subgame $\Gamma_{me|(e)}$ after history (e) is not weak immune for M, since E could take action pw. Propagating this result to the supertree Γ_{me}, we report weak immunity for M: as M will honestly take action n, we avoid $\Gamma_{me|(e)}$.

For E, $\Gamma_{me|(e)}$ is weak immune as action i can always be chosen, yielding positive utility. Propagating this result, we conclude that Γ_{me} is weak immune for E: all choices of M (whom we do not assume to be honest in the analysis of E), lead to either non-negative utility for E or to a subtree which is weak immune for E.

The definition for collusion resilience *against* a given player group is similar to Definition 5.11, by lifting the quantifier over the player subgroups $S \subset N$ to the front of the formula.

Definition 5.12 (Collusion Resilience against a Player Group)**.** *A subgame $\Gamma_{|h}$ of game Γ with honest history h^* is* collusion resilient against a group of players *$S \subset N$, if there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that no matter to which strategy $\tau \in \mathscr{S}_{|h}$ the players in S deviate, their joint utility will be not greater than their honest joint utility and, if h is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:*

$$\exists \sigma \in \mathscr{S}_{|h}.\ (h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \wedge \qquad (\mathrm{cr}_S(\Gamma_{|h}))$$
$$\forall \tau \in \mathscr{S}_{|h}.\ \sum_{p \in S} u_{|h,p}(\sigma) \geq u_{|h,p}(\tau_S, \sigma_{N-S})\ .$$

Defining practicality for a single player, though, requires slight changes: instead of considering an arbitrary player p, we define practicality for that player whose turn it is in the considered subtree.

Definition 5.13 (Practicality for the Current Player)**.** *A subgame $\Gamma_{|h}$ of a game Γ with honest history h^* is* practical for the current player*, if there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that in each subtree $\Gamma_{|(h,g)}$ no matter to which strategy $\tau \in \mathscr{S}_{|(h,g)}$ the current player $P(h,g)$ deviates, the utility of $P(h,g)$ in the subtree will not increase strictly and, if h is along h^*, then also $H_{|h}(\sigma) = h^*_{|h}$:*

$$\exists \sigma \in \mathscr{S}_{|h}. \qquad (\mathrm{pr}_P(\Gamma_{|h}))$$
$$(h \text{ along } h^* \rightarrow H_{|h}(\sigma) = h^*_{|h}) \wedge \forall g \in \mathscr{H}_{|h}\ \forall \tau \in \mathscr{S}_{|(h,g)}.$$
$$u_{|(h,g),P(h,g)}(\sigma_{|g}) \geq u_{|(h,g),P(h,g)}(\tau_{P(h,g)}, \sigma_{|g,N-P(h,g)})\ .$$

We now state our first crucial result towards compositionality: stratification of security analysis over players.

Theorem 5.3 (Player-Wise Security Properties)**.** *A game* Γ *satisfies a security property iff it satisfies the respective security property player-wise. That is, the following equivalences hold:*

1. Γ *weak immune* $\Leftrightarrow \forall p \in N.\ \Gamma$ *weak immune for* p*.*
2. Γ *weaker immune* $\Leftrightarrow \forall p \in N.\ \Gamma$ *weaker immune for* p*.*
3. Γ *collusion resilient* $\Leftrightarrow \forall S \subset N.\ \Gamma$ *collusion resilient against* S*.*
4. Γ *practical* $\Leftrightarrow\ \Gamma$ *practical for the current player.*

Proof of Theorem 5.3.1-2. We start with equivalence (1) for weak immunity.

Implication " $\Rightarrow$ ". By Definition 5.3 the game Γ, with honest history h^*, is weak immune if

$$\exists \sigma \in \mathscr{S}.\ H(\sigma) = h^* \ \wedge\ \forall p \in N\ \forall \tau \in \mathscr{S}.\ u_p(\sigma_p, \tau_{N-p}) \geq 0\ . \tag{wi}$$

Assuming Γ is weak immune, we consider such a strategy and call it σ'. The right-hand side of the equivalence is

$$\forall p \in N\ \exists \sigma \in \mathscr{S}.\ H(\sigma) = h^* \ \wedge\ \forall \tau \in \mathscr{S}.\ u_p(\sigma_p, \tau_{N-p}) \geq 0\ . \tag{$wi_{\forall p}$}$$

Therefore, we can just pick σ' for all the players and be done.

Implication " $\Leftarrow$ ". We assume Equation ($wi_{\forall p}$), i.e. Γ is weak immune for all players $p_i \in N$, $i = 1, \ldots, n$, where $n = |N|$. Let their corresponding weak immune for p_i strategies be $\sigma^i \in \mathscr{S}$, for $i = 1, \ldots, n$. Further let $\sigma \in \mathscr{S}$ be a new strategy constructed from the σ^i in the following way $\sigma := (\sigma^1_{p_1}, \ldots, \sigma^n_{p_n})$. That means at a history h where it is player p_i's turn, the choice in σ is the one from σ^i: $\sigma(h) = \sigma^i(h)$. The strategy σ yields the honest history $H(\sigma) = h^*$, since all σ^i yield h^* and combining strategies that extend the same history leads to the new strategy extending the same one.

It remains to show that σ is weak immune, i.e. that this one strategy works for all the players (right conjunct in Equation (wi)). We therefore consider an arbitrary player $p_i \in N$ and an arbitrary strategy $\tau \in \mathscr{S}$. By construction of σ and the fact that σ^i is weak immune for p_i, it follows $u_{p_i}(\sigma_{p_i}, \tau_{N-p_i}) = u_{p_i}(\sigma^i_{p_i}, \tau_{N-p_i}) \geq 0$. This concludes the proof that σ is weak immune, which implies that Γ is weak immune.

The proof of Theorem 5.3.2 is analog. □

Proof of Theorem 5.3.3. Implication " ⇒ ". For readability, let us restate the formula for collusion resilience of Γ

$$\exists \sigma \in \mathscr{S}.\, H(\sigma) = h^* \ \wedge\ \forall S \subset N\ \forall \tau \in \mathscr{S}.\\ \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\tau_S, \sigma_{N-S})\ , \tag{cr}$$

as well as for collusion resilience against all subgroups of the game Γ

$$\forall S \subset N\ \exists \sigma \in \mathscr{S}.\, H(\sigma) = h^* \ \wedge\ \forall \tau \in \mathscr{S}.\\ \sum_{p \in S} u_p(\sigma) \geq \sum_{p \in S} u_p(\tau_S, \sigma_{N-S}). \tag{cr$_{\forall S}$}$$

The implication then again follows from the definitions *cr* and $cr_{\forall S}$. The one strategy σ in *cr* can be used for all $S \subset N$ in $cr_{\forall S}$.

Implication " ⇐ ". We prove this direction by contraposition, showing $\neg cr \Rightarrow \neg cr_{\forall S}$. In [BKK+23a], it is shown that $\neg cr$ is equivalent to the existence of a counterexample (Definition 5.9): There exists a set of deviating players S together with their strategy s_S such that s_S extended by any strategy σ'_{N-S} of players $N - S$, which follows the honest history h^*, yields a terminal history $H(\sigma'_{N-S}, s_S) = t_{\sigma'_{N-S}}$ with

$$\sum_{p \in S} u_p(t_{\sigma'_{N-S}}) > \sum_{p \in S} u_p(h^*) \tag{5.12}$$

Let a group of deviating players $S \subset N$ and their strategy s_S be a counterexample. To show $\neg cr_{\forall S}$ we fix an arbitrary $\sigma \in \mathscr{S}$ that yields the honest history h^*. For the strategy $\tau = (s_S, \sigma_{N-S})$ we have by Equation (5.12):

$$\sum_{p \in S} u_p(\sigma) = \sum_{p \in S} u_p(h^*)\\ < \sum_{p \in S} u_p(H(\sigma_{N-S}, s_S)) = \sum_{p \in S} u_p(\tau_S, \sigma_{N-S}).$$

Therefore, $\neg cr_{\forall S}$ holds and implication "⇐" is proven.

□

Proof of Theorem 5.3.4. Implication " ⇒ ". We again restate the formulae for better readability. Practicality of Γ:

$$\exists \sigma \in \mathscr{S}.\, H(\sigma) = h^* \ \wedge\ \forall h \in \mathscr{H}\ \forall p \in N\ \forall \tau \in \mathscr{S}_{|h}.\\ u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h,N-p}, \tau_p) \tag{pr}$$

and practicality for the current player of Γ:

$$\exists \sigma \in \mathscr{S}.\, H(\sigma) = h^* \ \wedge \forall h \in \mathscr{H}\ \forall \tau \in \mathscr{S}_{|h}.\\ u_{|h,P(h)}(\sigma_{|h}) \geq u_{|h,P(h)}(\sigma_{|h,N-P(h)}, \tau_{P(h)})\ . \tag{pr$_{P(h)}$}$$

It is now easy to see that pr implies $pr_{P(h)}$, since for all $h \in \mathscr{H}$, $P(h) \in N$.

Implication " $\Leftarrow$ ". Assuming Equation $(pr_{P(h)})$ holds, we fix σ that satisfies it and show σ also satisfies Equation (pr). Let $h \in \mathscr{H}$, $p \in N$ and $\tau \in \mathscr{S}_{|h}$ as in (pr) be arbitrary.

If $p = P(h)$, then the inequality $u_{|h,p}(\sigma_{|h}) \geq u_{|h,p}(\sigma_{|h,N-p}, \tau_p)$ is immediately implied by $pr_{P(h)}$. If $p \neq P(h)$, we consider the first point in $H_{|h}(\sigma_{|h})$ where it is player p's turn and call the corresponding history $t \in \mathscr{H}_{|h}$. In case no such t exists, then $H_{|h}(\sigma_{|h}) = H_{|h}(\sigma_{|h,N-p}, \tau_p)$, and hence $u_{|h,p}(\sigma_{|h}) = u_{|h,p}(\sigma_{|h,N-p}, \tau_p)$.

If such a t exists, we know from $(pr_{P(h)})$ that $u_{|(h,t),p}(\sigma_{|(h,t)}) \geq u_{|(h,t),p}(\sigma_{|(h,t),N-p}, \tau_{|t,p})$, since $p = P(h,t)$. From the fact that t lies along $H_{|h}(\sigma_{|h})$ follows $u_{|(h,t),p}(\sigma_{|(h,t)}) = u_{|h,p}(\sigma_{|h})$, and as player p has no choices in t, it follows

$$H_{|h}(\sigma_{|h,N-p}, \tau_p) = (t, H_{|(h,t)}(\sigma_{|(h,t),N-p}, \tau_{|t,p}))$$

which leads to identical utilities. Therefore, the inequality in pr is also proven for $p \neq P(h)$. This concludes the proof of the theorem, as h, p, and τ were chosen arbitrarily. □

5.5.2 Splitting and Combining Player-Wise Security Properties

Theorem 5.3 proves that the security analysis of a game can be carried out player-wise, instead of analyzing interactions between all (groups of) players. We now show that not only can players be treated individually, but *(super)game security can also be split into subgame security.* That is, the security of a supergame can be proven by proving player-wise security over subgames. This implies compositional game-theoretic security is sound and complete.

Theorem 5.4 (Compositional Game-Theoretic Security)**.** *The game-theoretic security of an EFG Γ with honest history h^* can be computed compositionally. That is, the only information needed of a subtree $\Gamma_{|h}$, to decide whether Γ satisfies security property is*

- *for weak(er) immunity: for which players $p \in N$ the subtree $\Gamma_{|h}$ is weak(er) immune;*
- *for collusion resilience: against which player groups $S \subset N$ the subtree $\Gamma_{|h}$ is collusion resilient;*
- *for practicality:*
  - *if h is along h^*: whether $h^*_{|h}$ is practical in $\Gamma_{|h}$;*
  - *if h is not along h^*: the set $\mathbb{U}(h)$ containing all practical utilities of $\Gamma_{|h}$. A utility $u(t)$ after terminal history $t \in \mathscr{T}$ is practical in subgame $\Gamma_{|h}$ iff t is practical in $\Gamma_{|h}$.*

Proof of Theorem 5.4. The theorem follows from the Theorems 5.5 to 5.7. □

Theorems 5.5 to 5.7 establish how to compositionally compute player-wise security for each security property, yielding a constructive proof of Theorem 5.4.

Theorem 5.5 (Compositional Weak Immunity)**.** *Let Γ be an EFG with honest history h^* and $p \in N$ a player. The following holds.*

1) A leaf of Γ is weak immune for p iff p's utility is non-negative:

$$\forall t \in \mathscr{T}.\, wi_p(\Gamma_{|t}) \quad \Leftrightarrow \quad u_p(t) \geq 0\,.$$

2) A branch of Γ is weak immune for p, where p is not the current player, iff all children are weak immune for p:

$$\forall h \in \mathscr{H} \setminus \mathscr{T}.\, p \neq P(h) \;\Rightarrow\; \big(wi_p(\Gamma_{|h}) \;\Leftrightarrow\; \forall a \in A(h).\, wi_p(\Gamma_{|(h,a)})\big)\,.$$

3) A branch of Γ along the honest history h^* is weak immune for the current player p, iff the child following h^* is weak immune for p. Let $a^* \in A(h)$ be the honest choice, i.e. (h, a^*) along h^*, then:

$$\begin{aligned}\forall h \in \mathscr{H} \setminus \mathscr{T}.\, p = P(h) \wedge h \text{ along } h^* \quad &\Rightarrow \\ \big(wi_p(\Gamma_{|h}) \;\Leftrightarrow\; wi_p(\Gamma_{|(h,a^*)})\big)\,.&\end{aligned}$$

4) A branch of Γ off the honest history h^* is weak immune for the current player p, iff there exists a child that is weak immune for p:

$$\begin{aligned}\forall h \in \mathscr{H} \setminus \mathscr{T}.\, p = P(h) \wedge h \text{ off } h^* \quad &\Rightarrow \\ \big(wi_p(\Gamma_{|h}) \;\Leftrightarrow\; \exists a \in A(h).\, wi_p(\Gamma_{|(h,a)})\big)\,.&\end{aligned}$$

Proof. We start by proving 1). By Definition 5.3, there has to exist a strategy $\sigma \in \mathscr{S}_{|t}$ such that for all $\tau \in \mathscr{S}_{|t}$ $u_{|t,p}(\sigma_p, \tau_{N-p}) \geq 0$. But for a terminal history $t \in \mathscr{T}$, the utility function $u_{|t,p} = u_p(t)$ is just a constant and the only strategy in $\mathscr{S}_{|t}$ is the empty strategy. Thus, equivalence 1) holds by definition.

For equivalence 2), let $h \in \mathscr{H} \setminus \mathscr{T}$ be a non-terminal history such that p is not the current player $p \neq P(h)$. We prove implication "$\Rightarrow$" first: Assuming $\Gamma_{|h}$ is weak immune for p, there exists a strategy $\sigma \in \mathscr{S}_h$ such that – if h is along h^*, it yields the honest history and – for all $\tau \in \mathscr{S}_{|h}$ the player p's utility $u_{|h,p}(\sigma_p, \tau_{N-p}) \geq 0$. Let now $a \in A(h)$ be an arbitrary choice after history h. We consider $\sigma_{|(a)} \in \mathscr{S}_{|(h,a)}$ – if (h, a) along h^*, then $H_{|(h,a)}(\sigma_{|(a)}) = h^*_{|(h,a)}$. Let further $\tau^a \in \mathscr{S}_{|(h,a)}$ be arbitrary. For $\tau' \in \mathscr{S}_{|h}$, such that $\tau'_{|(a)} = \tau^a$ and $\tau'(h) = a$ we know by assumption that $u_{|h,p}(\sigma_p, \tau'_{N-p}) \geq 0$. Since $p \neq P(h)$ and $\tau'(h) = a$, it follows $H_{|h}(\sigma_p, \tau'_{N-p}) = (a, H_{|(h,a)}(\sigma_{|(a),p}, \tau^a_{N-p}))$. Therefore, $u_{|(h,a),p}(\sigma_{|(a),p}, \tau^a_{N-p}) = u_{|h,p}(\sigma_p, \tau'_{N-p}) \geq 0$. As τ' was chosen arbitrarily, $\sigma_{|(a)}$ was constructed based on a, and a was also chosen arbitrarily, it follows the right-hand side of the equivalence: $\forall a \in A(h).\, wi_p(\Gamma_{|(h,a)})$.

To show implication "$\Leftarrow$", we assume $\forall a \in A(h).\, wi_p(\Gamma_{|(h,a)})$ and we construct a $\sigma' \in \mathscr{S}_{|h}$ weak immune for p. For all $a \in A(h)$ let $\sigma'_{|(a)} := \sigma^a$, where $\sigma^a \in \mathscr{S}_{|(h,a)}$ weak immune

for p – and if (h, a) along h^*, then σ^a also yields the honest history. Such strategies have to exist according to our assumptions. If h is along h^*, then define $\sigma'(h) := a^*$, where a^* is the next choice in h^* after h, which makes $H_{|h}(\sigma') = h^*_h$. Otherwise, let it be arbitrary. For an arbitrary $\tau \in \mathscr{S}_{|h}$, call $\tau(h) = a'$. Since $p \neq P(h)$, follows $H_{|h}(\sigma'_p, \tau_{N-p}) = (a', H_{|(h,a')}(\sigma^a_p, \tau_{|(h,a),N-p}))$. Hence, as σ^a was weak immune for p, we get $u_{|h,p}(\sigma'_p, \tau_{N-p}) = u_{|(h,a'),p}(\sigma^a_p, \tau_{|(h,a),N-p}) \geq 0$. Strategy τ was chosen arbitrarily; therefore, $\Gamma_{|h}$ is weak immune for player p.

We proceed by showing equivalence 3). Let $h \in \mathscr{H} \setminus \mathscr{T}$ be such that $p = P(h)$ and h along h^*. We further fix the honest choice after h to be $a^* \in A(h)$, i.e. (h, a^*) is along h^*. Let an honest (i.e. h^*-yielding) strategy $\sigma' \in \mathscr{S}_{|h}$ and an honest strategy $\sigma^{a^*} \in \mathscr{S}_{|(h,a^*)}$ be connected in the following way: $\sigma'_{|(a^*)} = \sigma^{a^*}$. Then, one of them is weak immune for p iff the other is. Assume σ' is weak immune for p. Pick an arbitrary $\tau^{a^*} \in \mathscr{S}_{|(h,a^*)}$, any extension of τ^{a^*} to a $\tau \in \mathscr{S}_{|h}$ has the property $\tau^{a^*}_{N-p} = \tau_{N-p}$, since $p = P(h)$. As σ' is weak immune for p, we know $u_{|h,p}(\sigma'_p, \tau_{N-p}) \geq 0$. Further, by definition of σ^{a^*}, follows $H_{|h}(\sigma'_p, \tau_{N-p}) = (a^*, H_{|(h,a^*)}(\sigma^{a^*}_p, \tau^{a^*}_{N-p}))$ and thus $u_{|(h,a^*),p}(\sigma^{a^*}_p, \tau^{a^*}_{N-p}) \geq 0$. Strategy $\tau^{a^*} \in \mathscr{S}_{|(h,a^*)}$ was arbitrary, so σ^{a^*} is weak immune for p. The other direction works the same way.

For equivalence 4), let $h \in \mathscr{H} \setminus \mathscr{T}$ be such that $p = P(h)$ and h not along h^*. For implication "$\Rightarrow$", we assume $\sigma \in \mathscr{S}$ is weak immune for p in $\Gamma_{|h}$. Let $a := \sigma(h) \in A(h)$ and $\sigma^a := \sigma_{|(a)} \in \mathscr{S}_{|(h,a)}$. We now fix an arbitrary $\tau^a \in \mathscr{S}_{|(h,a)}$, let $\tau \in \mathscr{S}_{|h}$ be any extension of τ^a, i.e. $\tau_{|(a)} = \tau^a$. Then, since $p = P(h)$, we know $H_{|h}(\sigma_p, \tau_{N-p}) = (a, H_{|(h,a)}(\sigma^a_p, \tau^a_{N-p})$, which further implies $u_{|(h,a),p}(\sigma^a_p, \tau^a_{N-p}) = u_{|h,p}(\sigma_p, \tau_{N-p}) \geq 0$. As τ^a was chosen arbitrarily and as we are not along h^*, that means σ^a is weak immune for p in $\Gamma_{(h,a)}$, which proves the implication. For the other direction "$\Leftarrow$", the same reasoning applies, when we assume $\sigma^a \in \mathscr{S}_{|(h,a)}$ is weak immune for p in $\Gamma_{(h,a)}$ and construct $\sigma \in \mathscr{S}$ such that $\sigma(h) = a$, $\sigma_{|(a)} = \sigma^a$ and the rest arbitrary. This concludes the last implication and, therefore, the proof of the theorem. □

Similar results to Theorem 5.5 hold for weaker immunity.

Example 5.13 (Compositional Weak Immunity)**.** We revisit the Market Entry game Γ_{me} of Figure 5.1, with honest history (n). We compute that Γ_{me} is weak immune using our compositional approach, where we stratify over players first and then split Γ_{me} into subtrees.

We start with player M. Theorem 5.5 implies that Γ_{me} is weak immune for M iff $\Gamma_{me|(n)}$ is weak immune for M; since $\Gamma_{me|(n)}$ is a leaf, we must check that the utility of M is non-negative, i.e. $0 \geq 0$. As this is true, game Γ_{me} is weak immune for M.

Next, E. According to Theorem 5.5, the game Γ_{me} is weak immune iff $\Gamma_{me|(n)}$ and $\Gamma_{me|(e)}$ are weak immune for E. The subgame $\Gamma_{me|(n)}$ is weak immune for E if their utility is non-negative, i.e. $p \geq 0$, true by assumption. The subtree $\Gamma_{me|(e)}$ is now weak immune

for E iff either $\Gamma_{me|(e,i)}$ or $\Gamma_{me|(e,pw)}$ is. E's utility at $\Gamma_{me|(e,i)}$ is $p/2 \geq 0$. Therefore Γ_{me} is weak immune for E, and from Theorem 5.3 it follows that Γ_{me} is weak immune.

Theorem 5.6 (Compositional Collusion Resilience)**.** *Let Γ be an EFG with honest history h^* and honest utility $u^* = u(h^*)$. The following equivalences hold.*

1) A leaf of Γ is collusion resilient against $S \subset N$ iff the honest joint utility of the deviating players $p \in S$ is greater than or equal to their joint utility at that leaf:

$$\forall t \in \mathscr{T}.\, cr_S(\Gamma_{|t}) \quad \Leftrightarrow \quad \sum_{p \in S} u_p^* \geq \sum_{p \in S} u_p(t)\ .$$

2) A branch of Γ, where the current player is in the deviating group $S \subset N$, is collusion resilient against S iff all children are collusion resilient against S:

$$\begin{aligned} &\forall h \in \mathscr{H} \setminus \mathscr{T}.\, P(h) \in S \quad \Rightarrow \\ &\quad \left(cr_S(\Gamma_{|h}) \ \Leftrightarrow\ \forall a \in A(h).\, cr_S(\Gamma_{|(h,a)})\right)\ . \end{aligned}$$

3) A branch of Γ along the honest history h^*, where the current player is not in the deviating group $S \subset N$, is collusion resilient against S iff the child following h^* is collusion resilient against S. Let $a^* \in A(h)$ be the honest action, i.e. (h, a^*) along h^*, then:

$$\begin{aligned} &\forall h \in \mathscr{H} \setminus \mathscr{T}.\, P(h) \notin S \wedge h \text{ along } h^* \quad \Rightarrow \\ &\qquad \left(cr_S(\Gamma_{|h}) \ \Leftrightarrow\ cr_S(\Gamma_{|(h,a^*)})\right)\ . \end{aligned}$$

4) A branch of Γ off the honest history h^*, where the current player is not in the deviating group $S \subset N$, is collusion resilient against S iff there exists a child that is collusion resilient against S:

$$\begin{aligned} &\forall h \in \mathscr{H} \setminus \mathscr{T}.\, P(h) \notin S \wedge h \text{ off } h^* \quad \Rightarrow \\ &\quad \left(cr_S(\Gamma_{|h}) \ \Leftrightarrow\ \exists a \in A(h).\, cr_S(\Gamma_{|(h,a)})\right)\ . \end{aligned}$$

Proof. We show equivalence 1) first. By Definition 5.12 there has to exist a $\sigma \in \mathscr{S}_{|t}$ (yielding h^* if applicable) such that for all $\tau \in \mathscr{S}_{|t}$ the inequality $\sum_{p \in S} u_p^* \geq \sum_{p \in S} u_{|t,p}(\sigma_{N-S}, \tau_S)$ holds. Since t is a terminal history, the utility function $u_{|t,p} = u_p(t)$ is constant and the only existing strategy is the empty strategy. Hence, the collusion resilience against S of $\Gamma_{|t}$ is equivalent to $\sum_{p \in S} u_p^* \geq \sum_{p \in S} u_p(t)$.

To show claim 2), we assume $h \in \mathscr{H} \setminus \mathscr{T}$ is a non-terminal history at which a player from the deviating group S has a turn $P(h) \in S$. We start with implication "$\Rightarrow$", assuming $\sigma \in \mathscr{S}$ is a collusion resilient strategy against S in $\Gamma_{|h}$. We pick an arbitrary action $a \in A(h)$ and define strategy $\sigma^a := \sigma_{|(a)} \in \mathscr{S}_{|(h,a)}$. If (h, a) is along the honest history h^*, then so is h. Thus the collusion resilient against S strategy σ yields the honest history $H_{|h}(\sigma) = h^*_{|h}$ which implies $H_{|(h,a)}(\sigma^a) = h^*_{|(h,a)}$. Let $\tau^a \in \mathscr{S}_{|(h,a)}$ be arbitrary. We extend it to $\tau \in \mathscr{S}_{|h}$ by defining $\tau(h) = a$, $\tau_{|(a)} = \tau^a$ and letting the rest be arbitrary. With this construction

and the fact that $P(h) \in S$ we get $H_{|h}(\sigma_{N-S}, \tau_S) = (a, H_{|(h,a)}(\sigma^a_{N-S}, \tau^a_S))$. Therefore, also $\sum_{p\in S} u^*_p \geq \sum_{p\in S} u_{|h,p}(\sigma_{N-S}, \tau_S) = \sum_{p\in S} u_{|(h,a),p}(\sigma^a_{N-S}, \tau^a_S)$, by the collusion resilience against S of σ. As τ^a was chosen arbitrarily, and σ^a was constructed for an arbitrary a the implication was proven.

For the other direction of 2) "$\Leftarrow$", we assume for all $a \in A(h)$ that $\sigma^a \in \mathscr{S}_{|(h,a)}$ is collusion resilient against S. Let $\sigma \in \mathscr{S}_{|h}$ be such that for all $a \in A(h)$ $\sigma_{|(a)} := \sigma^a$, and if h is along h^*, we additionally require $\sigma(h) = a^*$, where a^* is the honest choice after h. In this case follows $H_{|h}(\sigma) = h^*_{|h}$, since $H_{|(h,a^*)}(\sigma^{a^*}) = h^*_{|(h,a^*)}$ by assumption. We now pick an arbitrary strategy $\tau \in \mathscr{S}_{|h}$ and consider $a' := \tau(h) \in A(h)$. As $P(h) \in S$, for $\tau^{a'} := \tau_{|a'}$, it holds $H_{|h}(\sigma_{N-S}, \tau_S) = (a', H_{|(h,a')}(\sigma^{a'}_{N-S}, \tau^{a'}_S))$. Thus, also their utilities are identical which implies by the assumption that $\sigma^{a'}$ is collusion resilient against S, that $\sum_{p\in S} u^*_p \geq \sum_{p\in S} u_{|(h,a),p}(\sigma^a_{N-S}, \tau^a_S) = \sum_{p\in S} u_{|h,p}(\sigma_{N-S}, \tau_S)$. Hence, σ – and therefore $\Gamma_{|h}$ – is collusion resilient against S.

To show equivalence 3), we assume $h \in \mathscr{H} \setminus \mathscr{T}$ is a non-terminal history along the honest history h^* at which a not-deviating player has a turn $P(h) \notin S$. We again start by proving implication "$\Rightarrow$" and assume strategy $\sigma \in \mathscr{S}_{|h}$ yields the honest history and is collusion resilient against S in $\Gamma_{|h}$. We define $\sigma^{a^*} := \sigma_{|(a^*)} \in \mathscr{S}_{|(h,a^*)}$, which by construction also yields the honest history. For an arbitrary $\tau^{a^*} \in \mathscr{S}_{|(h,a^*)}$, let $\tau \in \mathscr{S}_{|h}$ be an extension, i.e. $\tau_{|(a^*)} := \tau^{a^*}$, rest arbitrary. Then, due to $P(h) \notin S$ and $\sigma(h) = a^*$, follows $H_{|h}(\sigma_{N-S}, \tau_S) = (a^*, H_{|(h,a^*)}(\sigma^{a^*}_{N-S}, \tau^{a^*}_S))$. As before, this implies that the collusion resilience against S of σ^* and, therefore, the collusion resilience against S of $\Gamma_{|(h,a^*)}$. For the other direction "$\Leftarrow$", we assume $\sigma^{a^*} \in \mathscr{S}_{|(h,a^*)}$ is honest and collusion resilient against S. We construct $\sigma \in \mathscr{S}_{|h}$ such that $\sigma_{|(a^*)} := \sigma^{a^*}$, $\sigma(h) = a^*$ and the rest arbitrary, with similar reasoning as before we conclude that σ yields the honest history and is collusion resilient against S.

For the last equivalence 4), let $h \in \mathscr{H} \setminus \mathscr{T}$ be a non-terminal history not along the honest history h^* at which a not-deviating player has a turn $P(h) \notin S$. The first implication "$\Rightarrow$" can be shown by assuming $\sigma \in \mathscr{S}$ is collusion resilient against S and by choosing $\sigma^a := \sigma_{|(a)} \in \mathscr{S}_{|(h,a)}$, where $a := \sigma(h)$. Fixing now an arbitrary $\tau^a \in \mathscr{S}_{|(h,a)}$, and an extension $\tau \in \mathscr{S}$ of it, since $\sigma(h) = a$ and $P(h) \notin S$, follows $H_{|h}(\sigma_{N-S}, \tau_S) = (a, H_{|(h,a)}(\sigma^a_{N-S}, \tau^a_S))$. As shown before this implies σ^a is collusion resilient against S in $\Gamma_{|(h,a)}$. The other implication "$\Leftarrow$" is similar to prove. Assume $a \in A(h)$ is such that $\sigma^a \in \mathscr{S}_{|(h,a)}$ is collusion resilient against S in $\Gamma_{|(h,a)}$. We construct a strategy $\sigma \in \mathscr{S}_{|h}$, by setting $\sigma(h) = a$ and $\sigma_{|(a)} := \sigma^a$, the rest can be arbitrary. Let now be $\tau \in \mathscr{S}_{|h}$ arbitrary. With $\tau^a := \tau_{|(a)} \in \mathscr{S}_{|(h,a)}$, we arrive at $H_{|h}(\sigma_{N-S}, \tau_S) = (a, H_{|(h,a)}(\sigma^a_{N-S}, \tau^a_S))$. This again implies that σ is collusion resilient against S in $\Gamma_{|h}$, which concludes the proof of equivalence 4) and hence the theorem. □

Note that, if a player is in a deviating group, all child subtrees need to be collusion resilient even if we are along honest history, as the deviator might choose any action and potentially harm honest players. In contrast, for an honest player and a node off

honest history, there needs to merely exist one collusion resilient child that the player can choose to defend against the deviating group.

Example 5.14 (Compositional Collusion Resilience)**.** We compositionally compute the collusion resilience of the Market Entry game Γ_{me} (Figure 5.1) with honest history (n). We have two possible colluding groups, both singletons $\{M\}$ and $\{E\}$.

Consider $\{M\}$. At the root of Γ_{me}, since the player M is in the colluding group, all subtrees must be collusion resilient against $\{M\}$. Along the honest history we reach a leaf $\Gamma_{me|(n)}$, which is collusion resilient (it is the honest leaf). For subtree $\Gamma_{me|(e)}$ there needs to exist a collusion resilient child, which is the case in the leaf after (e, pw): utility $-a$ is strictly smaller than the honest utility 0.

Next, $\{E\}$. At the root, M is not in the deviating group. Hence, only the honest child $\Gamma_{me|(n)}$ need be collusion resilient against $\{E\}$, which it is, as it is the honest leaf; so the utility is equal to the honest one in part (1) of Theorem 5.6. This suffices to establish collusion resilience against $\{E\}$; checking $\Gamma_{me|(e)}$ is unnecessary.

Using Theorem 5.3, it follows that Γ_{me} is collusion resilient.

To establish compositional practicality, we need the following small lemma.

Lemma 5.2. *A strategy $\sigma \in \mathscr{S}_{|h}$ is practical in subtree $\Gamma_{|h}$ iff strategy $\sigma_{|g} \in \mathscr{S}_{|(h,g)}$ is practical in subtree $\Gamma_{|(h,g)}$, for all $g \in \mathscr{H}_{|h}$.*

Proof. By Definition 5.6 and Definition 5.7, a strategy $\sigma \in \mathscr{S}_{|h}$ is practical if for all $g' \in \mathscr{H}_{|h}$ and all $\tau \in \mathscr{S}_{|(h,g')}$ the utility inequality holds. Whereas, $\sigma_{|g} \in \mathscr{S}_{|(h,g)}$ is practical for all $g \in \mathscr{H}_{|h}$, if for all $k \in \mathscr{H}_{|(h,g)}$ and for all $\tau \in \mathscr{S}_{|(h,g,k)}$ the utility inequality holds. Hence, by substituting $g' = (g, k)$ in the above-sketched formulae, it is easy to see that they are equivalent. □

Theorem 5.7 (Compositional Practicality)**.** *Let Γ be an EFG with honest history h^* and $\mathbb{U}(h)$ be the set of practical utilities of subtree $\Gamma_{|h}$. Let u^* be the honest utility $u^* = u(h^*)$. Then the following identities and equivalences hold.*

1) In a leaf of Γ the only practical utility is that of the leaf.

$$\forall t \in \mathscr{T}.\, \mathbb{U}(t) = \{u(t)\}\ .$$

2) The honest utility u^* is practical in a branch of Γ along h^* iff it is practical in the child following h^* and if for every other child at least one practical utility is not greater than u^* for the current player. Let $a^* \in A(h)$ be the honest action after h, then:

$$\begin{aligned} &\forall h \in \mathscr{H} \setminus \mathscr{T}.\, h \text{ along } h^* \quad \Rightarrow \quad \big(pr(\Gamma_{|h}) \quad \Leftrightarrow \\ &pr(\Gamma_{|(h,a^*)}) \ \wedge\ \forall a \in A(h) \setminus \{a^*\}\ \exists u \in \mathbb{U}((h,a)).\, u^*_{P(h)} \geq u_{P(h)}\big)\ . \end{aligned}$$

3) A utility is practical in a branch of Γ off the honest history h^* iff it is practical in a child and if, for every other child, at least one practical utility is not greater for the current player.

$$\begin{aligned}\forall h \in \mathscr{H} \setminus \mathscr{T}.\ h \text{ off } h^* \quad \Rightarrow \quad & \big(\forall t \in \mathscr{T}_{|h}.\ u(t) \in \mathbb{U}(h) \quad \Leftrightarrow \\ & \exists a \in A(h).\ u(t) \in \mathbb{U}((h,a)) \wedge \\ & \forall a' \in A(h) \setminus \{a\}\ \exists u' \in \mathbb{U}((h,a')).\ u_{P(h)}(t) \geq u'_{P(h)}\big)\ .\end{aligned}$$

Proof. We show claim 1) first. By Definitions 5.7 and 5.13, we know that the utility in the leaf after terminal history t is practical in subtree $\Gamma_{|h}$, if there is $\sigma \in \mathscr{S}_{|h}$, extending history t and $\forall g \in \mathscr{H}_{|h}\, \forall \tau \in \mathscr{S}_{|(h,g)}.\ u_{|(h,g),P(h,g)}(\sigma_{|g}) \geq u_{|(h,g),P(h,g)}(\sigma_{|g,N-P(h,g)}, \tau_{P(h,g)})$. In our case, we have $h = t$; hence the only strategy $\sigma \in \mathscr{S}_{|t}$ is the empty one. Further, the only history $g \in \mathscr{H}_{|t}$ is the empty history (hence $(h,g) = t$), and thus also the only strategy $\tau \in \mathscr{S}_{|t}$ is the empty strategy. This leaves us with $u(t)$ being practical iff $u_{|t,P(t)}(\sigma) \geq u_{|t,P(t)}(\tau)$, which is equivalent to $u_{P(t)}(t) = u_{P(t)}(t)$. Therefore, $u(t) \in \mathbb{U}(t)$, and since there are no other utilities is $\Gamma_{|t}$, follows $\mathbb{U}(t) = \{u(t)\}$.

To show equivalence 2), we fix a history $h \in \mathscr{H} \setminus \mathscr{T}$, which is along the honest history, and let a^* be the honest action after h. For implication "$\Rightarrow$" we assume strategy $\sigma \in \mathscr{S}_{|h}$ is practical in $\Gamma_{|h}$ and yields the honest history. Theorem 5.2 implies that also $\Gamma_{|(h,a^*)}$ is practical. To show the other conjunct of the right-hand side, let $a \in A(h) \setminus \{a^*\}$ be arbitrary. We apply Lemma 5.2, to get that $\sigma_{|(a)}$ is practical in $\Gamma_{|(h,a)}$. Hence, $u := u_{|(h,a)}(\sigma_{|(a)}) \in \mathbb{U}(h,a)$. Since σ is practical in $\Gamma_{|h}$, for $g = \emptyset \in \mathscr{H}_{|h}$ and $\tau \in \mathscr{S}_{|h}$ such that $\tau(h) = a$ and $\tau_{|(a)} = \sigma_{|(a)}$ follows

$$u^* = u_{|h,P(h)}(\sigma) \geq u_{|h,P(h)}(\tau_{P(h)}, \sigma_{N-P(h)}) = u_{|(h,a)}(\sigma_{|(a)})\ .$$

The last utility is exactly the considered $u \in \mathbb{U}(h,a)$, which concludes the proof of the implication.

To show the other direction "$\Leftarrow$", we assume the right-hand side holds. I.e. there exists an honest strategy $\sigma^{a^*} \in \mathscr{S}_{|(h,a^*)}$ that is practical in $\Gamma_{|(h,a^*)}$ as well as strategies $\sigma^a \in \mathscr{S}_{|(h,a)}$ for $a \neq a^*$, that are practical in $\Gamma_{|(h,a)}$ and whose utilities $u^a \in \mathbb{U}(h,a)$ satisfy $u^*_{P(h)} \geq u^a_{P(h)}$. We now construct a strategy $\sigma \in \mathscr{S}_{|h}$ in the following way: Let $\sigma(h) = a^*$ and for all $a \in A(h)$ (including a^*) let $\sigma_{|(a)} = \sigma^a$. First, we note that σ yields the honest history by construction. Second, we prove it is practical in $\Gamma_{|h}$ by picking an arbitrary history $g \in \mathscr{H}_{|h}$ and an arbitrary strategy $\tau \in \mathscr{S}_{|(h,g)}$ and showing $u_{|(h,g),P(h,g)}(\sigma_{|g}) \geq u_{|(h,g),P(h,g)}(\tau_{P(h,g)}, \sigma_{|g,N-P(h,g)})$:

Case 1: $g = \emptyset$. Then $u_{|(h),P(h)}(\sigma) = u^*_{P(h)}$, as σ yields the honest history and h along h^*. Also, $u_{|(h),P(h)}(\tau_{P(h)}, \sigma_{N-P(h)}) = u_{|(h,a),P(h)}(\tau_{|(a),P(h)}, \sigma_{|(a),N-P(h)})$, where $a = \tau(h)$. Using Theorem 5.3 and the fact that $\sigma_{|(a)}$ is practical in $\Gamma_{|(h,a)}$, follows $u_{|(h,a),P(h)}(\tau_{|(a),P(h)}, \sigma_{|(a),N-P(h)}) \leq u_{|(h,a),P(h)}(\sigma_{|(a)}) = u^a_{P(h)}$. Applying our assumption $u^*_{P(h)} \geq u^a_{P(h)}$, the required inequality holds. Case 2: $g = (a, g')$, where $a \in A(h)$,

$g' \in \mathscr{H}_{|(h,a)}$. Then, $u_{|(h,g),P(h,g)}(\sigma_{|g}) = u_{|(h,g),P(h,g)}(\sigma^a_{|g'})$ which is – due to the practicality of σ^a – greater than or equal to

$$\begin{aligned}
&u_{|(h,g),P(h,g)}(\tau_{P(h,g)}, \sigma^a_{|g',N-P(h,g)}) \\
&= u_{|(h,a,g'),P(h,a,g')}(\tau_{P(h,a,g')}, \sigma^a_{|g',N-P(h,a,g')}) \\
&= u_{|(h,g),P(h,g)}(\tau_{P(h,g)}, \sigma_{|g,N-P(h,g)}).
\end{aligned}$$

Hence, the second direction of the second claim is proven.

For equivalence 3), let $h \in \mathscr{H} \setminus \mathscr{T}$ be off the honest history h^* and $t \in \mathscr{T}_{|h}$. We prove implication "$\Rightarrow$" first and assume $u_{|h}(t) \in \mathbb{U}(h)$ is a practical utility in $\Gamma_{|h}$. Thus, there exists a strategy $\sigma \in \mathscr{S}_{|h}$ such that $H(\sigma) = t$ and σ is practical. History t is terminal, while h is not. Therefore, t is not empty. Hence, we can split it into $t := (a^t, t')$, where a^t is the first action along t. Applying Lemma 5.2, we know that also $\sigma_{(a^t)} \in \mathscr{S}_{|(h,a^t)}$ is practical. Since $H(\sigma) = t$, follows $\sigma(t') = a^t$, and thus $u(t) = u_{|(h,a^t)}(\sigma_{|(a^t)}) \in \mathbb{U}(h, a^t)$ which proves the first conjunct of the implication. For the second, pick an arbitrary $a \in A(h) \setminus \{a^t\}$ and consider the strategy $\sigma \in \mathscr{S}_{|h}$ from before. Using Lemma 5.2 again, it follows that also $\sigma_{|(a)} \in \mathscr{S}_{|(h,a)}$ is practical and by definition of $\mathbb{U}$ we know $u_{|(h,a)}(\sigma_{|(a)}) \in \mathbb{U}(h, a)$. Towards using the practicality of σ in $\Gamma_{|h}$, let $\tau \in \mathscr{S}_{|h}$ be such that $\tau(h) = a$ and $\tau_{|(a)} = \sigma_{|(a)}$. Finally, we conclude $u_{|h,P(h)}(t) = u_{|h,P(h)}(\sigma) \geq u_{|h,P(h)}(\tau_{P(h)}, \sigma_{N-P(h)}) = u_{|(h,a),P(h)}(\sigma_{|(a)})$. Hence, also the second conjunct holds, and the implication is proven.

The other direction, "$\Leftarrow$" of equivalence 3), can be shown similarly to "$\Leftarrow$" of equivalence 2). We assume $u_{|h}(t) \in \mathbb{U}(h, a')$ and for all other actions $a \in A(h) \setminus \{a'\}$ exists a practical utility $u^a \in \mathbb{U}(h, a)$ such that $u_{|h,P(h)}(t) \geq u^a_{P(h)}$. Then, we construct a strategy $\sigma \in \mathscr{S}_{|h}$ in the following way. Let $\sigma(h) = a'$. For all $a \in A(h) \setminus \{a'\}$, let $\sigma_{|(a)} = \sigma^a$, where σ^a is a practical strategy in $\Gamma_{|(h,a)}$ with practical utility $u^a = u_{|(h,a)}(\sigma^a) \in \mathbb{U}(h, a)$. For a', let $\sigma_{(a')}$ be the practical strategy in $\Gamma_{|(h,a')}$ that yields utility $u_{|h}(t)$. It follows that strategy σ also yields utility $u_{|h}(t)$. With this construction, we can proceed as in equivalence 2), direction "$\Leftarrow$", to prove the practicality of σ which implies the practicality of $u_{|h}(t)$ in $\Gamma_{|h}$ to conclude the proof of the theorem. □

Example 5.15 (Compositional Practicality)**.** To compositionally compute the practicality of the Market Entry game Γ_{me} of Figure 5.1 with honest history (n), we start with the leaves of the tree, where the practical utilities are the utilities of the leaves. Moving upwards in the tree, we look at the subtree $\Gamma_{me|(e)}$, which is off the honest history, so we take the better utility for player E, setting $\mathbb{U}(e) = \{(\frac{p}{2}, \frac{p}{2})\}$. At the root of the tree, which is along the honest history, the practical utility of the honest subtree $(0, p)$ should be practical. Since all practical utilities of the non-honest child (there is just one) are better for player M (as $\frac{p}{2} > 0$), the honest utility is not practical. Theorem 5.3 then implies that Γ_{me} is not practical.

5.6 Automating Compositional Security Analysis

Section 5.5 assumed a total order $\preceq$ on game utility terms T_u. This section lifts fixed ordering constraints $(\preceq, T_u)$ and interprets the game variables in the utility terms T_u as real-valued variables $\vec{x}$, as explained in Section 5.3.2. The results of Theorem 5.1 are also propagated to our player-wise security properties from Theorem 5.3, as the universal quantification over players, player groups, subgames, and strategies is independent of the values $\vec{x}$ the variables in the utility terms take. As such and as an example, weak immunity (5.9) becomes equivalent to the player-wise weak immunity property:

$$\forall(\preceq, T_u)\ \forall p \in N\ \exists \sigma \in \mathscr{S}.\, H(\sigma) = h^* \wedge \forall \tau \in \mathscr{S}\ .$$
$$\forall \vec{x}.\, \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \rightarrow u_p(\sigma_p, \tau_{N-p})[\vec{x}] \geq 0\,. \tag{5.13}$$

Mapping game-theoretic security from Theorem 5.1 to the player-wise security of Theorem 5.3 is crucial for automating compositional security: we only forward relatively small first-order expressions of the form

$$\forall \vec{x}.\, \left(\bigwedge_{c \in C \cup \preceq} c[\vec{x}] \right) \rightarrow ut_1[\vec{x}] \geq ut_2[\vec{x}]\ , \tag{5.14}$$

to an SMT solver, where ut_1 and ut_2 are term expressions over $\vec{x}$; checking such formulas is very feasible for SMT solvers.

As usual, to check whether (5.14) is a theorem, the property is first negated, and then an SMT solver is used to check satisfiability. This is where the simplified quantified structure of (5.14) becomes especially *friendly for automation*: The SMT solving of (5.14) happens in a purely existential fragment, for which efficient decision procedures exist [BT18, BN24]. The remaining reasoning in (5.13), about the players and the existence of strategies σ witnessing player-wise security, is performed using the compositional security results of Theorems 5.5 to 5.7, without burdening the SMT solver. Such an interplay between SMT solving and compositional security eases automation, as illustrated below and detailed further in Section 5.6.1.

Example 5.16 (SMT Reasoning for Compositional Security)**.** Revisiting the Market Entry game Γ_{me} of Figure 5.1, we study the SMT formulae resulting from (5.14). Initial constraints $C = \{a > 0, p > 0\}$. All symbolic utility terms occurring in the security properties of Γ_{me} are already totally ordered by the constraints in C. Hence, the only relevant total order $\preceq$ here is that consistent with C, $-a \prec 0 \prec p/2 \prec p$.

To analyze, for example, the weak immunity of the Market Entry game for player E compositionally, we follow the algorithm induced by Theorem 5.5. The SMT reasoning is only needed when we reach a leaf, such as the one after history (n). The resulting SMT query is

$$\forall a, p.\ a > 0 \wedge p > 0 \rightarrow p \geq 0\ ,$$

which is trivially valid. Hence, the subtree after history (n) is weak immune for E for all allowed utility values.

5.6.1 Divide-and-Conquer Algorithms for Compositional Security

Our compositionality results from Theorem 5.3 and Theorems 5.5 to 5.7, extended by a lazy total-order approach, induce a *divide-and-conquer approach for splitting and combining reasoning over game subtrees and supertrees.* Our overall divide-and-conquer framework for automating compositional game-theoretic reasoning is summarized in Algorithm 5.1, which in turn relies upon Algorithm 5.3 as well as upon Algorithms 5.2, 5.4 and 5.5 from the appendix. We compositionally compute the game-theoretic security of a protocol, analyzing the (protocol) game for all real-valued variables $\vec{x}$ of utitilty terms T_u, considering all total orders $\preceq$ at once. If we fail, we split the total orders into multiple cases, unless we can conclude that the respective security property cannot be satisfied even if we restrict the values of $\vec{x}$ to one total order $\preceq$. The case split we consider is induced by an SMT query as in property (5.14) when some but not all $\vec{x}$ satisfy the implication. We then split into total orders that enforce $ut_1[\vec{x}] \geq ut_2[\vec{x}]$, respectively $ut_1[\vec{x}] < ut_2[\vec{x}]$, in (5.14).

Algorithm 5.1: Function `SatisfiesProperty`. In Algorithm 5.1 an instance Π, which contains the game tree Γ, the set of infinitesimal variables inf (as introduced in Section 5.2), and the set of initial constraints C, is given as input. The input to Algorithm 5.1 also contains the honest history h^*, the security property to be analyzed, and the currently considered case `case`.

The function `SatisfiesProperty` in Algorithm 5.1 is called initially with the empty case to analyze all total orders. This case can be refined throughout Algorithm 5.1, using case splits. Hence, in the first call of the function, the set S, representing the constraints handed to an SMT solver, contains only the initial constraints C. The relevant player groups `RelevantGroups` of security property sp are set according to the stratified definitions of sp from Section 5.5.1: N for $w(er)i$, as we stratify over players; $2^N \setminus \{\emptyset, N\}$ for cr, as we stratify over deviating subgroups; and {"none"} for pr.

Function `ComputeSP` in line 6 of Algorithm 5.1 stands for `ComputeWI`, `ComputeWERI`, `ComputeCR` or `ComputePR` (Algorithms 5.3 to 5.5), depending on the security property sp, as summarized in Algorithm 5.2.

The result of `ComputeSP` depends on whether Γ with honest history h^* satisfies property sp for/against pg, given the constraints in S. Here, we also keep track of utility comparisons we cannot decide. Importantly, the constraint $ut_1[\vec{x}] \geq ut_2[\vec{x}]$ to whether Γ satisfies sp in `case` is returned as $\mathsf{split}_{\mathsf{pg}}$, if it exists.

The loop in lines 5–12 of Algorithm 5.1 incorporates player-wise security from Theorem 5.3. It additionally provides a necessary case split if the security property is violated for a player group. Subsequently, the respective results are returned: `true` for all groups yields `true`; `false` but nothing to split on for at least one group yields `false`; and

Algorithm 5.1: Function `SatisfiesProperty` for Compositional Game-Theoretic Security Reasoning.

input : input instance $\Pi = (\Gamma, inf, C)$, honest history h^*, the name of a security property $sp \in \{wi, weri, cr, pr\}$, and the currently analyzed case (as set of SMT constraints) `case`.

output : `true` if Π satisfies sp in case `case`, `false` otherwise

```
1  S ← ∅
2  AddConstraints (S, C ∪ case)
3  result ← true
4  split ← null
5  for pg ∈ RelevantGroups(Π, sp) do
6  |  (result_pg, split_pg) ← ComputeSP (Π,h*,S,sp,pg)
7  |  if result_pg = false then
8  |  |  result ← result_pg
9  |  |  split ← split_pg
10 |  |  break
11 |  end
12 end
13 if result = true then
14 |  return true
15 end
16 if split = null then
17 |  return false
18 end
19 for constr ∈ {split, ¬split} do
20 |  if ¬SatisfiesProperty(Π, h*, sp, case ∪ {constr}) then
21 |  |  return false
22 |  end
23 end
24 return true
```

`false` together with a split leads to further case splits (lines 19–24). If we split the total orders into multiple cases, all the cases have to return `true` for the property to be satisfied.

Example 5.17. Let us revisit the Market Entry game from Example 5.1, but this time let us assume only $p > 0$ is the initial constraint and $a \in \mathbb{R}$ can take any value. We check whether the honest history (n) is collusion resilient. Algorithm 5.1 will in line 5 consider each singleton player group individually, suppose we start with $\{M\}$. The function `ComputeCR`, specified in Algorithm 5.4, returns $(\texttt{false}, 0 \geq -a)$, as the comparison between 0 and $-a$ is missing to determine collusion resilience. So the

Algorithm 5.2: Relay Function `ComputeSP`.

input : input instance Π, honest history h^*, security property $sp \in \{wi, weri, cr, pr\}$, set S containing initial constraints and currently analyzed case, player group pg.

output : $(\mathsf{result}, \mathsf{split})$, where result states whether Π satisfies sp for pg, given S, and split a crucial utility comparison we cannot decide.

```
1  if sp = wi then
2  |  return ComputeWI(Γ, h*, S, pg)
3  else if sp = weri then
4  |  return ComputeWERI(Γ, h*, S, pg)
5  else if sp = cr then
6  |  return ComputeCR(Γ, h*, S, pg)
7  else
8  |  (result, split) ← ComputePR(Γ, h*, S)
9  |  if result = ∅ then
10 |  |  return (false, split)
11 |  else
12 |  |  return (true, split)
13 |  end
14 end
```

result is set to `false` and split to $0 \geq -a$. For the colluding group $\{E\}$ the function `ComputeCR` returns $(\texttt{true}, \mathsf{null})$, as the honest player M can choose a collusion resilient honest action. Algorithm 5.1 then proceeds with line 19, refining the constraints by first adding $0 \geq -a$ to the case. The function `SafisfiesProperty` will return `true` (and empty split); and then adding the negated constraint $0 < -a$ to the case, at which point `SafisfiesProperty` will return `false`, since M can profit by deviating from the honest action (n), as both $\frac{p}{2}$ and $-a$ are better utilities than 0. Algorithm 5.1 thus terminates by returning `false`.

The security-property-specific function variants of `ComputeSP` recursively apply the compositional results of Theorem 5.4. To illustrate case splitting of total orders, we describe function `ComputeWI` of Algorithm 5.3 in detail.

Algorithm 5.3: Function `ComputeWI` The function `ComputeWI` of Algorithm 5.3 is initially called with the entire game tree Γ from function `SatisfiesProperty` of Algorithm 5.1. We then proceed recursively, according to Theorem 5.5. Note that the player group pg is just one player.

In a leaf, `GetUtility` in Algorithm 5.3 returns $ut(\vec{x})$ as the utility of player pg. We then – in line 2 of Algorithm 5.3 – check whether the constraints in S together with $ut(\vec{x}) < 0$ are `unsat`. This is equivalent to the constraints in S implying $ut(\vec{x}) \geq 0$,

which is an instance of property (5.14), except that we do not (necessarily) have one total order $\preceq$ at hand, only some constraints from `case`. If the implication holds, we return `true`.

Algorithm 5.3: Function `ComputeWI` for Weak Immunity.

input : game tree Γ, honest history h^*, set S containing initial constraints and current case, player group pg.

output : $(\mathsf{result}, \mathsf{split})$, where result states whether Γ is weak immune for pg, given S, and split a crucial utility comparison we cannot decide.

```
1  if isLeaf(Γ) then
2  |  if Check(S, GetUtility(Γ, pg) < 0) = unsat then
3  |  |  return (true, null)
4  |  end
5  |  if Check(S, GetUtility(Γ, pg) ≥ 0) = unsat then
6  |  |  return (false, null)
7  |  end
8  |  return (false, GetUtility(Γ, pg) ≥ 0)
9  end
10 if CurrentPlayer(Γ) ≠ pg then
11 |  for a ∈ Actions(Γ) do
12 |  |  (result, split) ← ComputeWI(Γ_|(a), h*, S, pg)
13 |  |  if result = false then
14 |  |  |  return (result, split)
15 |  |  end
16 |  end
17 |  return (true, null)
18 end
19 if AlongHonest(Γ, h*) then
20 |  a* ← HonestAction(Γ, h*)
21 |  return ComputeWI(Γ_|(a*), h*, S, pg)
22 end
23 newsplit ← null
24 for a ∈ Actions(Γ) do
25 |  (result, split) ← ComputeWI(Γ_|(a), h*, S, pg)
26 |  if result = true then
27 |  |  return (true, null)
28 |  else if split ≠ null then
29 |  |  newsplit ← split
30 |  end
31 end
32 return (false, newsplit)
```

Otherwise, we check the opposite condition, by asking in line 5 of Algorithm 5.3 whether

$$\forall \vec{x}. \bigwedge_{c \in C \cup \mathsf{case}} c[\vec{x}] \to ut[\vec{x}] < 0 \tag{5.15}$$

holds. If it does (line 6), the leaf is not weak immune. Otherwise (line 8), the leaf's weak immunity depends on the total order, which induces a case split on $ut[\vec{x}] \geq 0$.

At a branch (lines 10–32 of Algorithm 5.3), we check in which of the cases of Theorem 5.5 we are. We then call the function `ComputeWI` recursively on immediate subgames $\Gamma_{|(a)}$ and propagate the result accordingly. Note that, for simplicity, in line 13 of Algorithm 5.3 we do not wait for a null split that would immediately return `false`, but rather proceed with a split. However, as there are only finitely many possible case splits, we will eventually see the null split for a `false` subtree if it exists and return it to reach the correct result.

Example 5.18. We mimic the execution of the function `ComputeWI` from Algorithm 5.3 on the Market Entry game from Example 5.1, but this time only assume $a > 0$ is the initial constraint, and $p \in \mathbb{R}$ can take any value. Suppose we enter Algorithm 5.3 with the entire tree Γ_{me}, honest history (e, i) and player $pg = M$. Since the root of the tree is along the honest history, the function will jump to line 19, and recursively call `ComputeWI` for the honest subtree $\Gamma_{me|(e)}$. Then the current player E is not pg, so we proceed with line 10, and iterate through the actions pw and i. Suppose we first look at the action pw and, from line 12, recursively compute the weak immunity for the leaf after (e, pw). Algorithm 5.3 will execute lines 1 and 2, and since the utility of player M is 0, which is a non-negative number, the check in line 2 will be unsat, so the function returns (`true`, null). For the other action i, we recursively compute (line 12 of the algorithm) the weak immunity for the leaf after (e, i). The function `GetUtility`$(\Gamma_{me|(e,i)}, M)$ will return $\frac{p}{2}$, for which we cannot decide whether it is non-negative (there are no initial constraints on p). Both conditions from lines 2 and 5 are thus false and we return the pair $(\texttt{false}, \frac{p}{2} \geq 0)$ in line 8. Proceeding from the supertree $\Gamma_{me|(e)}$ in line 12, with the result being `false`, we return in line 14 the pair $(\texttt{false}, \frac{p}{2} \geq 0)$.

Function `ComputeWERI`. The function for weaker immunity is identical to `ComputeWI` in Algorithm 5.3, except that `GetUtility` returns only the real part of the requested utility.

Algorithm 5.4: Function `ComputeCR`. The function in Algorithm 5.4 is similar to `ComputeWI` in Algorithm 5.3. It differs in the considered utility comparison $ut_1[\vec{x}] \geq ut_2[\vec{x}]$. While ut_1 was player pg's utility and $ut_2 = 0$, for collusion resilience, ut_1 is the sum of the honest utilities of the deviating players pg and ut_2 is the sum of the utilities of the deviating players $p \in$ pg in the current leaf. Hence pg is a group of players rather than a single player. Other than that, in the branch-case (lines 10–32) the roles of whether the current player `CurrentPlayer`(Γ) is in pg or not, are reversed, as it is in Theorem 5.6.

Algorithm 5.4: Function `ComputeCR` for Collusion Resilience.

input : a game tree Γ, an honest history h^*, the set S containing the initial constraints and the current case, and the player group pg.

output : $(\mathsf{result}, \mathsf{split})$, where result is true iff Γ is collusion resilient against pg, given S, and split a crucial utility comparison we cannot decide.

```
1  if isLeaf(Γ) then
2  |  if Check(S, GetUtility(h*, pg) < GetUtility(Γ, pg)) = unsat then
3  |  |  return (true, null)
4  |  end
5  |  if Check(S, GetUtility(h*, pg) ≥ GetUtility(Γ, pg)) = unsat then
6  |  |  return (false, null)
7  |  end
8  |  return (false, GetUtility(h*, pg) ≥ GetUtility(Γ, pg)))
9  end
10 if CurrentPlayer(Γ) ∈ pg then
11 |  for a ∈ Actions(Γ) do
12 |  |  (result, split) ← ComputeCR(Γ_|(a), h*, S, pg)
13 |  |  if result = false then
14 |  |  |  return (result, split)
15 |  |  end
16 |  end
17 |  return (true, null)
18 end
19 if AlongHonest(Γ, h*) then
20 |  a* ← HonestAction(Γ, h*)
21 |  return ComputeCR(Γ_|(a*), h*, S, pg)
22 end
23 newsplit ← null
24 for a ∈ Actions(Γ) do
25 |  (result, split) ← ComputeCR(Γ_|(a), h*, S, pg)
26 |  if result = true then
27 |  |  return (true, null)
28 |  else if split ≠ null then
29 |  |  newsplit ← split
30 |  end
31 end
32 return (false, newsplit)
```

Algorithm 5.5: Function `ComputePR` for Practicality.

input : a game tree Γ, honest history h^*, set S containing the initial constraints and the current case.

output : $(\mathbb{U}_\Gamma, \mathsf{split})$, where $\mathbb{U}_\Gamma$ are all practical (and honest if along h^*) utilities in Γ, and split a crucial utility comparison that cannot be decided.

```
1  if isLeaf(Γ) then
2  |  return ([GetUtility(Γ)], null)
3  end
4  subtrees ← [ComputePR(Γ_|(a), h*, S) for a ∈ Actions(Γ)]
5  p ← CurrentPlayer(Γ)
6  U_Γ ← ∅
7  if AnyUtilityEmpty(subtrees) then
8  |  return subtrees.ReturnEmpty()
9  end
10 if AlongHonest(Γ, h*) then
11 |  ([u*], split*) ← GetHonestResult(subtrees)
12 |  for (U^a, _) ∈ subtrees \ {([u*], split*)} do
13 |  |  (result, split) ← ExistsDominated(U^a, u*, S, p)
14 |  |  if ¬result then
15 |  |  |  return (∅, split)
16 |  |  end
17 |  end
18 |  return ([u*], null)
19 end
20 for (U^a, split^a) ∈ subtrees do
21 |  for u ∈ U^a do
22 |  |  toAdd ← true
23 |  |  for (U^s, _) ∈ subtrees \ (U^a, split^a) do
24 |  |  |  (result, split) ← ExistsDominated(U^s, u, S, p)
25 |  |  |  if ¬result ∧ split ≠ null then
26 |  |  |  |  return (∅, split)
27 |  |  |  else if ¬result ∧ split = null then
28 |  |  |  |  toAdd ← false
29 |  |  |  end
30 |  |  end
31 |  |  if toAdd then
32 |  |  |  U_Γ.Add(u)
33 |  end
34 end
35 return (U_Γ, null)
```

Algorithm 5.5: Function `ComputePR`. The function `ComputePR` to compute the practicality of game tree Γ for honest history h^*, and in the case specified in set S operates a little differently than the ones for the other properties. As described in Theorem 5.7, we have to keep track of all practical utilities of subtrees, in order to decide the practicality of h^*.

Hence, for a leaf, trivially, the only practical utility is its utility (lines 1–3). For branches, we first check if any results of the subtrees yield an empty list of practical utilities in lines 4–9. If yes, then either we need a case split (if $\mathsf{split} \neq \mathsf{null}$) or the honest history was not practical in a subtree, which implies that it is not practical in the entire tree (Theorem 5.2), otherwise. In any case, returning the respective `ComputePR` result does precisely that.

If none of the subtrees led to an empty set of practical utilities, we distinguish between whether the current subtree Γ is along the honest history h^* ($\texttt{AlongHonest}(\Gamma, h^*)$). If so, in lines 10–19 it is analyzed whether the honest utility u^* dominates at least one practical utility for each sibling, thereby proceeding as in Theorem 5.7. In case we found a dominated utility per sibling, the honest utility is returned (line 18), otherwise the empty list of utilities, together with a crucial case split (if it exists) is returned (line 15).

The function `ExistsDominated`, as defined in Algorithm 5.6, computes whether one of the utilities in $\mathbb{U}$ has to be less than or equal to u for the current player p for all values of $\vec{x}$ that satisfy the preconditions in S. If the relation of two terms cannot be decided, we store it in split.

For the case where Γ is not along h^* (lines 20–35), the list of all practical utilities has to be computed. Similarly, as before, a utility that was practical in a subtree is practical now if it dominates at least one practical utility of each sibling. To decide domination, again function `ExistsDominated` is employed. If a further case distinction is needed, the empty list together with that split is returned (line 26); otherwise, the list of practical utilities together with the null-split (line 35).

In addition to compositional security via Algorithm 5.1, our work supports additional features to debug a protocol and better understand its structure. Those include (i) strategy extraction in case the considered security property was satisfied (Section 5.6.3), (ii) finding counterexamples (Section 5.6.4), and (iii) providing weakest preconditions to make the game secure otherwise. Computing preconditions in our compositional setting can be done by collecting all cases where the security property is violated, and then conjoining and negating them afterwards.

5.6.2 Correctness of the Algorithm

To prove Theorem 5.8, we need some preliminary results first. We start with two lemmas about the `ComputePR` function.

Lemma 5.3. *Let Γ be a subtree of a game Γ', h^* an honest history of the Γ', S be a set of initial constraints C and case `case`, and* $\texttt{ComputePR}(\Gamma, h^*, \mathsf{S}) = (\mathbb{U}_\Gamma, \mathit{split})$*. If a utility*

Algorithm 5.6: Function `ExistsDominated`.

input : a list of utility tuples $\mathbb{U}$, a single utility tuple u, set S containing the initial constraints and the current case, a player p.

output : $(\mathsf{result}, \mathsf{split})$, where result is `true` if there exists a utility $u' \in \mathbb{U}$ such that $u_p \geq u'_p$ for all values that satisfy S; `false` otherwise; and split a crucial utility comparison that cannot be decided.

```
1  existsDominated ← false
2  split ← null
3  for u' ∈ 𝕌 do
4  |  if Check(S, u[p] < u'[p]) = unsat then
5  |  |  existsDominated ← true
6  |  else if Check(S, u[p] ≥ u'[p]) = sat then
7  |  |  split ← u[p] ≥ u'[p]
8  |  end
9  end
10 return (existsDominated, split)
```

u is an element of $\mathbb{U}_\Gamma$, then for all values $\vec{x}$ satisfying $C \cup \mathtt{case}$ $u[\vec{x}]$ is practical in Γ, and – if Γ is along h^* – $u = u(h^*)$.

Proof. We prove this claim using structural induction. For the base case, we assume Γ is a leaf. Then, according to Algorithm 5.5, lines 1–3, the utility of the leaf will be returned. It is by definition also the only practical utility of the leaf (for all values of $\vec{x}$). If the leaf is along the honest history, it is further the honest utility, hence the base case is shown.

Let us now assume Γ is a branch and that we have shown the property for all subtrees of Γ. We first consider the case where Γ is along h^*. Then, by induction hypothesis and assuming a^* is the honest action, if $\mathbb{U}_{\Gamma_{|(a^*)}}$ contains an element u_{a^*} it has to be the honest utility $u_{a^*} = u(h^*)$ and it has to be practical for all values of $\vec{x}$ that satisfy $C \cup \mathtt{case}$. Hence, $u^* = u(h^*)$ in line 11. Following lines 12–17 of the algorithm and Algorithm 5.6, $u(h^*)$ is returned iff for all siblings Γ_a of $\Gamma_{(a^*)}$ exists a (by induction hypothesis practical) utility u_a such that the (by construction satisfiable) constraints in S together with the constraint $u_p(h^*) < u_{a,p}$ is unsatisfiable, where p is the current player at Γ. This equivalent to all $\vec{x}$ that satisfy $C \cup \mathtt{case}$ also satisfy $u_p(h^*)[\vec{x}] \geq u_{a,p}[\vec{x}]$. Applying now Theorem 5.7, it follows that $u(h^*)[x]$ is practical in Γ for all $\vec{x}$ that satisfy $C \cup \mathtt{case}$ in this exact case. Hence, if $u \in \mathbb{U}_\Gamma$, then $u = u(h^*)$ and it is practical for all $\vec{x}$ that satisfy $C \cup \mathtt{case}$.

Secondly, assume Γ is not along the honest history. Again, by induction hypothesis all utilities occurring in line 4 $\mathsf{subtrees}$, are practical in their subgames for all $\vec{x}$ satisfying $C \cup \mathtt{case}$. A utility $u_a \in \mathbb{U}_{\Gamma_{|(a)}}$ is now added to the returned set $\mathbb{U}_\Gamma$, exactly if for all siblings $\Gamma_{|b}$ there exists a (by induction hypothesis practical in $\Gamma_{|b}$ for the $\vec{x}$ satisfying the constraints) utility u_b such that $C \cup \mathtt{case} \cup u_{a,p} < u_{b,p}$ is unsat. Which is again according

to Theorem 5.7 equivalent to u_a being practical in Γ for all $\vec{x}$ satisfying $C \cup \texttt{case}$. This concludes the induction step and hence the proof of the lemma. □

Lemma 5.4. *Let Γ be a subtree of a game Γ', h^* an honest history of Γ', S a set of initial constraints C and a case `case`, and $\texttt{ComputePR}(\Gamma, h^*, \mathsf{S}) = (\mathbb{U}_\Gamma, \mathit{null})$. If a utility u from Γ is not an element of $\mathbb{U}_\Gamma$, then for all values $\vec{x}$ satisfying $C \cup \texttt{case}$ the utility $u[\vec{x}]$ is not practical in Γ, or – if Γ is along h^* – $u \neq u(h^*)$.*

Proof. We prove the lemma by structural induction. For the base case we assume Γ is a leaf. If a utility is not in $\mathbb{U}_\Gamma$, that means that it is not the leaf utility which is equivalent to not being practical in Γ.

For the inductive case, we assume Γ is a branch. First, we further assume that Γ is along h^*. By induction hypothesis, Lemma 5.3 and by Algorithm 5.5, the only candidate for being in $\mathbb{U}_\Gamma$ is $u(h^*)$, if it was in $\mathbb{U}_{\Gamma_{|(a^*)}}$. All others are by the algorithm not in $\mathbb{U}_{\Gamma_{a^*}}$ and by induction hypothesis therefore either not $u(h^*)$ or are $u(h^*)$ but not practical in $\Gamma_{|(a^*)}$. In any case, all of those are also in Γ not $u(h^*)$ or are $u(h^*)$ but not practical in Γ, according to Theorem 5.7. Therefore, assume $u(h^*) \in \mathbb{U}_{\Gamma_{|(a^*)}}$, but is not in $\mathbb{U}_{|\Gamma}$. Following the algorithm, this implies that there is a child $\Gamma_{|(a)}$, for which all practical utilities $u \in \mathbb{U}_{\Gamma_{|(a)}}$ (by induction hypothesis and Lemma 5.3) the constraints in S imply $u_p(h^*) < u_p$, where p is the current player. This has to be the case because otherwise either the returned split would not have been null or $u(h^*) \in \mathbb{U}_\Gamma$. It, however, implies that $u(h^*)$ is not practical for all $\vec{x}$ satisfying the constraints in S.

Finally, we assume Γ is not along the honest history. We know that only utilities that occurred in a $\mathbb{U}_{\Gamma_{|(a)}}$ are candidates to be in $\mathbb{U}_\Gamma$, the others are not. The others are, according to the induction hypothesis, not practical in $\mathbb{U}_{\Gamma_{|(a)}}$ for all $\vec{x}$ satisfying the constraints in S. Therefore, they are also not practical in $\mathbb{U}_\Gamma$ for all $\vec{x}$ satisfying the constraints in S. The ones that are in a $\mathbb{U}_{\Gamma_{|(a)}}$ are practical for all such $\vec{x}$ by Lemma 5.3. Hence, if a utility $u \in \mathbb{U}_{\Gamma_{|(a)}}$ but not in $\mathbb{U}_\Gamma$ according to the algorithm, there has to exist a sibling $\Gamma_{|(a')}$ such that for all their practical utilities $u_{a'}$, the constraints of S together with $u_p \geq u_{a',p}$ are unsatisfiable. Otherwise, either the split would not be null, or u would have been added to $\mathbb{U}_\Gamma$. By Theorem 5.7, u is not practical in Γ for all $\vec{x}$ that satisfy the constraints in S. □

The next lemma reasons about the negative output of `ComputeSP`.

Lemma 5.5. *Given an input instance Π, an honest history h^*, a security property sp and S the set of initial constraints C and a case `case`, if there exists a player group pg such that*

$$\texttt{ComputeSP}(\Pi, h^*, \mathsf{S}, sp, \mathsf{pg}) = (\texttt{false}, \mathit{null}) \ ,$$

then

$$\forall \vec{x}.\ \forall c \in C \cup \texttt{case}.\ c[\vec{x}] \rightarrow \neg sp(\Gamma, h^*)[\vec{x}] \ .$$

Proof. We prove the lemma per security property algorithm. First, we consider security properties weak and weaker immunity: There exists a player group such that `ComputeSP` returns $(\texttt{false}, \mathsf{null})$, iff `ComputeWI` returns $(\texttt{false}, \mathsf{null})$ for pg. We proceed by structural induction. For the base case, we assume Γ is a leaf. Following Algorithm 5.3, one can see in lines 1–9 that $(\texttt{false}, \mathsf{null})$ is returned if $C \cup \texttt{case} \cup u_{\mathsf{pg}} < 0$ is satisfiable but $C \cup \texttt{case} \cup u_{\mathsf{pg}} \geq 0$ is unsatisfiable, where u is the utility of the leaf respectively its real part for weaker immunity. This implies that for all $\vec{x}$ that satisfy $C \cup \texttt{case}$ the inequality $u_{\mathsf{pg}} < 0$ holds. Therefore, $\neg sp(\Gamma, h^*)$ for all such $\vec{x}$.

For the induction step we assume Γ is a branch and by induction hypothesis all subtrees of Γ satisfy the property. For the first case we assume pg is not the current player in Γ. If $(\texttt{false}, \mathsf{null})$ is returned from `ComputeWI`, there had to be a child $\Gamma_{|(a)}$ of Γ such that `ComputeWI` returned `false` in line 14. Applying the induction hypothesis this implies that for all $\vec{x}$ satisfying the constraints in S $\Gamma_{|(a)}$ is not weak(er) immune for pg. Let us now fix such an $\vec{x}$ arbitrarily. Following Theorem 5.5, we know that for values $\vec{x}$ the game Γ is not weak(er) immune for pg. Since $\vec{x}$ was chosen arbitrarily, Γ is not weak(er) immune for pg for all $\vec{x}$ that satisfy the constraints in S.

Secondly, assume pg is the current player and Γ is along h^* and `ComputeWI` returned $(\texttt{false}, \mathsf{null})$. Following Algorithm 5.3 lines 19–21, this can only happen, if $(\texttt{false}, \mathsf{null})$ was returned in line 21. By induction hypothesis this implies that for all $\vec{x}$ satisfying the constraints in S the subgame $\Gamma_{|(a^*)}$ is not weak(er) immune for pg. By Theorem 5.5, it follows that also Γ is not weak(er) immune for pg, for all $\vec{x}$ satisfying the constraints in S.

Lastly, we assume pg is the current player and Γ is not along h^* and `ComputeWI` returned $(\texttt{false}, \mathsf{null})$. According to the algorithm, that implies that all children had returned $(\texttt{false}, \mathsf{null})$ in line 25. By induction hypothesis, this means that all children are not weak(er) immune for pg for all $\vec{x}$ satisfying the constraints in S. Applying Theorem 5.5, it follows that also Γ is not weak(er) immune for pg for all $\vec{x}$ satisfying $C \cup \texttt{case}$.

The property for collusion resilience holds due to the same reasoning as for weak(er) immunity. For practicality, note that $\texttt{ComputeSP}(\Pi, h^*, \mathsf{S}, pr, \mathsf{pg}) = (\texttt{false}, \mathsf{null})$ iff $\texttt{ComputePR}(\Gamma, h^*, \mathsf{S})$ returns $(\emptyset, \mathsf{null})$.

According to Lemma 5.4 $\texttt{ComputePR}(\Gamma, h^*, \mathsf{S}) = (\emptyset, \mathsf{null})$ implies that first Γ has to be along h^* (as otherwise no utility would be practical in Γ for an arbitrary $\vec{x}$ satisfying the constraints in S, which was proven impossible in Lemma 5.1) and second that the honest history is not practical in Γ for all $\vec{x}$ that satisfy the constraints in S. This concludes the proof for practicality and, thus, the lemma. □

The last lemma we need to prove the theorem gives insight into the positive output of the `ComputeSP` function:

Lemma 5.6. *Given an input instance Π, an honest history h^*, a security property sp and a set S of initial constraints C and a case `case`, if for all player groups pg*

$$\texttt{ComputeSP}(\Pi, h^*, \mathsf{S}, sp, \mathsf{pg}) = (\texttt{true}, \mathit{split}) \ ,$$

independent of what split *is, then*

$$\forall \vec{x}.\ \forall c \in C \cup \texttt{case}.\ c[\vec{x}] \rightarrow sp(\Gamma, h^*)[\vec{x}]\ .$$

Proof. Let us fix a player group pg and assume sp is not pr. We again prove the lemma by structural induction on the game tree Γ. For the base case we assume Γ is a leaf and the return value is (true, split). According to Algorithms 5.3 and 5.4, this can only happen if $C \cup \texttt{case}$ together with the negated property inequality of the leaf utility u ($u_{\mathsf{pg}} < 0$, respectively $u_{\mathsf{pg}}(h^*) < u_{\mathsf{pg}}$) is unsat. This implies according to Theorems 5.5 and 5.6, that for all $\vec{x}$ satisfying $C \cup \texttt{case}$ the game Γ does not satisfy the security property for/against pg. The induction step is analogous to that of Lemma 5.5.

For $sp = pr$, we employ Lemma 5.3 to conclude that Γ with honest history h^* is practical for all $\vec{x}$ that satisfy the constraints in S.

We showed that for all $\vec{x}$ that satisfy the constraints in S, Γ with honest history h^* satisfies sp for/against pg. Since pg was chosen arbitrarily, it holds for all player groups. We can further separate the for-all quantification into both sides of the implication. As the player group quantification and the $\vec{x}$ quantification are independent, their order can be switched. By Theorem 5.3, it finally follows that for all $\vec{x}$ that satisfy the constraints in S, the $sp(\Gamma, h^*)$ holds. □

Theorem 5.8 (Correctness of Algorithm 5.1)**.** *The compositional approach to compute the game-theoretic security of an input instance* Π *for honest history* h^* *described in Algorithm 5.1 is sound and complete. That is,* $\texttt{SatisfiesProperty}(\Pi, h^*, sp, \emptyset) = \texttt{true}$ *iff* Π *with honest history* h^* *satisfies the property sp. Otherwise, it returns* false*.*

Proof. We prove direction "⇐" first, by contraposition. That is, we assume the function $\texttt{SatisfiesProperty}(\Pi, h^*, sp, \emptyset)$ returns false and show that Π with honest history h^* does not satisfy the security property sp. From our assumption and Algorithm 5.1, we know that for the return value to be false, there has to exist a set of utility term comparisons case such that

$$\texttt{SatisfiesProperty}(\Pi, h^*, sp, \texttt{case}) = \texttt{false}\ .$$

This implies that there has to exist a player group pg such that $\texttt{ComputeSP}(\Pi, h^*, \mathsf{S}, sp, \mathsf{pg})$ returns (false, null), where S contains the constraints from C, (defined in Π) and case.

We can now apply Lemma 5.5 to conclude that for all values of $\vec{x}$ that satisfy the (satisfiable) set of constraints in S, the game Γ (of Π) with honest history h^* violates security property sp for player group pg. The constraint set case can be extended to a total order $\preceq$ on the utility terms T_u. Thus, by the fact the all $\vec{x}$ that satisfy $C \cup \preceq$ also satisfy $C \cup \texttt{case}$, it follows that for all $\vec{x}$ that satisfy $C \cup \preceq$ security property sp does not hold for player group pg. Using Theorem 5.1, this means Equation (5.9) does not hold. That means exactly $\neg sp(\Pi, h^*)$. Hence, the entire input instance Π violates security property sp in general.

For the other direction "⇒", we first assume that $\texttt{SatisfiesProperty}(\Pi, h^*, sp, \emptyset)$ returns `true`. Following then the Algorithm 5.1, this implies that all final cases returned `true`. Further, all considered cases are pairwise disjoint, and their disjunction is a tautology. Considering one such case `case`, it has to be the case that for all player groups pg

$$\texttt{ComputeSP}(\Pi, h^*, \mathsf{S}, sp, \mathsf{pg}) = (\texttt{true}, \mathsf{split}) \ ,$$

where the value of split is irrelevant and S contains the constraints of C and `case`. According to Lemma 5.6, this implies that for all $\vec{x}$ that satisfy C and `case` that security property sp holds for player group pg, game Γ and honest history h^*. As before, this implies that for all total orders $\preceq$ extending `case`, the security property holds for pg. Further, the disjunction of all total orders that extend `case` is equivalent to `case` itself. Since this result holds for all the considered cases and those cases span the considered universe, it follows that for all total orders $\preceq$ holds that all $\vec{x}$ that satisfy $C \cup \preceq$ satisfy the security property for all player groups. According to Theorem 5.1, follows Equation (5.9) holds. Hence, the input instance Π with honest history h^* satisfies the security property.

Finally, we have to show that `SatisfiesProperty` always returns either `true` or `false`. This is equivalent to proving termination. The `Compute<SP>` functions terminate since they walk through the finite game tree once, and all SMT queries are decidable, as unquantified non-linear real arithmetic is decidable. Further, the function `SatisfiesProperty` splits only on utility comparisons of players/groups of players, which are also finitely many. Hence, the algorithm terminates. □

5.6.3 Extracting Compositional Strategies

The way compositional security analysis in Algorithm 5.1 works, unfortunately, does not immediately provide witness strategies. However, Algorithm 5.1 can still carry around enough information to compute witnesses.

Theorem 5.9 (Weak(er) Immune Strategies)**.** *For a weak(er) immune game Γ, with honest history h^* and total order $\preceq$, strategy σ is honest and weak(er) immune for all $\vec{x}$ satisfying $\preceq$, where*

$$\sigma := (\sigma^{p_1}, \ldots, \sigma^{p_{|N|}}) \ ,$$

and $\sigma^{p_i} \in \mathscr{S}_{p_i}$ is a strategy for player p_i. Strategy σ^{p_i} picks the honest choice along the honest history, whereas at other nodes, where it is p_i's turn, it picks an arbitrary action a that yields a weak(er) immune for p_i subtree after action a.

Proof. Consider $\sigma^{p_i} \in \mathscr{S}_{p_i}$ as in the theorem. This strategy is weak(er) immune for p_i. To show this, consider an arbitrary joint strategy τ. Then $u_{p_i}(\tau_{N-p_i}, \sigma^{p_i}) \geq 0$ (respectively its real part), since along the generated history of $(\tau_{N-p_i}, \sigma^{p_i})$ either it is not player p_i's turn, in which case according to Theorem 5.5, all choices have to be weak(er) immune for p_i, or it is player p_i's turn, in which case we chose a weak(er) immune for p_i action for

σ^{p_i}. Eventually, we thus have to reach a weak(er) immune for p_i leaf. A leaf is weak(er) immune for p_i, iff its (real part of the) utility for p_i is non-negative for all $\vec{x}$ satisfying $\preceq$.

Applying now the proof of Theorem 5.3 for weak(er) immunity, it follows that strategy σ is weak(er) immune for all $\vec{x}$ satisfying $\preceq$. □

Theorem 5.9 is constructive in nature, yielding thus an algorithmic approach for extracting a weak(er) immune strategy. For each player pg, function `ComputeWI` (and `ComputeWERI`) proceeds as follows. If it is their turn after history h, h off h^*, and we found a weak(er) immune choice, we store this action as the choice of a possible weak(er) immune and honest strategy σ. If the game is weak(er) immune for all players, we can simply compute σ by collecting all the stored choices throughout the tree.

Example 5.19. We compute the weak immune strategy of the Market Entry game from Example 5.1 with honest history (n), which was analyzed as in Example 5.13. The strategy σ^M for player M has to choose the honest action n at the root, which is the only choice point for M. The strategy σ^E for player E needs to choose one weaker immune subtree after history (e). Since the subtree after history (e, i) is the only candidate, we set $\sigma^E(e) = i$. The strategy $\sigma = (\sigma^M, \sigma^E)$ is the desired weak immune strategy.

For collusion resilience it is also possible to compute an honest collusion resilient strategy compositionally. However, proving that the constructed strategy is collusion resilient requires more involved reasoning.

Theorem 5.10 (Collusion Resilient Strategies)**.** *For a collusion resilient game Γ, with honest history h^* and total order $\preceq$, strategy σ is honest and collusion resilient for all $\vec{x}$ satisfying $\preceq$, where we proceed top-down and breadth-first, to pick the following action for strategy σ at history $h \in \mathscr{H} \setminus \mathscr{T}$:*

1. *if h is along h^*, pick the honest action a^*: $\sigma(h) = a^*$;*
2. *otherwise, maintain the set of players $S \subseteq N$ that had to deviate from σ to reach h. Then pick an arbitrary action a, for which the subtree $\Gamma_{|(h,a)}$ is collusion resilient against all supersets of S other than N.*

Note that in case (2) of Theorem 5.10, there always exists an action a, such that the subtree $\Gamma_{|(h,a)}$ is collusion resilient against all supersets of S other than N. Theorem 5.10 is also constructive, yielding an algorithmic way to compute a collusion resilient strategy. We can proceed in the same way as for weak immunity in Theorem 5.9: during analysis for each player group pg and each branch, we store whether the branch is collusion resilient against pg. If the game turns out to be collusion resilient, we can collect the choices for σ according to the theorem.

Proof. First, we have to prove that such a strategy always exists, provided that h^* is collusion resilient. We fix an arbitrary total order $\preceq$ and consider only values for $\vec{x}$ that

satisfy $\preceq$. Towards a contradiction, we assume we cannot pick an action according to the theorem at history h. We further assume h is a shortest history with that property. As we can always pick the honest choice along h^*, h has to be off the honest history and for each choice $a \in A(h)$ there has to exist a superset of S (other than N) against which $\Gamma_{|(h,a)}$ is not collusion resilient.

Consider the set S of players who had to deviate from the partially defined strategy σ to reach h. Pick now the last time in h where a player $p \notin S$ has a turn, and call the respective history t. If only players of S ever have turns along h, we define $t := \emptyset$. In any case $\Gamma_{|t}$ is collusion resilient against all supersets of S (other than N) by construction (since we could pick an action according to the theorem to reach t; and if $t = \emptyset$ because Γ is collusion resilient).

From the proof of Theorem 5.3.3 for collusion resilience follows that there exists a strategy σ^S that is collusion resilient against all supersets S' of S, $S' \neq N$.

Consider action $a \in A(h)$, with $\sigma^S(h) = a$. Such an a has to exist since $\Gamma_{|t}$ is a supertree of $\Gamma_{|h}$. By assumption, $\Gamma_{|(h,a)}$ is not collusion resilient against at least one S'. Fix such an S'. Therefore, there exists a strategy $\tau^a \in \mathscr{S}_{|(h,a)}$ such that

$$\sum_{p \in S'} u_p^* < \sum_{p \in S'} u_{|(h,a),p}(\sigma^S_{|(t',a),N-S'}, \tau^a_{S'}) \,, \tag{5.16}$$

where u^* is the honest utility $u(h^*)$ and t' is the suffix of h after t: $(t, t') = h$.

Towards a contradiction, we construct $\tau \in \mathscr{S}_{|t}$ as follows. Let $\tau_{|(t',a)} = \tau^a$, and let τ yield (t', a). The rest can be picked arbitrarily. It can be observed that

$$u_{|(h,a)}(\sigma^S_{|(t',a),N-S'}, \tau^a_{S'}) = u_{|t}(\sigma^S_{N-S'}, \tau_{S'}) \,, \tag{5.17}$$

as only players $p \in S'$ have turns in t', $\sigma^S(h) = a$, $\tau(h) = a$ and $h = (t, t')$. But σ^S is collusion resilient against S' in Γ_t, hence

$$\sum_{p \in S'} u_{|t,p}(\sigma^S_{N-S'}, \tau_{S'}) \leq \sum_{p \in S'} u_p^* \,. \tag{5.18}$$

Finally, Equations (5.16) to (5.18) yield a contradiction.

What remains to be shown is that the constructed strategy σ is collusion resilient and honest in Γ. It yields the honest history h^* by construction. We again prove the collusion resilience by contradiction and assume σ is not cr (but the tree Γ still is collusion resilient, for another strategy). Then, there has to exist a set of players $S \subset N$ and a strategy $\tau \in \mathscr{S}$ such that

$$\sum_{p \in S} u_p(\sigma) = \sum_{p \in S} u_p^* < \sum_{p \in S} u_p(\sigma_{N-S}, \tau_S) \,. \tag{5.19}$$

Let h be the prefix of history $H(\sigma_{N-S}, \tau_S)$ at which for the last time in $H(\sigma_{N-S}, \tau_S)$ an honest player $p = P(h) \notin S$ has a turn. Note that such an h has to exist and has to be

off the honest history h^*. If this were not the case, τ would cause every honest strategy to not be collusion resilient as the right side of inequality in (5.19) would not depend on σ_{N-S} at all, which would imply that h^* is not collusion resilient.

By the construction of σ, we know that $a = \sigma(h)$ leads to a subtree $\Gamma_{|(h,a)}$ that is collusion resilient against the set of deviating players S^d (deviating from σ to h) and all supersets (other than N). Further, $S \supseteq S^d$ is such a superset, since more players could deviate from σ in other parts of the tree. Hence, there exists a strategy $\sigma^S \in \mathscr{S}_{|(h,a)}$ that is collusion resilient against S. Therefore, also for the considered strategy τ

$$\sum_{p \in S} u_p^* \geq \sum_{p \in S} u_{|(h,a),p}(\sigma^S_{N-S}, \tau_{|(h,a),S}) \; . \tag{5.20}$$

Note that after h no more honest players have a turn in $H(\sigma_{N-S}, \tau_S)$, and $(\sigma_{N-S}, \tau_S)(h) = \sigma(h) = a$, because $P(h) \in N - S$. Therefore, $H(\sigma_{N-S}, \tau_S) = (h, a, H(\tau_{|(h,a)}))$. This implies that also no honest player has a turn in $H(\sigma^S_{N-S}, \tau_{|(h,a),S})$, as $H(\sigma^S_{N-S}, \tau_{|(h,a),S}) = H(\tau_{|(h,a)})$. Since the histories align, the strategies also have to yield the same utilities $u(\sigma_{N-S}, \tau_S) = u_{|(h,a)}(\sigma^S_{N-S}, \tau_{|(h,a),S})$. This contradicts Equations (5.19) and (5.20) and shows that σ is an honest and collusion resilient strategy. □

Lastly, we describe how to extract practical strategies in our compositional framework.

Theorem 5.11 (Practical Strategies)**.** *Let Γ be a game tree, $\preceq$ a total order, $h \in \mathscr{H}$ a history. Further, let u be a utility practical (under $\preceq$) in $\Gamma_{|h}$. The following strategy $\sigma^u \in \mathscr{S}_{|h}$ yields utility u and is practical (under $\preceq$).*

1. *For action $a \in A(h)$, where u is practical in $\Gamma_{|(h,a)}$ (under $\preceq$) we set $\sigma^u(h) = a$ and $\sigma^u_{|(a)} = \sigma^{u,a}$, where $\sigma^{u,a} \in \mathscr{S}_{|(h,a)}$ is a practical strategy in $\Gamma_{|(h,a)}$ that yields utility u.*
2. *For all other $a' \in A(h) \setminus \{a\}$ we define $\sigma^u_{|(a')} = \sigma^{u',a'}$, where u' is a utility practical in $\Gamma_{|(h,a')}$, $\sigma^u_{|(a')}$ a practical strategy yielding u' and for all $\vec{x}$ satisfying $\preceq$: $u_{P(h)}[\vec{x}] \geq u'_{P(h)}[\vec{x}]$.*

Similarly to our previous results, Theorem 5.11 provides an algorithmic solution to extract a practical strategy for the honest history. For a game tree Γ with practical honest history h^* and total order $\preceq$, an honest and practical strategy can be computed as follows: bottom-up, for each subtree and each practical utility, a corresponding practical strategy is stored. When proceeding one level up, we do as stated in Theorem 5.11. Along the honest history, the only stored practical strategy is the one for the honest utility $u^* = u(h^*)$ as, ultimately, this is the only one required at the root.

Proof. According to Theorem 5.7, u can only be practical in $\Gamma_{|h}$, if it was practical in at least one child $\Gamma_{|(h,a)}$. For every other child a' there had to exist a utility u' practical in

$\Gamma_{|(h,a')}$ such that $u_{P(h)} \geq u'_{P(h)}$. Hence, strategy σ^u can always be constructed. It further yields utility u (by construction) and is practical since by construction and the proof of Lemma 5.1 no player can at any point of the game deviate profitably. □

5.6.4 Finding Compositional Counterexamples

Counterexamples to the security properties, as defined in Definitions 5.8 to 5.10, serve the important purpose of providing attack vectors and thus pinpointing the weaknesses of a protocol underlying the considered game model. We use the following *pseudo-algorithms* to compute counterexamples compositionally.

Compositional Counterexamples to Weak(er) Immunity. We first store information during Algorithm 5.3: When analyzing the weak(er) immunity for a player p, whenever it is not p's turn and there exists an action leading to a not weak(er) immune subtree (line 14 with split = null in Algorithm 5.3), we store the action, the current history and the player p.

Secondly, after the analysis terminated and the result was not weak(er) immune, we generate a counterexample to the weak(er) immunity of player p by walking through the tree again. Assume the current history is h and we proceed from the root as follows.

- If p is the current player and h is along the honest history, we follow the honest action to a subtree. This is sufficient since an honest p follows the honest history.
- If it is p's turn but h is not along the honest history, all choices had to lead to not weak(er) immune for p subtrees for the current tree to be not weak(er) immune for p. We, therefore, have to follow all choices to compute a counterexample.
- Otherwise, if it is not p's turn, we check our stored data for a not weak(er) immune for p choice a. By construction and using Theorem 5.5, it has to exist. We add it to our partial strategy s_{N-p}, i.e. $s_{N-p}(h) = a$. Then, we continue at history (h, a).
- At a leaf nothing has to be considered. A not weak(er) immune for p leaf leads to a negative (real) utility for p.

According to Theorem 5.5, the steps outlined above provide a player p and a partial strategy s_{N-p} for the other players $N - p$, no matter how the honest p behaves off the honest history. It also yields only negative utilities for p and it thus provides a counterexample to the weak(er) immunity of p and, therefore, a counterexample to the weak immunity of the game with the considered honest history.

Example 5.20. Let us adapt the Market Entry game from Example 5.1 by changing the initial constraint on the variable p to $p < 0$. The honest history (n) is not weak immune for player E, as they get a negative utility $p < 0$ in the honest leaf. We can thus construct the counterexample as follows: starting from the root, it is not E's turn and

the not weak immune choice is (n), so we add the action n to the partial strategy for player M. We then continue at history (n), which is a leaf, so we are done.

Compositional Counterexamples to Collusion Resilience. When analyzing the collusion resilience against a group of players S, whenever it is the turn of one of the players in S and there exists an action leading to a not collusion resilient against S subtree (line 14 with $\mathsf{split} = \mathsf{null}$ in Algorithm 5.4), we store the action, the current history and player group S.

After the analysis terminated and the result was not collusion resilient, we generate a counterexample of the collusion resilience against player group S by walking through the tree again: Starting from the root, we proceed as follows, assuming the current history is h.

- If $P(h) \notin S$ and h is along the honest history, we follow the honest action to the honest subtree. This is sufficient since an honest player $P(h)$ follows the honest history.
- If $P(h) \notin S$ but h is not along the honest history, all choices had to lead to not collusion resilient against S subtrees for the current tree to be not collusion resilient against S. We, therefore, have to follow all choices to compute a counterexample.
- Otherwise, if $P(h) \in S$, we check our stored data for a choice a that is not collusion resilient against S. By construction and Theorem 5.6, it has to exist. We add it to our partial strategy s_S, i.e. $s_S(h) = a$. Then, we continue at history (h, a).
- At a leaf nothing has to be considered. A not collusion resilient against S leaf has a joint utility for S greater than their joint honest utility.

With the same arguments as for weak(er) immunity, we conclude that the generated partial strategy s_S together with player group S is a counterexample to the collusion resilience of game Γ with the considered honest history.

Compositional Counterexamples to Practicality. When analyzing the tree according to Algorithm 5.5, we can only return "false" (i.e. an empty list), together with case split null along the honest history. This is the case exactly when, for at least one sibling, all practical utilities are strictly better for the current player than the honest one (line 15). Before returning, we store the action a leading to said sibling together with the set of its practical utilities $\mathbb{U}(h, a)$. For convenience, we also provide the histories $t \in \mathscr{H}_{|(h,a)}$ to those utilities $u(t) = u \in \mathbb{U}(h, a)$. Using Theorem 5.7, we thus computed a counterexample to practicality.

Remark 6. It is also possible to compute *all* counterexamples to a security property. This can be done by simply storing *all* actions that lead to not weak(er) immune, respectively collusion resilient subtrees in the pseudo-algorithms, for weak(er) immunity and collusion

resilience. For practicality, it requires storing all siblings along the entire honest history, whose practical utilities lead to a better-than-honest utility.

5.7 Experimental Evaluation

We implemented the compositional security approach of Section 5.5 by exploiting its divide-and-conquer reasoning nature from Section 5.6. Our implementation is available online in the CheckMate2.0 tool[2].

Experimental Setup. We evaluated our tool using a machine with 2 AMD EPYC 7502 CPUs clocked at 2.5GHz with 32 cores and 1TB RAM using 16 game-theoretic security benchmarks. Our dataset contains the 15 examples from [RBK+24], which include realistic models of real-world blockchain protocols along with game scenarios of various sizes. Additionally, we detail later in this section one large example, named 4-Player Routing, in order to showcase the impact of interleaved sub- and supertree reasoning. To the best of our knowledge, the only other automated approach for game-theoretic security is the conventional CheckMate framework [RBK+24]. Our experiments also compare CheckMate2.0 to CheckMate.

Experimental Results. Tables 5.1 and 5.2 summarize our experiments. We report both on the results of CheckMate2.0 and CheckMate; the respective columns on times, nodes, and calls in Tables 5.1 and 5.2 detail these comparisons. In particular, the columns "Nodes evaluated" and "Nodes evaluated (reps)" of Table 5.1 indicate the number of game tree nodes visited during the security analysis without and, respectively, with repetitions. The "Calls" column of Table 5.1 shows the number of calls made to the SMT solver while proving the security property listed in column 4.

Experimental Analysis. Table 5.1 demonstrates that the compositional approach of CheckMate2.0 significantly outperforms the non-compositional CheckMate setting in execution time across nearly all benchmarks. The scalability of CheckMate2.0 is especially evident in the Tic Tac Toe benchmark, which involves a substantial 548,946 nodes. In this example, for the properties weak immunity (`wi`), weaker immunity (`weri`), and collusion resilience (`cr`), CheckMate2.0 completes the security analysis in approximately 5 seconds, whereas CheckMate requires between 255 and 287 seconds. When proving practicality (`pr`) of Tic Tac Toe, the conventional CheckMate fails to terminate within 8 hours while CheckMate2.0 succeeds in less than 37 seconds.

In some benchmarks, where a security property is not satisfied, CheckMate2.0 explores significantly fewer nodes, see 3-Player Routing for weak immunity and collusion resilience, the Pirate game for weak immunity, and Auction for weak immunity and collusion resilience.

[2] `https://github.com/apre-group/checkmate/tree/CCS25`

Game	Nodes	Players	Property	Secure yes/no	Time	Nodes evaluated	Nodes evaluated (reps)	Calls
						CheckMate2.0/CheckMate		
Pirate	79	4	wi	n	0.010 / 0.015	10 / 79	10 / 316	5 / 1
$(y,n,n$			cr	n	0.041 / 0.029	79 / 79	622 / 1,106	368 / 4
$n,y,y)$			pr	n	0.036 / 0.049	79 / 79	482 / 79	554 / 8
Auction	92	4	wi	n	0.012 / 0.033	16 / 92	16 / 368	9 / 1
(E,E,I,I)			cr	n	0.018 / 0.030	66 / 92	128 / 1,288	103 / 1
Closing	221	2	wi	y	0.011 / 0.024	20 / 221	22 / 442	16 / 1
(H)			weri	y	0.010 / 0.021	20 / 221	22 / 442	16 / 1
			cr	y	0.012 / 0.023	44 / 221	46 / 442	36 / 1
			pr	n	0.097 / 0.346	221 / 221	568 / 221	1454 / 1
(C_h,S)			wi	y	0.011 / 0.024	33 / 221	36 / 442	25 / 1
			weri	y	0.011 / 0.020	33 / 221	36 / 442	25 / 1
			cr	y	0.013 / 0.023	60 / 221	63 / 442	48 / 1
			pr	y	2.144 / 0.345	221 / 221	14353 / 221	38220 / 1
3-Player	21,688	3	wi	n	0.248 / 0.984	16 / 21,688	16 / 65,064	9 / 1
Routing			weri	y	0.514 / 1.008	7,084 / 21,688	7,570 / 65,064	5,441 / 1
$(S_H,L,L,$			cr	n	0.272 / 1.886	430 / 21,688	474 / 130,128	299 / 1
$U,U)$			pr	n	33.162 / 34.717	21,688 / 21,688	416,156 / 21,688	569,418 / 13
Tic Tac Toe	549,946	2	wi	y	5.276 / 255.368	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
$(CM,RU,LU,$			weri	y	5.256 / 255.600	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
$RD,RM,LM,$			cr	y	5.302 / 286.574	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
$CU,CD,LD)$			pr	y	36.530 / TO	549946 / TO	549946 / TO	527198 / TO

Table 5.1: Selected experimental results of game-theoretic security, using the compositional CheckMate2.0 approach and the non-compositional CheckMate setting of [RBK+24]. A full summary of our experiments is in Table 5.4. Runtimes are given in seconds, with timeout (TO) after 8 hours. For each game, columns 2–3 list the size (tree nodes and game players) of the game from column 1. Column 4 shows the game-theoretic security property we analyzed and (dis)proved, as indicated in column 5. Columns 6–9 present the results of CheckMate2.0 compared to CheckMate, using the slash / sign.

Game	Property	Time (one CE) CheckMate2.0/CheckMate	Time (all CEs) CheckMate2.0/CheckMate
Pirate (y, n, n, n, y, y)	cr	0.041 / 0.039	3.232 / 79.839
Auction (E, E, I, I)	wi	0.012 / 0.048	0.025 / 4.172
	cr	0.018 / 0.066	0.036 / 15.106
3-Player Routing (S_H, L, L, U, U)	wi	0.251 / 1.925	5.909 / 110.716
	cr	0.279 / 5.619	1.657 / 7.815
	pr	33.561 / 46.480	291.236 / 3 033.784

Table 5.2: Selected experiments on counterexample (CE) generation using our CheckMate2.0 approach and the non-compositional CheckMate tool of [RBK+24]. Full details and experiments are given Table 5.5. Runtimes are given in seconds.

We note that CheckMate2.0 requires considerably more SMT-solving calls. Notable examples include the Closing Game (38,220 CheckMate2.0 calls vs. 1 CheckMate call for practicality), 3-Player Routing (546,418 vs. 13 calls for practicality), and Tic Tac Toe (10,694 vs. 1 call for weak(er) immunity and collusion resilience). Despite the higher number of SMT calls in CheckMate2.0, the SMT queries generated by CheckMate2.0 are considerably smaller than those of CheckMate; moreover, CheckMate2.0 calls inhabit a quantifier-free fragment, easing reasoning significantly as reflected in the improved execution times.

In general, CheckMate2.0 analysis may occasionally result also in suboptimal splits, leading to longer execution times. This issue is exemplified in the Closing game when analyzing practicality of the honest history (C_h, S). Additionally, analyzing collusion resilience can sometimes take longer, particularly when more players are involved, for example in the Pirate game. This might be explained by the very large number of colluding groups combined with a small game resulting in many trivial SMT calls compared to CheckMate.

Counterexamples and Strategies. Table 5.2 presents the CheckMate2.0 runtimes to generate counterexamples compared to CheckMate, for selected benchmarks. It reports the execution time required to find one counterexample (for one case split) as well as finding all counterexamples in all cases for violated security properties. The former is useful for quickly identifying scenarios where the property is not met, while the latter proves particularly helpful when revising and refining a protocol. Comprehensive data for all benchmarks can be found in Table 5.5.

The use of compositionality in CheckMate2.0 demonstrates notable improvements in execution time, particularly when retrieving all counterexamples. Additionally, the execution times for compositional analysis with and without counterexample extraction are quite similar, indicating that CheckMate2.0 enables counterexample extraction with minimal overhead. The counterexamples to collusion resilience for the Pirate game show this clearly. While CheckMate2.0 requires slightly more time for property

Game	Property	Time CheckMate2.0/CheckMate
$\text{Splits}_{\text{wi}}$	wi	0.010 / 0.019
(q)	weri	0.010 / 0.018
	cr	0.010 / 0.015
	pr	0.011 / 0.017
$\text{Splits}_{\text{cr}}$	wi	0.011 / 0.018
(n)	weri	0.011 / 0.018
	cr	0.011 / 0.019
	pr	0.011 / 0.018
Market Entry	cr	0.010 / 0.014
(e, i)	pr	0.010 / 0.014
Simplified Closing	wi	0.009 / 0.012
(H)	weri	0.009 / 0.011
	cr	0.009 / 0.012
(C_h, S)	cr	0.010 / 0.012
	pr	0.010 / 0.014
Simplified Routing	weri	0.010 / 0.011
$(S_H, L, L, L, L, U, U, U, U)$	pr	0.010 / 0.012
Auction	weri	0.017 / 0.028
(E, E, I, I)	pr	0.022 / 0.153
Closing	wi	0.011 / 0.025
(H)	weri	0.011 / 0.022
	cr	0.023 / 0.025
(C_h, S)	wi	0.012 / 0.025
	weri	0.011 / 0.021
	cr	0.024 / 0.025
	pr	2.185 / 0.0364
3-Player Routing (S_H, L, L, U, U)	weri	0.539 / 1.163
Unlocking Routing	weri	2.194 / 4.233
(U, U, U, U)	pr	4.241 / 5.718
Tic Tac Toe Concise	wi	0.556 / 7.003
$(CM, LU, RU,$	weri	0.561 / 7.780
$LD, LM, RM,$	cr	1.883 / 8.894
$CU, CD, RD)$	pr	8.644 / 219.689
Tic Tac Toe	wi	5.509 / 276.333
$(CM, RU, LU,$	weri	5.507 / 306.763
$RD, RM, LM,$	cr	18.608 / 347.093
$CU, CD, LD)$	pr	276.719 / TO

Table 5.3: Full experiments on strategy extraction using our CheckMate2.0 approach and the non-compositional CheckMate tool of [RBK+24]. Runtimes are given in seconds, with a timeout (TO) after 8 hours.

analysis compared to CheckMate, we note that the new CheckMate2.0 identifies all counterexamples across all cases in approximately 3 seconds, whereas CheckMate takes almost 80 seconds. Similarly, in the case of the 3-Player Routing game, CheckMate2.0 retrieves all counterexamples for all cases within 291 seconds, while it takes CheckMate

over 3,000 seconds (50 minutes).

Similar benefits of CheckMate2.0 can also be observed for strategy extraction, with details in Table 5.3. The findings closely mirror those observed for counterexamples: Firstly, strategy extraction in the compositional approach outperforms the previous method across nearly all benchmarks. Secondly, the compositional approach incurs almost no additional overhead for strategy extraction, maintaining its overall runtime efficiency. One benchmark that stands out is Tic Tac Toe, where the additional overhead for strategy extraction is clearly noticeable for collusion resilience and practicality. However, strategy extraction for these properties is still achievable within a reasonable time, namely 18 seconds for collusion resilience and 276 seconds for practicality. This represents a significant improvement over the non-compositional approach, which takes 347 seconds for collusion resilience and fails to terminate within the 8-hour time limit for practicality.

Sub– and Supertree Reasoning. One of the most significant contributions of compositional reasoning is that CheckMate2.0 enables analyzing subtrees independently and integrating only their security results in the supertree. This feature of CheckMate2.0 is particularly beneficial in larger models. For instance, the 3-Player Routing and Routing Unlocking benchmarks based on the routing protocol [PD16] are generated using a script, as it is not feasible to model protocols of this size manually. Modeling the routing protocol for 3 players results in a game with 21,688 nodes (3-Player Routing), taking 20 MB on disk.

We next detail a more challenging routing example with 4 players, called *4-Player Routing*, which has 144,342,306 nodes. This example exceeds our 200 GB of allocated disk space, and thus could not even be created fully. However, by leveraging compositionality, we intertwine model generation and analysis, making it possible to discard generated subtrees after the results of security analysis have been obtained. Specifically, during the game generation process, each subtree corresponding to a specific phase of the protocol called the unlocking phase (a total of 1440 subtrees) is analyzed on the fly, with only the results kept. The final outcome, the *4-Player Routing* game, is a supertree with 396 regular nodes and 1440 nodes representing subtrees, or 1,836 nodes in total. The supertree has a size of about 60 MB and in it all subtrees for the unlocking phase have already been solved. This allows us to directly apply CheckMate2.0 to the supertree. Using CheckMate2.0 compositionally, we conclude that 4-Player Routing is weaker immune, but not weak immune, nor collusion resilient, nor practical.

5.8 Related Work and Conclusion

We present the first approach to compositionally analyze the security properties of game-theoretic protocol models. By mapping our work to SMT-based reasoning in combination, we introduce a divide-and-conquer framework to automate compositional reasoning in a

Game	Nodes	Players	Security property	Secure yes/no	Time	Nodes evaluated CheckMate2.0/CheckMate	Nodes evaluated	Calls
$\text{Splits}_{\text{wi}}$	5	2	wi	y	0.010 / 0.018	5	18 / 10	10 / 3
(q)			weri	y	0.010 / 0.018	5	18 / 10	10 / 3
			cr	y	0.010 / 0.015	4 / 5	6 / 10	3 / 1
			pr	y	0.011 / 0.017	5	25 / 5	19 / 3
$\text{Splits}_{\text{cr}}$	5	2	wi	y	0.011 / 0.019	4 / 5	6 / 10	3 / 1
(n)			weri	y	0.011 / 0.019	4 / 5	6 / 10	3 / 1
			cr	y	0.011 / 0.018	5	16 / 10	10 / 3
			pr	y	0.011 / 0.018	5	15 / 5	20 / 3
Market Entry	5	2	wi	n	0.011 / 0.014	5	8 / 10	5 / 1
(e, i)			weri	n	0.010 / 0.014	5	8 / 10	5 / 1
			cr	y	0.010 / 0.014	5	8 / 10	4 / 1
			pr	y	0.010 / 0.015	5	5	2 / 1
G	5	2	wi	n	0.010 / 0.012	5 / 5	28 / 10	18 / 4
(r_A, l_B)			weri	n	0.009 / 0.012	5 / 5	28 / 10	18 / 4
			cr	n	0.008 / 0.010	2 / 5	2 / 10	2 / 1
			pr	n	0.009 / 0.010	5 / 5	9 / 5	5 / 1
Simplified Closing	8	2	wi	y	0.009 / 0.012	8	10 / 16	8 / 1
(H)			weri	y	0.009 / 0.011	8	10 / 16	8 / 1
			cr	y	0.008 / 0.011	7 / 8	9 / 16	6 / 1
			pr	n	0.009 / 0.012	8	8	8 / 1
(C_h, S)			wi	n	0.008 / 0.012	3 / 8	3 / 16	2 / 1
			weri	n	0.008 / 0.011	3 / 8	3 / 16	2 / 1
			cr	y	0.008 / 0.012	8	11 / 16	7 / 1
			pr	y	0.009 / 0.013	8	8	6 / 1
Simplified Routing	17	5	wi	n	0.008 / 0.012	7 / 17	7 / 85	2 / 1
$(S_H, L, L, L, L, U, U, U, U)$			weri	y	0.009 / 0.011	17	77 / 85	28 / 1
			cr	n	0.010 / 0.017	16 / 17	105 / 510	24 / 1
			pr	y	0.009 / 0.012	17	17	8 / 1
Centipede	19	3	wi	n	0.044 / 0.051	19	602 / 57	345 / 18
$(C, C, C, C, C, C, C, C, C, C)$			weri	n	0.033 / 0.052	19	602 / 57	345 / 18
			cr	n	0.044 / 0.038	19	534 / 114	305 / 9
			pr	n	0.011 / 0.028	19	103 / 19	39 / 7
EBOS	31	4	wi	n	0.009 / 0.013	28 / 31	38 / 124	21 / 1
$(Mine, Mine, Mine, Mine)$			weri	n	0.008 / 0.013	28 / 31	38 / 124	21 / 1
			cr	n	0.039 / 0.021	31	476 / 434	304 / 4
			pr	n	0.019 / 0.024	31	167 / 31	184 / 5
Pirate	79	4	wi	n	0.010 / 0.015	10 / 79	10 / 316	5 / 1
(y, n, n, n, y, y)			weri	n	0.009 / 0.016	10 / 79	10 / 316	5 / 1
			cr	n	0.041 / 0.029	79	622 / 1106	368 / 4
			pr	n	0.036 / 0.049	79	482 / 79	554 / 8
Auction	92	4	wi	n	0.012 / 0.033	16 / 92	16 / 368	9 / 1
(E, E, I, I)			weri	y	0.016 / 0.027	90 / 92	229 / 368	162 / 1
			cr	n	0.018 / 0.030	66 / 92	128 / 1,288	103 / 1
			pr	y	0.021 / 0.145	92	92	188 / 1
Closing	221	2	wi	y	0.011 / 0.024	20 / 221	22 / 442	16 / 1
(H)			weri	y	0.010 / 0.021	20 / 221	22 / 442	16 / 1
			cr	y	0.012 / 0.023	44 / 221	46 / 442	36 / 1
			pr	n	0.097 / 0.346	221	568 / 221	1454 / 1
(C_h, S)			wi	y	0.011 / 0.024	33 / 221	36 / 442	25 / 1
			weri	y	0.011 / 0.020	33 / 221	36 / 442	25 / 1
			cr	y	0.013 / 0.023	60 / 221	63 / 442	48 / 1
			pr	y	2.144 / 0.345	221	14353 / 221	38220 / 1
3-Player Routing	21,688	3	wi	n	0.248 / 0.984	16 / 21,688	16 / 65,064	9 / 1
(S_H, L, L, U, U)			weri	y	0.514 / 1.008	7,084 / 21,688	7,570 / 65,064	5,441 / 1
			cr	n	0.272 / 1.886	430 / 21,688	474 / 130,128	299 / 1
			pr	n	33.162 / 34.717	21,688	416,156 / 21,688	569,418 / 13
Unlocking Routing	36,113	5	wi	n	0.621 / 2.121	1,184 / 36,113	1,184 / 180,565	714 / 1
(U, U, U, U)			weri	y	1.525 / 1.625	32,429 / 36,113	55,090 / 180,565	27,897 / 1
			cr	n	0.584 / 15.247	319 / 36,113	373 / 1,083,390	60 / 1
			pr	y	2.848 / 4.382	36,113 / 36,113	36,113 / 36,113	46,636 / 1
Tic Tac Toe Concise	58,748	2	wi	y	0.557 / 6.372	1,345 / 58,748	1,355 / 117,496	698 / 1
$(CM, LU, RU, LD, LM$			weri	y	0.541 / 6.373	1,345 / 58,748	1,355 / 117,496	698 / 1
$RM, CU, CD, RD)$			cr	y	0.543 / 7.352	1,345 / 58,748	1,355 / 117,496	698 / 1
			pr	y	3.937 / 227.807	58,748 / 58,748	58,748 / 58,748	57,250 / 1
Tic Tac Toe	549,946	2	wi	y	5.276 / 255.368	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
$(CM, RU, LU, RD, RM,$			weri	y	5.256 / 255.600	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
$LM, CU, CD, LD)$			cr	y	5.302 / 286.574	18,026 / 549,946	18,036 / 1,099,892	10,694 / 1
			pr	y	36.530 / TO	549,946 / TO	549,946 / TO	527,198 / TO

Table 5.4: Full experimental results of game-theoretic security, using the compositional CheckMate2.0 approach and the non-compositional CheckMate setting of [RBK+24]. Runtimes are given in seconds, with a timeout (TO) after 8 hours. For each game, columns 2–3 list the size (tree nodes and game players) of the game from column 1. Column 4 shows the game-theoretic security property we analyzed and (dis)proved, as indicated in column 5. Columns 6–9 present the results of CheckMate2.0 compared to CheckMate, using the slash / sign.

Game	**Property**	**Time (one CE)** CheckMate2.0/CheckMate	**Time (all CE)** CheckMate2.0/CheckMate
Market Entry	wi	0.010 / 0.016	0.010 / 0.023
(e, i)	weri	0.010 / 0.017	0.009 / 0.019
G	wi	0.010 / 0.013	0.010 / 0.020
(r_A, l_B)	weri	0.010 / 0.014	0.009 / 0.020
	cr	0.008 / 0.011	0.008 / 0.013
	pr	0.009 / 0.016	0.009 / 0.028
Simplified Closing (H)	pr	0.009 / 0.019	0.008 / 0.019
(C_h, S)	wi	0.009 / 0.014	0.008 / 0.016
	weri	0.009 / 0.014	0.008 / 0.014
Simplified Routing	wi	0.009 / 0.014	0.010 / 0.033
$(S_H, L, L, L, L, U, U, U, U)$	cr	0.010 / 0.023	0.016 / 0.096
Centipede	wi	0.046 / 0.049	0.080 / 0.495
$(C, C, C, C, C, C, C, C, C, C)$	weri	0.034 / 0.050	0.061 / 0.495
	cr	0.045 / 0.047	0.078 / 0.538
	pr	0.012 / 0.062	0.022 / 0.400
EBOS	wi	0.010 / 0.015	0.011 / 0.058
$(Mine, Mine, Mine, Mine)$	weri	0.010 / 0.015	0.010 / 0.057
	cr	0.040 / 0.028	0.057 / 10.760
	pr	0.020 / 0.032	0.021 / 0.032
Pirate	wi	0.010 / 0.020	0.157 / 9.465
(y, n, n, n, y, y)	weri	0.009 / 0.020	0.157 / 9.495
	cr	0.041 / 0.039	3.232 / 79.839
	pr	0.037 / 0.064	7.414 / 35.227
Auction	wi	0.012 / 0.048	0.025 / 4.172
(E, E, I, I)	cr	0.018 / 0.066	0.036 / 15.106
Closing (H)	pr	0.096 / 0.650	2.204 / 8.846
3-Player Routing	wi	0.251 / 1.925	5.909 / 110.716
(S_H, L, L, U, U)	cr	0.279 / 5.619	1.657 / 7.815
	pr	33.561 / 46.480	291.236 / 3 033.784
Unlocking Routing	wi	0.602 / 5.219	2.090 / 1 988.997
(U, U, U, U)	cr	0.564 / 116.906	3.562 / error

Table 5.5: Full experiments on counterexample (CE) generation using our CheckMate2.0 approach and the non-compositional CheckMate tool of [RBK+24]. Runtimes are given in seconds; *error* means we encountered an exception thrown from CheckMate's Z3 backend.

sound and complete manner. Our experiments clearly showcase scalability improvements, especially for real-world protocols with millions of nodes/actions.

Our compositional approach is a strong enhancement over the non-compositional setting of [BKK+23a]. We not only improve practical usage but also provide a sound and complete way to split and combine game-theoretic properties of subgames/supergames. Compared to [BKK+23a], we minimize the use of SMT solving by applying it only over game leaves.

Compositional game theory, without considering game-theoretic security, is also addressed in [GHWZ18, GKLNF20, BHZ23]. Here, so-called open games are introduced to represent games played relative to a given environment. Open games are, however, restricted to constant numeric utilities and assuming rational behavior of players. Unlike these works, we work with symbolic utilities and capture honest/rational behavior, and thus security,

in games.

Related to compositional verification, [WCN+21] presents compositional analysis of smart contracts. Instead of verifying a smart contract relative to all users, a few representative users are chosen, thereby avoiding intractability due to state explosion. While game-theoretic security is not addressed in [WCN+21], program verification and synthesis are worthy approaches to be further considered in our future work.

Importantly, (automatically) synthesizing game models from the protocol's definition, respectively source code in the case of smart contracts, is a challenge we aim to address in the future. Allowing infinite games and modeling game actions impacted by external factors are other tasks for future work, allowing us to model uncontrollable protocol effects, such as price changes.

CHAPTER 6

Further Reasoning about Implementation Security

This chapter is based on article [ERI+21b]:

Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up Smart Transitions. arXiv preprint, 2021.

This article is an extended version of the publication [ERI+21a]:

Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up Smart Transitions. In Proceedings of 33rd International Conference on Computer Aided Verification, pages 317–340, Cham, Switzerland, 2021.

6.1 Problem Statement

A basic challenge in smart contract verification is how to express the functional correctness of transactions, such as currency minting or transferring between accounts. Typically, the correctness of such a transaction can be verified by proving that the transaction leaves the sum of certain account balances unchanged.

Consider, for example, the task of minting an unbounded number of tokens in the simplified ERC-20 token standard of the Ethereum community [VB15], as illustrated in Figure 6.1[1]. This example deposits the minted amount (`n`) into the receiver's address (`a`) and we need to ensure that the `mint` operation *only* changed the balance of the receiver. To do so, in addition to (i) proving that the balance of the receiver has been increased by `n`, we also need to verify that (ii) the account balance of every user address $\mathtt{a}'$ different

[1]The `old-` prefix denotes the value of a function before the `mint` transition, and the `new-` prefix denotes the value afterwards.

```
a: Address
n: Nat
─────────────────────────────────────────────────────────────────
mint(a,n)
─────────────────────────────────────────────────────────────────
# Post-conditions
assert new-bal(a) = old-bal(a) + n                                  #(i)
for each Address a′ ≠ a:                                            #(ii)
    assert new-bal(a′) = old-bal(a′)
assert new-sum() = old-sum() + n                                    #(iii)
```

Figure 6.1: Minting n Tokens in ERC-20.

than `a` has not been changed during the `mint` operation and that (iii) the `sum` of all balances changed exactly by the amount that was minted. The validity of these three requirements (i)-(iii), formulated as the post-conditions of Figure 6.1, imply its functional correctness.

Surprisingly, proving formulas similar to the post-conditions of Figure 6.1 is challenging for state-of-the-art automated reasoners, such as SMT solvers [DMB08, BCD+11, DDM06b] and first-order provers [KV13, Eme90, WDF+09]: it requires reasoning that links local changes of the receiver (`a`) with a global state capturing the `sum` of all balances, as well as constructing that global state as an aggregate of an unbounded but finite number of `Address` balances. Moreover, our encoding of the problem uses discrete coins that are minted and deposited, whose number is unbounded but finite as well.

In this work, we address verification challenges of software transactions with aggregate properties, such as preservation of sums by transitions that manipulate low-level, individual entities. Such properties are best expressed in higher-order logic, hindering the use of existing automated reasoners for proving them. To overcome such a reasoning limitation, we introduce *Sum Logic* (SL) as a generalization of first-order logic, in particular of Presburger arithmetic. Previous works [Lib99, Vää97, Ete97] have also introduced extensions of first-order logic with aggregates by counting quantifiers or generalized quantifiers. In Sum Logic (SL) we only consider the special case of integer sums over uninterpreted functions, allowing us to formalize SL properties with and about unbounded sums, in particular, sums of account balances, without higher-order operations (Section 6.3). We prove the decidability of one of our SL extensions and the undecidability of a slightly richer one (Section 6.4). Given previous results [Lib99], our undecidability result is not surprising. In contrast, what may be unexpected is our decidability result and the fact that we can use our first-order fragment for a convenient and practical new way to verify the correctness of smart contracts.

We further introduce first-order encodings which enable automated reasoning over software transactions with summations in SL (Section 6.5). Unlike [BREO+19], where SMT-specific extensions supporting higher-order reasoning have been introduced, the logical

encodings we propose allow one to use existing reasoners without any modification. We are not restricted to SMT reasoning, but can also leverage generic automated reasoners, such as first-order theorem provers, supporting first-order logic. We believe our results ease applying automated reasoning to smart contract verification even for non-experts.

We demonstrate the practical applicability of our results by using SMT solvers and first-order provers for validating the correctness of common financial transitions appearing in *smart contracts* (Section 6.6). We refer to these transitions as *smart transitions*. We encode SL into pure first-order logic by adding another sort that represents the tokens of the cryptocurrency themselves (which we dub "coins").

Although the encodings of Section 6.5 do not translate to our decidable SL fragment from Section 6.4, our experimental results show that automated reasoning engines can handle them consistently and fast. The decidability results of Section 6.5 set the boundaries for what one can expect to achieve, while our experiments from Section 6.5 demonstrate that the unknown middle-ground can still be automated.

While our work is mainly motivated by smart contract verification, our results can be used for arbitrary software transactions implementing sum/aggregate properties. Further, when compared to the smart contract verification framework of [WLC+19], we note that we are not restricted to proving the correctness of smart contracts as finite-state machines, but can deal with semantic properties expressing financial transactions in smart contracts, such as currency minting/transfers.

While ghost variable approaches [HJ19] can reason about changes to the global state (the sum), our approach allows the verifier to specify only the local changes and automatically prove the impact on the global state.

Contributions. In summary, we make the following contributions:

- We present a generalization to Presburger arithmetic (SL, in Section 6.3) that allows expressing properties about summations. We show how we can formalize verification problems of smart contracts in SL.
- We discuss the decidability problem of checking validity of SL formulas (Section 6.4): we prove that it is undecidable in the general case, but also that there exists a small decidable fragment.
- We show different encodings of SL to first-order logic (Section 6.5). To this end, we consider theory-specific reasoning and variations of SL, for example by replacing non-negative integer reasoning with term algebra properties.
- We evaluate our results with SMT solvers and first-order theorem provers, by using 31 new benchmarks encoding smart transitions and their properties (Section 6.6). Our experiments demonstrate the applicability of our results within automated reasoning, in a fully automated manner, without any user guidance.

6.2 Preliminaries

We consider many-sorted first-order logic (FOL) with equality, defined in the standard way. The equality symbol is denoted by $\approx$.

We denote by $\mathrm{STRUCT}\,[\Sigma]$ the *set of all structures* for the vocabulary Σ. A structure $\mathcal{A} \in \mathrm{STRUCT}\,[\Sigma]$ is a pair $(\mathcal{D}, \mathcal{I})$, where for each sort $\mathbf{s}$, its domain in $\mathcal{A}$ is $\mathcal{D}(\mathbf{s})$, and for each symbol S, its interpretation in $\mathcal{A}$ is $\mathcal{I}(S)$. Note that *models* of a formula φ over a vocabulary Σ are structures $\mathcal{A} \in \mathrm{STRUCT}\,[\Sigma]$.

A *first-order theory* is a set of first-order formulas closed under logical consequence. We will consider the first-order theory of the natural numbers with addition. This is Presburger arithmetic (PA) which is of course, decidable [Pre29].

We write $\mathbb{N}$ to denote the set of natural numbers. We consider $0 \in \mathbb{N}$ and write $\mathbb{N}^+$ to explicitly exclude 0 from $\mathbb{N}$. The vocabulary of PA is $\Sigma_{\text{Presburger}} = (0, 1, c_1, \ldots, c_l, +^2)$, with all constants $0, 1, c_i$ of sort `Nat`. A structure $\mathcal{A} = (\mathcal{D}, \mathcal{I}) \in \mathrm{STRUCT}\,[\Sigma_{\text{Presburger}}]$ is called a *Standard Model of Arithmetic* when $\mathcal{D}(\texttt{Nat}) = \mathbb{N}$ and $+^2$ is interpreted as the standard binary addition $+$ function over the naturals. The vocabulary $\Sigma_{\text{Presburger}}$ can be extended with a total order relation, yielding $\Sigma^*_{\text{Presburger}} = (0, 1, +^2, \leq^2)$, where $\leq^2$ is interpreted as the binary relation $\leq$ in Standard Models of Arithmetic.

6.3 Sum Logic (SL)

We now define *Sum Logic* (SL) as a generalization of Presburger arithmetic, extending Presburger arithmetic with unbounded sums. SL is motivated by applications of financial transactions over cryptocurrencies in smart contracts. Smart contracts are decentralized computer programs executed on a blockchain-based system, as explained in [SEM18]. Among other tasks, they automate financial transactions such as transferring and minting money. We refer to these transactions as *smart transitions*. The aim of this work and SL in particular is to express and reason about the post-conditions of smart transitions similar to Figure 6.1.

SL expresses smart transition relations among sums of accounts of various kinds, e.g., at different banks, times, etc. Each such kind, j, is modeled by an uninterpreted function symbol, b_j, where $b_j(a)$ denotes the balance of a's account of kind j, and a constant symbol s_j, which denotes the sum of all outputs of b_j. As such, our SL generalizes Presburger arithmetic with (i) a sort `Address` corresponding to the (unbounded) set of account *addresses*; (ii) *balance* functions b_j mapping account addresses from `Address` to account values of sort `Nat`; and (iii) *sum constants* s_j of sort `Nat` capturing the total sum of all account balances represented by b_j. Formally, the vocabulary of SL is defined as follows.

Definition 6.1 (SL Vocabulary)**.** *Let*

$$\Sigma^{l,m,d}_{+,\leq} = \left(a_1, \ldots, a_l, b^1_1, \ldots, b^1_m, c_1, \ldots, c_d, s_1, \ldots, s_m, 0, 1, +^2, \leq^2\right)$$

Function	**Encoding in** SL	**Reference in ERC-20**
`sum`	s or s'	totalSupply
`bal`(a)	$b(a)$ or $b'(a)$	balanceOf
`mint`(a, v)	$b'(a) \approx b(a) + v$	transfer
`transferFrom`(f, t, v)	$b'(t) \approx b(t) + v \wedge b(f) \approx b'(f) + v$	transferFrom

Table 6.1: ERC-20 Token Standard

be a sorted first-order vocabulary of *SL over sorts* {`Address`, `Nat`}*, where*

- *(Addresses) The constants* $a_1, \ldots, a_l$ *are of sort* `Address`*;*
- *(Balance functions)* $b^1_1, \ldots, b^1_m$ *are unary function symbols from* `Address` *to* `Nat`*;*
- *(Constants and Sums) The constants* $c_1, \ldots, c_d, s_1, \ldots, s_m$ *and* $0, 1$ *are of sort* `Nat`*;*
- $+^2$ *is a binary function* `Nat` $\times$ `Nat` $\to$ `Nat`*;*
- $\leq^2$ *is a binary relation over* `Nat` $\times$ `Nat`*.*

In what follows, when the cardinalities in an SL vocabulary are clear from context, we simply write Σ instead of $\Sigma^{l,m,d}_{+,\leq}$. Further, by $\Sigma^{l,m,d}_{\not{+},\not{\leq}}$ we denote the sub-vocabulary where the crossed-out symbols are not available. Note that even when addition is not available, we still allow writing numerals larger than 1.

We restrict ourselves to *universal sentences* over an SL vocabulary, with quantification only over the `Address` sort.

We now extend the Tarskian semantics of first-order logic to ensure that the sum constants of an SL vocabulary $(s_1, \ldots, s_m)$ are equal to the sum of outputs of their associated balance functions (b_j for each s_j) over the respective entire domains of sort `Address`.

Let Σ be an SL vocabulary. An SL structure $\mathcal{A} = (\mathcal{D}, \mathcal{I}) \in \text{STRUCT}\,[\Sigma]$ representing a model for an SL formula φ is called an SL *model* iff

$$\mathcal{I}(s_j) = \sum_{a \in \mathcal{D}(\texttt{Address})} [\mathcal{I}(b_j)]\,(a), \quad \text{for each } 1 \leq j \leq m. \qquad \text{(Sum Property)}$$

We write $\mathcal{A} \vDash_{\text{SL}} \varphi$ to mean that $\mathcal{A}$ is an SL model of φ. When it is clear from context, we simply write $\mathcal{A} \vDash \varphi$.

Example 6.1 (Encoding ERC-20 in SL)**.** As a use case of SL, we showcase the encoding of the ERC-20 token standard of the Ethereum community [VB15] in SL. To this end, we consider an SL vocabulary $\Sigma^{l,2,d}$. We respectively denote the balance functions and their associated sums as b, b', s, s' in the SL structure over $\Sigma^{l,2,d}$. The resulting instance of SL can then be used to encode ERC-20 operations/smart transitions as SL formulas,

as shown in Table 6.1. Using this encoding, the post-condition of Figure 6.1 is expressed as the SL formula

$$b'(a) \approx b(a) + n \ \wedge \ \forall a' \not\approx a . b'(a') \approx b(a') \ \wedge \ s' \approx s + n \tag{6.1}$$

formalizing the correctness of the smart transition of minting n tokens in Figure 6.1. In the applied verification examples in Section 6.6, rather than verifying the low-level implementation of built-in functions such as $\texttt{mint}_n$, we assume their correctness by including suitable axioms.

6.4 Decidability of SL

We consider the decidability problem of verifying formulas in SL. We show that when there are several function symbols b_j to sum over, the satisfiability problem for SL becomes undecidable. We first present, however, a useful decidable fragment of SL[2].

6.4.1 A Decidable Fragment of SL

We prove decidability for a fragment of SL, which we call the $(l, 1, d)$-FRAG fragment of SL (Theorem 6.4). To do so, we reduce the fragment to Presburger arithmetic, by using regular Presburger constructs to encode SL extensions, that is the uninterpreted functions and sum constants of SL.

The first step of our reduction proof is to consider distinct models, which are models where the `Address` constants a_i represent distinct elements in the domain $\mathcal{D}(\texttt{Address})$. While this restriction is somewhat unnatural, we show that for each vocabulary and formula that has a model, there exists an equisatisfiable formula over a different vocabulary that has a *distinct* model (Theorem 6.1). The crux of our decidability proof is then proving that $(l, 1, d)$-FRAG has *small* `Address` *space*: given a formula φ, if it is satisfiable, then there exists a model where $|\mathcal{D}(\texttt{Address})| \leq \kappa(|\varphi|)$, $|\varphi|$ is the length of φ, and $\kappa(.)$ is some computable function (Theorem 6.3)[3].

Distinct Models. An SL structure $\mathcal{A}$ is considered *distinct* when the l `Address` constants represent l distinct elements in $\mathcal{D}(\texttt{Address})$. I.e.,

$$|\{\mathcal{I}(a_1), \ldots, \mathcal{I}(a_l)\}| = l \, .$$

Since each SL model induces an equivalence relation over the `Address` constants, we consider partitions P over $\{a_1, \ldots, a_l\}$. For each possible partition P we define a transformation of terms and formulas $\mathcal{T}_P$ that substitutes equivalent `Address` constants with a single `Address` constant. The resulting formulas are defined over a vocabulary that has $|P|$ `Address` constants. We show that given an SL formula φ, if φ has a

[2]The proofs of the results in this section are given in [ERI+21b].

[3]The function $\kappa(.)$ is defined per decidable fragment of SL, and not per formula.

model, we can always find a partition P such that each of its classes corresponds to an equivalence class induced by that model.

Theorem 6.1 (Distinct Models)**.** *Let φ be an SL formula over Σ, then φ has a model iff there exists a partition P of $\{a_1, \ldots, a_l\}$ such that $\mathcal{T}_P(\varphi)$ has a* distinct *model.*

Small `Address` Space. In order to construct a reduction to Presburger arithmetic, we bound the size of the `Address` sort. For a fragment of SL to be decidable, we therefore need a way to bound its models upfront. We formalize this requirement as follows.

Definition 6.2 (Small `Address` Space)**.** *Let FRAG be some fragment of SL over vocabulary $\Sigma = \Sigma_{+,\leq}^{l,m,d}$. FRAG is said to have* small `Address` space *if there exists a computable function $\kappa_\Sigma(.)$, such that for any SL formula $\varphi \in$ FRAG, φ has a distinct model iff φ has a distinct model $\mathcal{A} = (\mathcal{D}, \mathcal{I})$ with* small `Address` space*, where $|\mathcal{D}(\texttt{Address})| \leq \kappa_\Sigma(|\varphi|)$.*

We call $\kappa_\Sigma(.)$ the bound function *of FRAG; when the vocabulary is clear from context we simply write $\kappa(.)$.*

One instance of a fragment (or rather, family of fragments) that satisfies this property is the $(l, 1, d)$-FRAG fragment: the simple case of a *single* uninterpreted "balance" function (and its associated sum constant), further restricted by removing the binary function $+$ and the binary relation $\leq$. Therefore, we derive the following theorem:

Theorem 6.2 (Small `Address` Space of $(l, 1, d)$-FRAG)**.** *For any l, d, it holds $(l, 1, d)$-FRAG, the fragment of SL formulas over the SL vocabulary*

$$\Sigma_{\not+,\not\leq}^{l,1,d} = \left(a_1, \ldots, a_l, b^1, c_1, \ldots, c_d, s, 0, 1\right) ,$$

has small `Address` space *with bound function $\kappa(x) = l + x + 1$.*

An attempt to trivially extend Theorem 6.2 for a fragment of SL with two balance functions falls apart in a few places, but most importantly when comparing balances to the sum of a different balance function. In Section 6.4.2 we show that these comparisons are essential for proving our undecidability result in SL.

Presburger Reduction. For showing decidability of some FRAG fragment of SL, we describe a Turing reduction to pure Presburger arithmetic. We introduce a transformation $\tau(.)$ of formulas in SL into formulas in Presburger arithmetic. It maps universal quantifiers to disjunctions, and sums to explicit addition of all balances. In addition, we define an auxiliary formula $\eta(\varphi)$, which ensures only valid addresses are considered, and that invalid addresses have zero balances. The formal definitions of $\tau(.)$ and $\eta(\varphi)$ can be found in [ERI+21b]. By relying on the properties of *distinctness* and *small* `Address` *space* we get the following results.

Theorem 6.3 (Presburger Reduction)**.** *An SL formula φ has a* distinct*, SL model with small* `Address` *space iff $\tau(\varphi) \wedge \eta(\varphi)$ has a Standard Model of Arithmetic.*

Theorem 6.4 (SL Decidability)**.** *Let FRAG be a fragment of SL that has* small `Address` space*, as defined in Definition 6.2. Then, FRAG is decidable.*

Proof of Theorem 6.4. Let φ be a formula in FRAG. Then φ has an SL model iff for some partition P of $\{a_1, \ldots, a_l\}$, $\mathcal{T}_P(\varphi)$ has a *distinct* SL model. For any P, the formula $\mathcal{T}_P(\varphi)$ is in FRAG, therefore $\mathcal{T}_P(\varphi)$ has a *distinct* SL model iff it has a *distinct* SL model with *small* `Address` *space.*

From Theorem 6.3, we get that for any P, $\varphi_P \triangleq \mathcal{T}_P(\varphi)$ has a *distinct* SL model iff $\tau(\varphi_P) \wedge \eta(\varphi_P)$ has a Standard Model of Arithmetic. By using the PA decision procedure as an oracle, we obtain the following *decision procedure for a* FRAG *formula* φ:

- For each possible partition P of $\{a_1, \ldots, a_l\}$, let $\varphi_P = \mathcal{T}_P(\varphi)$;
- Using a PA decision procedure, check whether $\tau(\varphi_P) \wedge \eta(\varphi_P)$ has a model for each P;
- If a model for some partition P was found, the formula φ_P has a *distinct* SL model, and therefore φ has SL model;
- Otherwise, there is no *distinct* SL model for any partition P, and therefore there is no SL model for φ.

□

Remark 7. Our decision procedure for Theorem 6.4 requires B_l Presburger queries, where B_l is Bell's number for all possible partitions of a set of size l.

Using Theorem 6.4 and Theorem 6.2, we then obtain the following result.

Corollary 6.1. $(l, 1, d)$*-FRAG is decidable.*

6.4.2 SL Undecidability

We now show that simple extensions of our decidable $(l, 1, d)$-FRAG fragment lose its decidability (Theorem 6.5). For doing so, we encode the halting problem of a two-counter machine using SL with 3 balance functions, thereby proving that the resulting SL fragment is undecidable.

Consider a two-counter machine, whose transitions are encoded by the Presburger formula $\pi(c_1, c_2, p, c_1', c_2', p')$ with 6 free variables: 2 for each of the three registers, one of which is the program counter (PC). We assume w.l.o.g. that all three registers are within $\mathbb{N}^+$, allowing us to use addresses with a zero balance as a special "separator". In addition, we assume that the program counter is 1 at the start of the execution, and that there exists a single halting statement at line H. That is, the two-counter machine halts iff the PC is equal to H.

	Address	l(Address)	c(Address)	g(Address)
Time-step #0		0	1	0
		1	1	c_1 at #0
		2	1	c_2 at #0
	a_0	3	1	PC at $\#0 = 1$
	⋮	⋮	⋮	⋮
Time-step #i	x_1	$4i$	1	0
	x_2	$4i+1$	1	c_1 at #i
	x_3	$4i+2$	1	c_2 at #i
	x_4	$4i+3$	1	PC at #i
Time-step #$(i+1)$	x_5	$4i+4$	1	0
	x_6	$4i+5$	1	c_1 at #$(i+1)$
	x_7	$4i+6$	1	c_2 at #$(i+1)$
	x_8	$4i+7$	1	PC at #$(i+1)$
	⋮	⋮	⋮	⋮
Time-step #$n = \frac{s_c}{4} - 1$		$s_c - 4$	1	0
		$s_c - 3$	1	c_1 at #n
		$s_c - 2$	1	c_2 at #n
	a_1	$s_c - 1$	1	PC at $\#n = H$

Table 6.2: Transition System of a 2-Counter Machine, Array View.

Reduction Setting. We have 4 `Address` elements for each time-step, 3 of them hold one register each, and one is used to separate each group of `Address` elements (see Table 6.2). We have 3 uninterpreted functions from `Address` to `Nat` ("balances"). For readability we denote these functions as c, l, g (instead of b_1, b_2, b_3) and their respective sums as s_c, s_l, s_g:

1. Function c: Cardinality function, used to force size constraints. We set its value for all addresses to be 1, and therefore the number of addresses is s_c.
2. Function l: Labeling function, to order the time-steps. We choose one element to have a maximal value of $s_c - 1$ and ensure that l is injective. This means that the values of l are distinctly $[0, s_c - 1]$.
3. Function g: General purpose function, which holds either one of the registers or 0 to mark the `Address` element as a separating one.

Each group representing a time-step is a 4 `Address` element, ordered as follows:

1. First, a separating `Address` element x (where $g(x)$ is 0).
2. Then, the two general-purpose counters.

3. Lastly, the program counter.

In addition we have 2 `Address` constants, a_0 and a_1 which represent the PC value at the start and at the end of the execution. The element a_1 also holds the maximal value of l, that is, $l(a_1) + 1 \approx s_c$. Further, a_0 holds the fourth-minimal value, since it is the last element of the first group, and each group has four elements.

Formalization Using a Two-Counter Machine. We now formalize our reduction, proving undecidability of SL.

(i) We impose an injective labeling

$$\varphi_1 = \forall x, y.\, (l(x) \approx l(y)) \rightarrow (x \approx y)$$

(ii) We next formalize properties over the program counter PC. The `Address` constant that represents the program counter PC value of the last time-step is set to have the maximal labeling, that is

$$\varphi_2 = \forall x. l(x) \leq l(a_1)$$

Further, the `Address` constant that represents the PC value of the first time-step has the fourth labeling, hence

$$\varphi_3 = l(a_0) \approx 3$$

Finally, the first and last values of the program counter are respectively 1 and H, that is

$$\varphi_4 = g(a_0) \approx 1 \wedge g(a_1) \approx H$$

(iii) We express *cardinality constraints* ensuring that there are as many `Address` elements as the labeling of the last `Address` constant $(a_1) + 1$. We assert

$$\varphi_5 = (s_c \approx l(a_1) + 1) \wedge \forall x.\, (c(x) \approx 1)$$

(iv) We encode the transitions of the two-counter machine, as follows. For every 8 `Address` elements, if they represent two sequential time-steps, then the formula for the transitions of the two-counter machine is valid for the registers it holds.

$$\begin{aligned}\varphi_6 = \forall x_1, \ldots, x_8.\, &(F1 \wedge F2 \wedge F3) \\ &\rightarrow \pi\,(g(x_2), g(x_3), g(x_4), g(x_6), g(x_7), g(x_8))\end{aligned}$$

where the conjunction $F1 \wedge F2 \wedge F3$ expresses that $x_1, \ldots, x_8$ are two sequential time-steps, with $F1$, $F2$ and $F3$ defined as below. In particular, $F1$, $F2$ and $F3$ formalize that $x_1, \ldots, x_8$ have sequential labeling, starting with one zero-valued `Address` element ("separator") and continuing with 3 non-zero elements, as follows:

- Sequential:

$$l(x_2) \approx l(x_1) + 1 \wedge \cdots \wedge l(x_8) \approx l(x_7) + 1 \qquad \text{(F1)}$$

- Time-steps:

$$g(x_1) \approx 0 \wedge g(x_2) > 0 \wedge g(x_3) > 0 \wedge g(x_4) > 0\ , \tag{F2}$$
$$g(x_5) \approx 0 \wedge g(x_6) > 0 \wedge g(x_7) > 0 \wedge g(x_8) > 0 \tag{F3}$$

Based on the above formalization, the formula $\varphi = \varphi_1 \wedge \cdots \wedge \varphi_6$ is satisfiable iff the two-counter machine halts within a finite number of time steps (and the exact amount would be given by $\frac{s_c}{4}$). Since the halting problem for two-counter machines is undecidable, our SL, already with 3 uninterpreted functions and their associated sums, is also undecidable.

Theorem 6.5. *For any $l \geq 2, m \geq 3$ and d, any fragment of SL over $\Sigma_{+,\leq}^{l,m,d}$ is undecidable.*

Remark 8. Note that in the above formalization the only use of associated sums comes from expressing the size of the set of `Address` elements. As for our uninterpreted function $c(.)$ we have $\forall x.c(x) \approx 1$, its sum s_c is thus the number of addresses. Hence, we can encode the halting problem for two-counter machines in an almost identical way to the encoding presented here, using a generalization of PA with two uninterpreted functions for $l(.)$ and $g(.)$, and a *size operation* replacing $c(.)$ and its associated sum.

6.5 SL Encodings of Smart Transitions

The definition of SL models in Sections 6.3 and 6.4 ensured that the summation constants s_j were respectively equal to the actual summation of all balances $b_j(.)$. In this section, we address the challenge to formalize relations between s_j and $b_j(.)$ in a way that the resulting encodings can be expressed in the logical frameworks of automated reasoners, in particular of SMT solvers and first-order theorem provers.

In what follows, we consider a single transaction or one time-step of multiple transactions over $s_j, b_j(.)$. We refer to such transitions as *smart transitions*. Smart transitions are common in smart contracts, expressing, for example, the minting and/or transferring of some coins, as evidenced in Figure 6.1 and discussed later.

Based on Section 6.3, our smart transitions are encoded in the $\Sigma^{l,2,d}$ fragment of SL. Note, however, that neither decidability nor undecidability of this fragment is implied by Theorem 6.4, nor Theorem 6.5. In this section, we show that our SL encoding of smart transitions is expressible in first-order logic. We first introduce a sound, *implicit* SL *encoding*, by "hiding" away sum semantics and using invariant relations over smart transitions (Section 6.5.1). This encoding does not allow us to directly assert the values of any balance or sum, but we can prove that this implicit encoding is complete, relative to a translation function (Section 6.5.2).

By further restricting our implicit SL encoding to this relative complete setting, we consider counting properties to explicitly reason with balances and directly express verification conditions with unbounded sums on s_j and $b_j(.)$. This is shown in Section 6.5.3, and we evaluate different variants of the *explicit* SL *encoding* in Section 6.6, showcasing their practical use and relevance within automated reasoning.

To directly present our SL encodings and results in the smart contract domain, in what follows, we rely on the notation of Table 6.1. As such, we respectively denote b, b' by `old-bal`, `new-bal` and write `old-sum`, `new-sum` for s, s'. As already discussed in Figure 6.1, the prefixes `old-` and `new-` refer to the entire state expressed in the encoding before and after the smart transition. We explicitly indicate this state using `old-world`, `new-world` respectively. The non-prefixed versions `bal` and `sum` are stand-ins for *both* the `old-` and `new-` versions — Figure 6.2 illustrates our setting for the smart transition of minting one coin.

With this SL notation at hand, we are thus interested in finding first-order formulas that verify smart transition relations between `old-sum` and `new-sum`, given the relation between `old-bal` and `new-bal`. In this work, we mainly focus on the smart transitions of minting and transferring money, yet our results could be used in the context of other financial transactions/software transitions over unbounded sums.

Example 6.2. In the case of minting n coins in Figure 6.1, we require formulas that (a) describe the state before the transition (the `old-world`, thus pre-condition), (b) formalize the transition (the relation between `old-bal` and `new-bal`; (i)-(ii) in Figure 6.1) and (c) imply the consequences for the `new-world` ((iii) in Figure 6.1). These formulas verify that minting and depositing n coins into some address result in an increase of the sum by n, that is $\texttt{new-sum} = \texttt{old-sum} + n$, as expressed in the functional correctness formula (6.1) of Figure 6.1.

6.5.1 SL Encoding using Implicit Balances and Sums

The first encoding we present is a set of first-order formulas with equality over sorts {`Coin`, `Address`}. No additional theories are considered. The `Coin` sort represents money, where one coin is one unit of money. The `Address` sort represents the account addresses as before. As a consequence, balance functions and sum constants only exist implicitly in this encoding. As such, the property $\texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a)$ *cannot be directly expressed in this encoding.* Instead, we formalize this property by using so-called *smart invariant* relations between two predicates `has-coin` and `active` over coins $c \in \texttt{Coin}$ and $a \in \texttt{Address}$, as follows.

Definition 6.3 (Smart Invariants)**.** *Let* $\texttt{has-coin} \subseteq \texttt{Address} \times \texttt{Coin}$ *and consider* $\texttt{active} \subseteq \texttt{Coin}$*. A* smart invariant *of the pair* (`has-coin`, `active`) *is the conjunction of the following three formulas*

1. *Only active coins c can be owned by an address a:*

$$\forall c : \texttt{Coin}.\ \exists a : \texttt{Address}.\ \texttt{has-coin}(a, c) \rightarrow \texttt{active}(c)\ . \tag{I1}$$

2. *Every active coin c belongs to some address a:*

$$\forall c : \texttt{Coin}.\ \texttt{active}(c) \rightarrow \exists a : \texttt{Address}.\ \texttt{has-coin}(a, c)\ . \tag{I2}$$

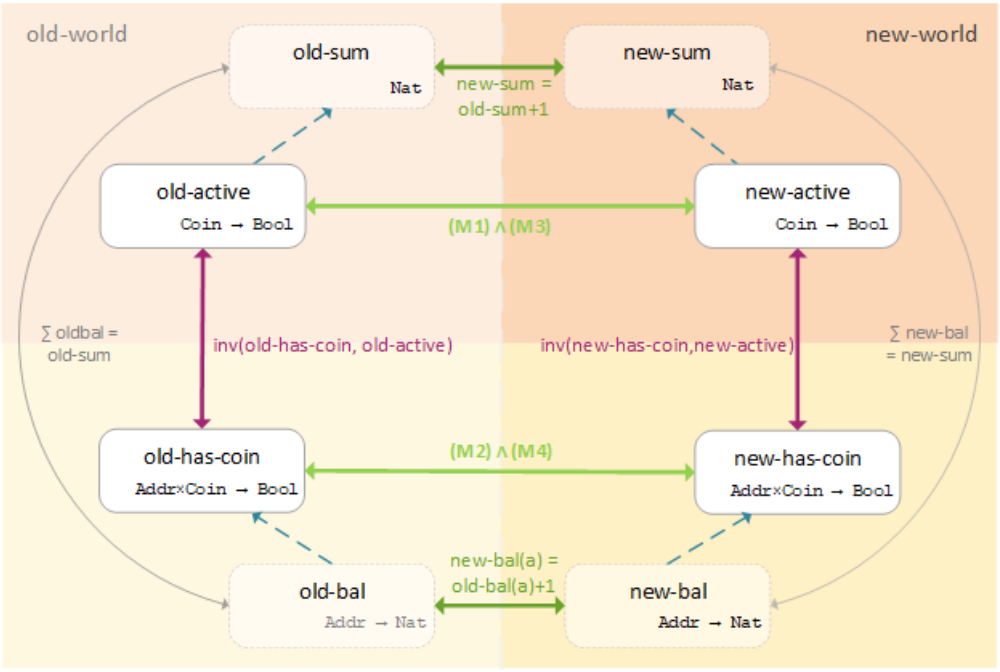


Figure 6.2: Implicit SL Encoding of $\texttt{mint}_1$, where `Addr` is Short for `Address`.

3. Every coin c belongs to at most one address a:

$$\forall c : \texttt{Coin}. \forall a, a' : \texttt{Address}. \quad (\texttt{has-coin}(a, c) \land \texttt{has-coin}(a', c) \to a \approx a') \,. \tag{I3}$$

We write $\texttt{inv}(\texttt{has-coin}, \texttt{active})$ *to denote the smart invariant (I1)* $\land$ *(I2)* $\land$ *(I3) of* $(\texttt{has-coin}, \texttt{active})$*.*

Intuitively, our *smart invariants* ensure that a coin c is *active* iff it is *owned* by precisely one address a. Our smart invariants imply the soundness of our implicit SL encoding, as follows.

Theorem 6.6 (Soundness of SL Encoding)**.** *Given that* $\texttt{sum} = |\texttt{active}|$ *and for every* $a \in \texttt{Address}$ *it holds* $\texttt{bal}(a) = |\{c \in \texttt{Coin} \mid (a, c) \in \texttt{has-coin}\}|$*, then the following implication is true:* $\texttt{inv}(\texttt{has-coin}, \texttt{active}) \implies \texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a)$*.*

Proof. The result follows from Theorem 6.9 by stating the properties of function f (Definition 6.8). □

We say that a *smart transition preserves smart invariants*, when

$$\texttt{inv}(\texttt{old-has-coin}, \texttt{old-active}) \iff \texttt{inv}(\texttt{new-has-coin}, \texttt{new-active}),$$

where old-has-coin, old-active and new-has-coin, new-active respectively denote the functions has-coin, active in the states before and after the smart transition. Based on the soundness of our implicit SL encoding, we formalize smart transitions preserving smart invariants as first-order formulas. We only discuss smart transitions implementing minting n coins here, but other transitions, such as transferring coins, can be handled in a similar manner. We first focus on minting a single coin, as follows.

Definition 6.4 (Transition $\texttt{mint}_1(a, c)$)**.** *Let there be* $c \in \texttt{Coin}, a \in \texttt{Address}$*. The transition* $\texttt{mint}_1(a, c)$ *activates coin* c *and deposits it into address* a*.*

1. *The coin* c *was inactive before and is active now:*

$$\neg\texttt{old-active}(c) \wedge \texttt{new-active}(c)\ . \tag{M1}$$

2. *The address* a *owns the new coin* c*:*

$$\texttt{new-has-coin}(a, c) \wedge \forall a' : \texttt{Address}.\ \neg\texttt{old-has-coin}(a', c)\ . \tag{M2}$$

3. *Everything else stays the same:*

$$\forall c' : \texttt{Coin}.\ c' \not\approx c \rightarrow \big(\texttt{new-active}(c') \leftrightarrow \texttt{old-active}(c')\big)\ , \tag{M3}$$

$$\forall c' : \texttt{Coin}.\ \forall a' : \texttt{Address}.\ (c' \not\approx c \vee a' \not\approx a) \rightarrow \big(\texttt{new-has-coin}(a', c') \leftrightarrow \texttt{old-has-coin}(a', c')\big)\ . \tag{M4}$$

The transition $\texttt{mint}_1(a, c)$ *is defined as (M1)* $\wedge$ *(M2)* $\wedge$ *(M3)* $\wedge$ *(M4).*

By minting one coin, the balance of precisely one address, that is of the receiver's address, increases by one, whereas all other balances remain unchanged. Thus, the expected impact on the sum of account balances is also increased by one, as illustrated in Figure 6.2. The following theorem proves that the definition of $\texttt{mint}_1$ is *sound*. That is, $\texttt{mint}_1$ affects the implicit balances and sums as expected and hence $\texttt{mint}_1$ preserves smart invariants.

Theorem 6.7 (Soundness of $\texttt{mint}_1(a, c)$)**.** *Let* $c \in \texttt{Coin}$*,* $a \in \texttt{Address}$ *such that* $\texttt{mint}_1(a, c)$*. Consider the balance functions* $\texttt{old-bal}, \texttt{new-bal} : \texttt{Address} \rightarrow \mathbb{N}$*, the non-negative integer constants* old-sum*,* new-sum*, the unary predicates* old-active*,* new-active $\subseteq$ Coin *and the binary predicates* old-has-coin*,* new-has-coin $\subseteq$ Address $\times$ Coin *such that*

$$|\texttt{old-active}| = \texttt{old-sum}\,,\ |\texttt{new-active}| = \texttt{new-sum},$$

and for every address a'*, we have*

$$\texttt{old-bal}(a') = \big|\{c' \in \texttt{Coin} \mid (a', c') \in \texttt{old-has-coin}\}\big|\ ,$$
$$\texttt{new-bal}(a') = \big|\{c' \in \texttt{Coin} \mid (a', c') \in \texttt{new-has-coin}\}\big|\ .$$

Then, $\texttt{new-sum} = \texttt{old-sum} + 1$, $\texttt{new-bal}(a) = \texttt{old-bal}(a) + 1$. *Moreover, for all other addresses* $a' \neq a$, *it holds* $\texttt{new-bal}(a') = \texttt{old-bal}(a')$.

Proof. To show that $\texttt{new-sum} = \texttt{old-sum} + 1$, we consider the subformula (M3) of $\texttt{mint}_1(a, c)$. It follows that $\texttt{old-active}\backslash\{c\} = \texttt{new-active}\backslash\{c\}$. Now using (M1), we get $\texttt{old-active} = \texttt{old-active}\backslash\{c\}$ and $(\texttt{new-active}\backslash\{c\}) \cup \{c\} = \texttt{new-active}$. Thus, $\texttt{old-active} \dot\cup \{c\} = \texttt{new-active}$ and hence $|\texttt{old-active}| + 1 = |\texttt{old-active} \dot\cup \{c\}| = |\texttt{new-active}|$ which implies $\texttt{new-sum} = \texttt{old-sum} + 1$.

Similar reasoning works to show $\texttt{new-bal}(a) = \texttt{old-bal}(a) + 1$. From (M4) it follows

$$\{d \mid (a, d) \in \texttt{old-has-coin}\}\backslash\{c\} = \{d \mid (a, d) \in \texttt{new-has-coin}\}\backslash\{c\} \,.$$

Now using (M2) we get

$$\{d \mid (a, d) \in \texttt{old-has-coin}\} = \{d \mid (a, d) \in \texttt{old-has-coin}\}\backslash\{c\} \quad \text{and}$$
$$\{d \mid (a, d) \in \texttt{new-has-coin}\}\backslash\{c\} \cup \{c\} = \{d \mid (a, d) \in \texttt{new-has-coin}\} \,.$$

As before, we have

$$\{d \mid (a, d) \in \texttt{old-has-coin}\} \dot\cup \{c\} = \{d \mid (a, d) \in \texttt{new-has-coin}\} \,,$$

which implies $\texttt{new-bal}(a) = \texttt{old-bal}(a) + 1$.

Finally, $\texttt{new-bal}(b) = \texttt{old-bal}(b)$ for $b \neq a$ follows from (M4), since it implies $\{d \mid (b, d) \in \texttt{old-has-coin}\} = \{d \mid (b, d) \in \texttt{new-has-coin}\}$. □

Smart transitions minting an arbitrary number of n coins, as in our Figure 6.1, is then realized by repeating the $\texttt{mint}_1$ transition n times. Based on the soundness of $\texttt{mint}_1$, ensuring that $\texttt{mint}_1$ preserves smart invariants, we conclude by induction that n repetitions of $\texttt{mint}_1$, that is *minting n coins, also preserves smart invariants*, as detailed below.

To define the smart transition $\texttt{mint}_n$ we need one pair of predicates for every time step. Thus we have an additional "parameter" i, the i-th time step, in `active` and `has-coin` instead of using the prefixes `old-`- and `new-`-. Other than that, the definition and the soundness result are analogous to the setting of $\texttt{mint}_1$.

Definition 6.5 (Transition $\texttt{mint}_n(a)$)**.** *Let* $a \in \texttt{Address}$. *Then, the transition* $\texttt{mint}_n(a)$ *activates n coins and deposits them into address a, one coin c in each time step.*

1. *The coin c was inactive before and is active now:*

$$\neg\texttt{active}(c, i) \wedge \texttt{active}(c, i + 1) \,. \tag{N1}$$

2. *The address a owns the new coin c:*

$$\texttt{has-coin}(a, c, i + 1) \wedge \forall a' : \texttt{Address}.\, \neg\texttt{has-coin}(a', c, i) \,. \tag{N2}$$

3. Everything else stays the same:

$$\forall c' : \texttt{Coin}.\, \forall a' : \texttt{Address}.\ (c' \not\approx c \vee a' \not\approx a) \rightarrow \tag{N3}$$
$$\big(\ (\texttt{active}(c', i+1) \leftrightarrow \texttt{active}(c', i)) \wedge$$
$$(\texttt{has-coin}(a', c', i+1) \leftrightarrow \texttt{has-coin}(a', c', i+1))\ \big)\ .$$

The transition $\texttt{mint}_n(a)$ *is defined as* $\forall i : \texttt{Nat}.\, \exists c : \texttt{Coin}.$ *(N1)* $\wedge$ *(N2)* $\wedge$ *(N3).*

The soundness result we get is similar to Theorem 6.7 but extended by the new parameter.

Theorem 6.8 (Soundness of $\texttt{mint}_n(a)$)**.** *Let* $a \in \texttt{Address}$ *such that* $\texttt{mint}_n(a)$*. Consider a balance function* $\texttt{bal} : \texttt{Address} \times \mathbb{N} \to \mathbb{N}$*, a summation function* $\texttt{sum} : \mathbb{N} \to \mathbb{N}^+$*, a binary predicate* $\texttt{active} \subseteq \texttt{Coin} \times \mathbb{N}$ *and a ternary predicate* $\texttt{has-coin} \subseteq \texttt{Address} \times \texttt{Coin} \times \mathbb{N}$ *such that for every* $i \in \mathbb{N}$

$$|\texttt{active}(., i)| = \texttt{sum}(i)$$

and for every address a' *and* $i \in \mathbb{N}$*, we have*

$$\texttt{bal}(a', i) = \big|\{c' \in \texttt{Coin} \mid (a', c', i) \in \texttt{has-coin}\}\big|\ .$$

Then for an arbitrary $n \in \texttt{Nat}$*,* $\texttt{sum}(n) = \texttt{sum}(0) + n$*,* $\texttt{bal}(a, n) = \texttt{bal}(a, 0) + n$*. Moreover, for all other addresses* $a' \neq a$*, it holds* $\texttt{bal}(a', n) = \texttt{bal}(a', 0)$*.*

Proof. We prove Theorem 6.8 by induction over $n \in \mathbb{N}$. The base case $n = 0$ is trivially satisfied. For the induction step, we get the induction hypothesis $\texttt{sum}(n) = \texttt{sum}(0) + n$, $\texttt{bal}(a, n) = \texttt{bal}(a, 0) + n$, $\forall a' \neq a.\ \texttt{bal}(a', n) = \texttt{bal}(a', 0)$. By defining $\texttt{old-sum} \triangleq \texttt{sum}(n)$, $\texttt{new-sum} \triangleq \texttt{sum}(n+1)$ and analogously for $\texttt{active}$, $\texttt{bal}$ and $\texttt{has-coin}$, all the preconditions of Theorem 6.7 hold. Therefore, we get $\texttt{sum}(n+1) = \texttt{sum}(n) + 1$, $\texttt{bal}(a, n+1) = \texttt{bal}(a, n) + 1$, $\forall a' \neq a.\ \texttt{bal}(a', n+1) = \texttt{bal}(a, n)$, by applying Theorem 6.7. Together with the induction hypothesis this yields $\texttt{sum}(n+1) = \texttt{sum}(0) + n + 1$, $\texttt{bal}(a, n+1) = \texttt{bal}(a, 0) + n + 1$, $\forall a' \neq a.\ \texttt{bal}(a', n+1) = \texttt{bal}(a', 0)$ and thus concludes the induction proof. □

6.5.2 Completeness Relative to a Translation Function

Smart invariants provide sufficient conditions for ensuring soundness of our SL encodings (Theorem 6.6). We next show that, under additional constraints, smart invariants are also necessary conditions, establishing thus *(relative) completeness of our encodings.*

A straightforward extension of Theorem 6.6 however does not hold. Namely, only under the assumptions of Theorem 6.6, the following formula is not valid:

$$\texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a) \quad \Longleftrightarrow \quad \texttt{inv}(\texttt{has-coin}, \texttt{active})\,.$$

As a counterexample, assume (i) $\texttt{sum} = |\texttt{active}|$, (ii) for every $a \in \texttt{Address}$ it holds that $\texttt{bal}(a) = |\{c \in \texttt{Coin} \mid (a, c) \in \texttt{has-coin}\}|$, that is the assumptions of Theorem 6.6. Further, let (iii) the smart invariants $\texttt{inv}(\texttt{has-coin}, \texttt{active})$ hold for all but the coins $c_1, c_2 \in \texttt{Coin}$ and all but the addresses $a_1, a_2 \in \texttt{Address}$. We also assume that (iv) c_1 is active but not owned by any address and (v) c_2 is active and owned by the two distinct addresses a_1, a_2. We thus have $\texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a)$, yet $\texttt{inv}(\texttt{has-coin}, \texttt{active})$ does not hold.

To ensure completeness of our encodings, we therefore introduce a translation function f that restricts the set $\mathcal{F} \triangleq 2^{\texttt{Address} \times \texttt{Coin}} \times 2^{\texttt{Coin}}$ of $(\texttt{has-coin}, \texttt{active})$ pairs, as follows. We exclude from $\mathcal{F}$ those pairs $(\texttt{has-coin}, \texttt{active})$ that violate smart invariants by both (i) not satisfying (I2), as (I2) ensures that there are not too many active coins, and by (ii) not satisfying at least one of (I1) and (I3), as (I1) and (I3) ensure that there are not too few active coins. The required translation function f assigns every pair $(\texttt{bal}, \texttt{sum})$ the set of all $(\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$ that satisfy $\texttt{sum} = |\texttt{active}|$, $\texttt{bal}(a) = |\{c \in \texttt{Coin} \mid \texttt{has-coin}(a, c)\}|$ for every address a and have not been excluded.

In order to establish a formal definition of f, some concepts have to be stated first. The mentioned exclusion of certain elements of $\mathcal{F}$ is based on an equivalence relation $\sim$.

Definition 6.6 (Relation $\sim$)**.** *Let the pairs* $p_1 = (\texttt{has-coin}_1, \texttt{active}_1) \in \mathcal{F}$ *and* $p_2 = (\texttt{has-coin}_2, \texttt{active}_2) \in \mathcal{F}$*. Then* $p_1 \sim p_2$ *iff*

1. $|\texttt{active}_1| = |\texttt{active}_2|$,
2. $|\{c \in \texttt{Coin} \mid \texttt{has-coin}_1(a, c)\}| = |\{c \in \texttt{Coin} \mid \texttt{has-coin}_2(a, c)\}|$*, for all* $a \in \texttt{Address}$*,*
3. p_1 *violates (I2) in* $V_{\leq}$ *cases and* p_2 *violates (I2) also* $V_{\leq}$ *times,*
4. p_1 *does not satisfy (I1) and (I3) in all together* $V_{\geq}$ *cases, which is also the number of times* p_2 *violates (I1) and (I3),*

where $V_{\leq}$ *and* $V_{\geq}$ *are as defined in Definition 6.7.*

The violation numbers $V_{\leq}$ and $V_{\geq}$ are introduced next.

Definition 6.7. *Given a pair* $(\texttt{active}, \texttt{has-coin}) \in \mathcal{F}$*. For an address* a*, we define* $C_a \triangleq \{c \in \texttt{Coin} \mid \texttt{has-coin}(a, c)\}$*. Further, we define three types of error coins:*

1. $M_{Inact} \triangleq \{c \in \texttt{Coin} \mid \neg\texttt{active}(c) \wedge \exists a.\ c \in C_a\}$,
2. $M_{Least} \triangleq \{c \in \texttt{Coin} \mid \texttt{active}(c) \wedge \forall a.\ c \notin C_a\}$ *and*
3. $M_{Most} \triangleq \{c \in \texttt{Coin} \mid \exists a, b.\ a \not\approx b \wedge c \in C_a \wedge c \in C_b\}$

and one type of error pairs $M_{Pairs} \triangleq \{(a, c) \mid c \in C_a \wedge \exists b.\ a \not\approx b \wedge c \in C_b\}$ *to refine the number of mistakes caused by the violation of (I3).*
The number of violations of (I2) is now $V_{\leq} \triangleq |M_{Least}|$*. and the number of violations of (I1) and (I3) is defined as* $V_{\geq} \triangleq |M_{Inact}| + |M_{Pairs}| - |M_{Most}|$*.*

Lemma 6.1. *The relation* $\sim$ *is an equivalence relation on* $\mathcal{F}$*.*

Proof. We have to show the reflexivity, symmetry, and transitivity of $\sim$.

- Reflexivity of $\sim$.
  Let $(\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$, then clearly $|\texttt{active}| = |\texttt{active}|$, for all a we have $|C_a| = |C_a|$ and also $V_{\leq} = V_{\leq}$, $V_{\geq} = V_{\geq}$. Hence $(\texttt{has-coin}, \texttt{active}) \sim (\texttt{has-coin}, \texttt{active})$.
- Symmetry of $\sim$.
  Let $p_1, p_2 \in \mathcal{F}$ such that $p_1 \sim p_2$, then due to symmetry of $=$ also $p_2 \sim p_1$ holds.
- Transitivity of $\sim$.
  Let $p_1, p_2, p_3 \in \mathcal{F}$, such that $p_1 \sim p_2$ and $p_2 \sim p_3$ then due to the transitivity of $=$ also $p_1 \sim p_3$ holds.

□

The translation function f can now be defined as a function that assigns every pair $(\texttt{bal}, \texttt{sum})$ a class from $\mathcal{F}/_{\sim}$.

Definition 6.8 (Translation Function f)**.** *The function* $f : \mathbb{N}^{\texttt{Address}} \times \mathbb{N} \to \mathcal{F}/_{\sim}$, $(\texttt{bal}, \texttt{sum}) \mapsto [(\texttt{has-coin}, \texttt{active})]_{\sim}$*, is defined to satisfy the following conditions for an arbitrary* $(\texttt{has-coin}, \texttt{active}) \in [(\texttt{has-coin}, \texttt{active})]_{\sim}$*.*

1. $\texttt{sum} = |\texttt{active}|$.
2. *For every* $a \in \texttt{Address}$ *it holds* $\texttt{bal}(a) = |\{c \in \texttt{Coin} \mid \texttt{has-coin}(a, c)\}|$.
3. *At least one of* $V_{\leq} = 0$ *and* $V_{\geq} = 0$ *holds.*

The function f is well-defined and injective, ensuring soundness and completeness of our SL encodings relative to f.

Theorem 6.9 (Relative Completeness of SL Encoding)**.** *Let* $(\texttt{bal}, \texttt{sum}) \in \mathbb{N}^{\texttt{Address}} \times \mathbb{N}$ *and let* $(\texttt{has-coin}, \texttt{active}) \in f(\texttt{bal}, \texttt{sum})$ *be arbitrary. Then,*

$$\texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a) \quad \Longleftrightarrow \quad \texttt{inv}(\texttt{has-coin}, \texttt{active})\,.$$

Proof. The proof is organized in 4 steps. The first step provides a technicality that is needed for steps 2 and 3, and finally, in the last step, the claim is proven.

1. Consider any pair $(\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$ with $V_{\leq} = V_{\geq} = 0$. Then, since there are no coins nor addresses violating the invariants here, we thus have $\bigcup_{a \in \texttt{Address}} C_a = \texttt{active}$ and all the C_a are disjoint. Thus, $\sum_{a \in \texttt{Address}} |C_a| = |\bigcup_{a \in \texttt{Address}} C_a| = |\texttt{active}|$.

2. Now we only assume $V_{\geq} = 0$. Consider

$$M_{\text{Least}} = \{c \in \texttt{Coin} \mid \texttt{active}(c) \wedge \forall a.\, c \notin C_a\} \subseteq \texttt{active} \,.$$

   Then the pair $p' = (\texttt{has-coin}, \texttt{active} \backslash M_{\text{Least}})$ satisfies $V'_{\leq} = V'_{\geq} = 0$, because all the coins were not active originally are active now and we did not change the any of the other mistake sets. From the first step we now get

$$\sum_{a \in \texttt{Address}} |C'_a| = |\texttt{active} \backslash M_{\text{Least}}|$$

   and therefore

$$\sum_{a \in \texttt{Address}} |C_a| = |\texttt{active}| + V_{\geq} - V_{\leq} \,.$$

3. Similarly to the second step we now only assume $V_{\leq} = 0$. By definition it holds that $M_{\text{Inact}} \cap \texttt{active} = \emptyset$, $M_{\text{Pairs}} \subseteq \texttt{has-coin}$ and $M_{\text{Most}} \subseteq \texttt{active} \cup M_{\text{Inact}}$. We now consider the pair

$$p'' = (\texttt{has-coin} \backslash M_{\text{Pairs}}, (\texttt{active} \cup M_{\text{Inact}}) \backslash M_{\text{Most}}) \,.$$

   Clearly, there is no coin assigned to two different addresses in p''. However, all the coins that were in two different addresses before are now not assigned to any address; this is why these coins have to be removed from $\texttt{active} \cup M_{\text{Inact}}$. Also, there are no coins that are active without belonging to any address. Further, all active coins are still assigned to an address as the problematic ones have been removed. Hence, $V''_{\leq} = V''_{\geq} = 0$. Now, we can again apply the result of the first step to get

$$\sum_{a \in \texttt{Address}} |C_a| - |M_{\text{Pairs}}| = |\texttt{active}| + |M_{\text{Inact}}| - |M_{\text{Most}}|$$

   and thus

$$\sum_{a \in \texttt{Address}} |C_a| = |\texttt{active}| + V_{\geq} - V_{\leq} \,.$$

4. Using the results of the previous two steps, we can now prove the theorem. Let $(\texttt{bal}, \texttt{sum}) \in \mathbb{N}^{\texttt{Address}} \times \mathbb{N}$ and $(\texttt{has-coin}, \texttt{active}) \in f(\texttt{bal}, \texttt{sum})$, then $V_{\leq} = 0$ or $V_{\geq} = 0$. In both cases it follows $\sum_{a \in \texttt{Address}} |C_a| = |\texttt{active}| + V_{\geq} - V_{\leq}$ and therefore by definition of f it holds $\sum_{a \in \texttt{Address}} \texttt{bal}(a) = \texttt{sum} + V_{\geq} - V_{\leq}$. Assume now $\sum_{a \in \texttt{Address}} \texttt{bal}(a) = \texttt{sum}$. It follows $V_{\geq} - V_{\leq} = 0$ and since we know that

one of these values has to be zero by definition of f it holds $V_{\leq} = V_{\geq} = 0$. But this statement is equivalent to $\texttt{inv}(\texttt{has-coin}, \texttt{active})$. For the other direction assume $\texttt{inv}(\texttt{has-coin}, \texttt{active})$, this implies $V_{\leq} = V_{\geq} = 0$ and hence $\sum_{a \in \texttt{Address}} \texttt{bal}(A) = \texttt{sum}$. This concludes the proof.

□

6.5.3 SL Encodings using Explicit Balances and Sums

We now restrict our SL encoding from Section 6.5.1 to explicitly reason with balance functions during smart transitions. We do so by expressing our translation function f from Section 6.5.2 in first-order logic. We now use the summation constant $\texttt{sum} \in \mathbb{N}$ and the balance function $\texttt{bal} : \texttt{Address} \to \mathbb{N}$ in our SL encoding. In particular, we use our smart invariants $\texttt{inv}(\texttt{has-coin}, \texttt{active})$ in this explicit SL encoding together with two additional axioms (Ax1, Ax2), ensuring that $\texttt{sum} = |\texttt{active}|$ and $\texttt{bal}(a) = |\{c \in \texttt{Coin} \mid \texttt{has-coin}(a, c)\}|$ for all $a \in \texttt{Address}$.

To formalize the additional properties, we introduce two counting mechanisms in our SL encoding. The first one is a bijective function $\texttt{count} : \texttt{Coin} \to \mathbb{N}^+$ and the second one is a function $\texttt{idx} : \texttt{Address} \times \texttt{Coin} \to \mathbb{N}^+$, where $\texttt{idx}(a, .) : \texttt{Coin} \to \mathbb{N}^+$ is bijective for every $a \in \texttt{Address}$. To ensure that $\texttt{count}$ and $\texttt{idx}(a, .)$ count coins, we impose the following two properties:

$$\forall c : \texttt{Coin}.\ \texttt{active}(c) \iff \texttt{count}(c) \leq \texttt{sum}\,, \tag{Ax1}$$

$$\forall c : \texttt{Coin}.\ \forall a : \texttt{Address}.\ \texttt{has-coin}(a, c) \iff \texttt{idx}(a, c) \leq \texttt{bal}(a)\,. \tag{Ax2}$$

Figure 6.3 illustrates our revised SL encoding for our smart transition $\texttt{mint}_1$. We next ensure soundness of our resulting explicit encoding for summation, as follows.

Theorem 6.10 (Soundness of Explicit SL Encodings)**.** *Let there be a pair* $(\texttt{bal}, \texttt{sum}) \in \mathbb{N}^{\texttt{Address}} \times \mathbb{N}$*, a pair* $(\texttt{has-coin}, \texttt{active}) \in \mathcal{F}$*, and functions* $\texttt{count} : \texttt{Coin} \to \mathbb{N}^+$ *and* $\texttt{idx} : \texttt{Address} \times \texttt{Coin} \to \mathbb{N}^+$*.*

Given that $\texttt{count}$ *is bijective,* $\texttt{idx}(a, .) : \texttt{Coin} \to \mathbb{N}^+$ *is bijective for every* $a \in \texttt{Address}$*, and that (Ax1), (Ax2) and* $\texttt{inv}(\texttt{has-coin}, \texttt{active})$ *hold, then,* $\texttt{sum} = |\texttt{active}|$ *and* $\texttt{bal}(a) = |\{c \in \texttt{Coin} : \texttt{has-coin}(a, c)\}|$*, for every* $a \in \texttt{Address}$*.*

In particular, we have $\texttt{sum} = \sum_{a \in \texttt{Address}} \texttt{bal}(a)$*.*

Proof. Consider $(\texttt{bal}, \texttt{sum})$, $(\texttt{has-coin}, \texttt{active})$ as in the theorem. Then by property (Ax1) and the codomain of $\texttt{count}$ we have $\texttt{active} = \{c \in \texttt{Coin} \mid \texttt{count}(c) \in [1, \texttt{sum}]\}$. Since $\texttt{count}$ is bijective, it holds

$$|\texttt{active}| = |\{c \in \texttt{Coin} \mid \texttt{count}(c) \in [1, \texttt{sum}]\}| = \texttt{sum}\,.$$

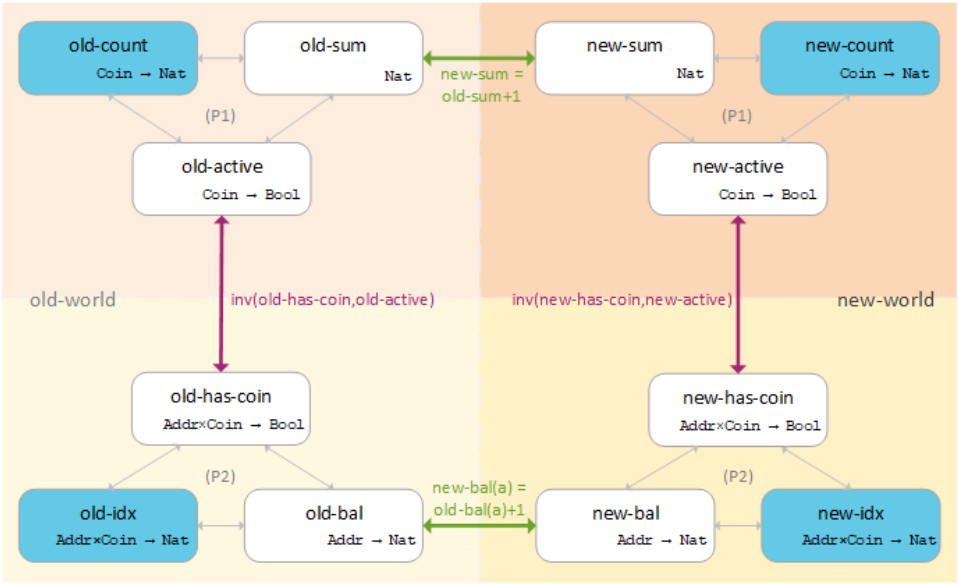


Figure 6.3: Explicit SL Encoding of $\texttt{mint}_1$, where `Addr` is Short for `Address`.

Similarly, by (Ax2) and the codomain of $\texttt{idx}(a,.)$ we know $C_a = \{c \in \texttt{Coin} \mid \texttt{idx}(a,c) \in [1, \texttt{bal}(a)]\}$. As $\texttt{idx}(a,.)$ is bijective as well it follows

$$|C_a| = |\{c \in \texttt{Coin} \mid \texttt{idx}(a,c) \in [1, \texttt{bal}(a)]\}| = \texttt{bal}(a) \; .$$

Hence $(\texttt{has-coin}, \texttt{active}) \in f(\texttt{bal}, \texttt{sum})$ and by Theorem 6.9, we get

$$\sum_{a \in \texttt{Address}} \texttt{bal}(a) = \texttt{sum} \; ,$$

since $\texttt{inv}\,(\texttt{has-coin}, \texttt{active})$. □

When compared to Section 6.5.1, our explicit SL encoding introduced above uses our smart invariants as axioms of our encoding, together with (Ax1) and (Ax2). In our explicit SL encoding, the post-conditions asserting functional correctness of smart transitions express thus relations among `old-sum` to `new-sum`. For example, for $\texttt{mint}_n$ we are interested in ensuring

$$\texttt{mint}_n \Rightarrow \texttt{new-sum} = \texttt{old-sum} + n \; . \tag{6.2}$$

By using two new constants $\texttt{old-total}, \texttt{new-total} \in \mathbb{N}$, we can use $\texttt{sum} = \texttt{total}$ as smart invariant for $\texttt{mint}_n$. As a result, the property to be ensured is then

$$\begin{aligned} &(\texttt{old-sum} = \texttt{old-total} \wedge \texttt{new-total} = \texttt{old-total} + n \wedge \texttt{mint}_n) \\ &\Rightarrow (\texttt{new-sum} = \texttt{new-total}) \; . \end{aligned} \tag{6.3}$$

It is easy to see that the negations of (6.2) and (6.3) are equisatisfiable. We note however that the additional constants `old-total`, `new-total` used in (6.3) lead to unstable results within automated reasoners, as discussed in Section 6.6.

6.6 Experimental Evaluation

6.6.1 From Theory to Practice

To make our explicit SL encodings handier for automated reasoners, we improved the setting illustrated in Figure 6.3 by applying the following restrictions without losing any generality.

(i) The predicates `has-coin` and `active` were removed from the explicit SL encodings, by replacing them with their equivalent expressions (Ax1)-(Ax2).

(ii) The surjectivity assertions of `count` and `idx` were restricted to the relevant intervals $[1, \texttt{sum}]$, $[1, \texttt{bal}(a)]$ respectively.

(iii) Compared to Figure 6.3, only one mutual `count` and one mutual `idx` functions were used. We, however, conclude that we do not lose expressivity of our resulting SL encoding, as shown in Theorem 6.11.

(iv) When our SL encoding contains expressions such as $\forall c : \texttt{Coin}.\ \texttt{idx}(a_0, c) \in [l_0, u_0] \iff \texttt{idx}(a_1, c) \in [l_1, u_1]$, with a_0, a_1 being distinct addresses such that either $u_i \leq \texttt{bal}(a_i)$ or $l_i > \texttt{bal}(a_i)$, $i \in \{0, 1\}$, then it can be assumed that the coins in those intervals are in the same order for both functions (Theorem 6.12).

No Loss of Generality: Restricting `idx` and `count`. To prove we do not lose any generality when considering mutual `count` and `idx` functions for the old- and the new-world, we need the following two preliminary lemmas.

Lemma 6.2. *Given two pairs* $h_x \triangleq (\texttt{x-bal}, \texttt{x-sum})$, $h_y \triangleq (\texttt{y-bal}, \texttt{y-sum})$ *with* $\sum_{a \in \texttt{Address}} \texttt{z-bal}(A) = \texttt{z-sum}$, *for* $\texttt{z} \in \{\texttt{old}, \texttt{new}\}$ *and* $\texttt{x-sum} \leq \texttt{y-sum}$. *Further, let* $p_x = (\texttt{x-has-coin}, \texttt{x-active}) \in f(h_x)$.

Then there exists $p_y = (\texttt{y-has-coin}, \texttt{y-active}) \in f(h_y)$ *satisfying the following properties:*

1. $\texttt{x-active} \subseteq \texttt{y-active}$.
2. $\texttt{x-bal}(a) \leq \texttt{y-bal}(a) \Rightarrow C_{x,a} \subseteq C_{y,a}$.
3. $\texttt{y-bal}(a) \leq \texttt{x-bal}(a) \Rightarrow C_{y,a} \subseteq C_{x,a}$.

Proof. We proceed by constructing $p_y = (\texttt{y-has-coin}, \texttt{y-active}) \in f(h_y)$ such that it satisfies properties (1)-(3). To fulfill property (1), let $\texttt{y-active} \triangleq \texttt{x-active} \cup S$, where $S \in \texttt{Coin} \backslash \texttt{x-active}$ and $|S| = \texttt{y-sum} - \texttt{x-sum}$. Then also $|\texttt{y-active}| = \texttt{y-sum}$ holds. To construct the $C_{y,a}$ properly, the set `y-active` has to be partitioned, since $p_y \in f(h_y)$ and thus $\texttt{inv}(\texttt{y-has-coin}, \texttt{y-active})$. For every a with $\texttt{x-bal}(a) \leq \texttt{y-bal}(a)$ we require $C_{x,a} \subseteq C_{y,a}$. Therefore there are

$$\sum_{a:\, \texttt{x-bal}(a) > \texttt{y-bal}(a)} \texttt{x-bal}(a) - \texttt{y-bal}(a)$$

additional spare coins. For a with $\texttt{x-bal}(a) \geq \texttt{y-bal}(a)$ we want $C_{y,a} \subseteq C_{x,a}$, which leaves us with

$$\sum_{a:\, \texttt{x-bal}(a)<\texttt{y-bal}(a)} \texttt{y-bal}(a) - \texttt{x-bal}(a)$$

missing coins. Hence, the difference is

$$\begin{aligned} \texttt{y-sum} - \texttt{x-sum} \quad &+ \sum_{\substack{a\in\texttt{Address},\\ \texttt{x-bal}(a)>\texttt{y-bal}(a)}} \texttt{x-bal}(a) - \texttt{y-bal}(a) \\ &- \sum_{\substack{a\in\texttt{Address},\\ \texttt{y-bal}(a)>\texttt{x-bal}(a)}} \texttt{y-bal}(a) - \texttt{x-bal}(a) \quad . \end{aligned}$$

By replacing $\texttt{z-sum}$ by $\sum_{a\in\texttt{Address}} \texttt{z-bal}(a)$ all the summands with either $\texttt{y-bal}(a) > \texttt{x-bal}(a)$ or $\texttt{x-bal}(a) > \texttt{y-bal}(a)$ disappear and the remaining value is 0. Therefore, such a partition of $\texttt{y-active}$ exists and thus, there exists

$$p_y = (\texttt{y-active}, \texttt{y-has-coin}) \in f(h_y)$$

satisfying (1), (2) and (3). □

Lemma 6.3. *Given two pairs h_x, h_y with $\sum_{a\in\texttt{Address}} \texttt{z-bal}(a) = \texttt{z-sum}$, $p_z \in f(h_z)$, for $\texttt{z} \in \{\texttt{x}, \texttt{y}\}$ and $\texttt{x-sum} \leq \texttt{y-sum}$ as in Lemma 6.2. Then, there exist a bijective function $\texttt{count} : \texttt{Coin} \to \mathbb{N}^+$ with $\texttt{count}(\texttt{z-active}) = [1, \texttt{z-sum}]$ and bijective functions $\texttt{idx}(a,.) : \texttt{Coin} \to \mathbb{N}^+$, with $\texttt{idx}(C_{z,a}) = [1, \texttt{z-bal}(a)]$, for $\texttt{z} \in \{\texttt{x}, \texttt{y}\}$, $a \in \texttt{Address}$.*

Proof. At first, we construct $\texttt{count}$. We know $\texttt{y-active} = \texttt{x-active} \dot\cup S$, where $|\texttt{y-active}| = \texttt{y-sum}$, $|\texttt{x-active}| = \texttt{x-sum}$ and $|S| = \texttt{y-sum} - \texttt{x-sum}$. Thus, we can easily find an injective function with $\texttt{count}(\texttt{x-active}) = [1, \texttt{x-sum}]$ and $\texttt{count}(S) = [\texttt{x-sum}+1, \texttt{y-sum}]$. Further, this function can be bijectively extended onto $\mathbb{N}^+$. Similarly, for the addresses a, we construct $\texttt{idx}(a,.)$ in the following way. Since we know $|C_{z,a}| = \texttt{z-bal}(a)$, we can find an injective function with $\texttt{idx}(a, C_{x,a}) = [1, \texttt{x-bal}(a)]$. For all a, where $\texttt{y-bal}(a) \leq \texttt{x-bal}(a)$, we can assume that $\texttt{idx}(a, C_{y,a}) = [1, \texttt{y-bal}(a)]$, as $C_{y,a} \subseteq C_{x,a}$. For these addresses a, $\texttt{idx}(a,.)$ can now also be extended bijectively onto $\mathbb{N}^+$. Finally, for a with $\texttt{x-bal}(a) \leq \texttt{y-bal}(a)$ we know $C_{x,a} \subseteq C_{y,a}$ and can thus assume $\texttt{idx}(a, C_{y,a} \backslash C_{x,a}) = [\texttt{x-bal}(a) + 1, \texttt{y-bal}(a)]$. Now also these $\texttt{idx}(a,.)$ can be extended bijectively onto $\mathbb{N}^+$. □

Having these two lemmas at hand, we can now state and prove the following result.

Theorem 6.11. *Given pairs $h_o \triangleq (\texttt{old-bal}, \texttt{old-sum})$, $h_n \triangleq (\texttt{new-bal}, \texttt{new-sum})$ with $\sum_{a\in\texttt{Address}} \texttt{z-bal}(a) = \texttt{z-sum}$, for $\texttt{z} \in \{\texttt{old}, \texttt{new}\}$. There exist bijective functions $\texttt{count} : \texttt{Coin} \to \mathbb{N}^+$ and $\texttt{idx}(a,.) : \texttt{Coin} \to \mathbb{N}^+$, for every $a \in \texttt{Address}$, such*

that there exist a $p_o = (\texttt{old-active}, \texttt{old-has-coin}) \in f(h_o)$*, as well as a* $p_n = (\texttt{new-active}, \texttt{new-has-coin}) \in f(h_n)$ *with*

$$\forall c.\ (\texttt{z-active}(c) \leftrightarrow \texttt{count}(c) \leq \texttt{z-sum}) \quad \textit{and} \tag{6.4}$$

$$\forall a, c.\ (\texttt{z-has-coin}(a, c) \leftrightarrow \texttt{idx}(a, c) \leq \texttt{z-bal}(a))\ . \tag{6.5}$$

Proof. Let $h_x \in \{h_o, h_n\}$ such that $\texttt{x-sum} = \min\{\texttt{old-sum}, \texttt{new-sum}\}$. The other pair gets the prefix 'y-' from now on. Also, elements in $f(h_x)$ and $f(h_y)$ will be named accordingly.

Let $(\texttt{x-active}, \texttt{x-has-coin}) \in f(h_x)$ arbitrary, $(\texttt{y-has-coin}, \texttt{y-active}) \in f(h_y)$ as in Lemma 6.2 and count, idx as in Lemma 6.3.

Then it holds $\texttt{count}(\texttt{z-active}) = [1, \texttt{z-sum}]$. Thus, $\forall c.\ c \in \texttt{z-active} \rightarrow \texttt{count}(c) \in [1, \texttt{z-sum}]$. As count is bijective and therefore injective, it follows $\forall c.\ c \notin \texttt{z-active} \rightarrow \texttt{count}(c) \notin [1, \texttt{z-sum}]$. Together with the fact that the codomain of count is $\mathbb{N}^+$ we get Formula 6.4. The analog argumentation works for Formula 6.5. We know $\texttt{idx}(a, C_{z,a}) = [1, \texttt{z-bal}(a)]$. Thus $\forall c.\ c \in C_{z,a} \rightarrow \texttt{idx}(a, c) \in [1, \texttt{z-bal}(a)]$. Also $\texttt{idx}(a, .)$ is bijective and therefore injective which implies $\forall c.\ c \notin C_{z,a} \rightarrow \texttt{idx}(a, c) \notin [1, \texttt{z-bal}(a)]$. By definition of $C_{z,a}$ and the codomain of $\texttt{idx}(a, .)$ Formula 6.5 holds. This concludes the proof. □

No Loss of Generality: Ordering of Coins. The property to prove is that whenever a block of coins has the same order in two of our counting functions and they are not crossing their crucial value $(\texttt{sum}, \texttt{bal}(a_i))$, then we can assume that they are ordered in the same way. In order to do so, we have to formalize the notion of the former invariants $\texttt{inv'}(\texttt{idx}, \texttt{count})$. They are the formulas one gets by replacing has-coin and active by count and idx according to (Ax1) and (Ax2) in the invariants (I1)-(I3).

Definition 6.9. *Let* $\texttt{count} : \texttt{Coin} \rightarrow \mathbb{N}^+$ *and* $\texttt{idx} : \texttt{Address} \times \texttt{Coin} \rightarrow \mathbb{N}^+$*, then with the formulas*

$$\forall c.\ (\exists a.\ \texttt{idx}(a, c) \leq \texttt{bal}(a)) \leftrightarrow \texttt{count}(c) \leq \texttt{sum}\ , \tag{I1'}$$

$$\forall a, b, c.\ (\texttt{idx}(a, c) \leq \texttt{bal}(a) \wedge \texttt{idx}(b, c) \leq \texttt{bal}(b)) \rightarrow a \approx b \tag{I2'}$$

we define $\texttt{inv'}(\texttt{idx}, \texttt{count}) \triangleq$ *I1' ∧ I2'.*

Theorem 6.12. *We assume the following*

(i) $(\texttt{old-bal}, \texttt{old-sum}), (\texttt{new-bal}, \texttt{new-sum}) \in \mathbb{N}^{\texttt{Address}} \times \mathbb{N}$*,*

(ii) $\texttt{count} : \texttt{Coins} \rightarrow \mathbb{N}^+$ *bijective,*

(iii) $\texttt{idx} : \texttt{Address} \times \texttt{Coin} \rightarrow \mathbb{N}^+$*, such that* $\texttt{idx}(A, .)$ *bijective for every a and*

(iv) $\texttt{inv'}(\texttt{idx}, \texttt{count})$*.*

If now

(v) $\forall c : \texttt{Coin}.\ f_0(c) \in [l_0, u_0] \leftrightarrow f_1(c) \in [l_1, u_1]$, *where* $f_0, f_1 \in \{\texttt{count}\} \cup \{\texttt{idx}(a, .) : a \in \texttt{Address}\}$ *and*

(vi) *either* $u_i \leq x_i$ *or* $x_i < l_i$ *for* $i \in \{0, 1\}$, *where*

$x_i \triangleq \texttt{bal}(a_i)$, *if* $f_i = \texttt{idx}(a_i, .)$, *or*

$x_i \triangleq \texttt{sum}$, *if* $f_i = \texttt{count}$,

then there exist $\texttt{idx}'$, $\texttt{count}'$ *with the properties* (i)-(vi) *and* $\forall c : \texttt{Coin}.\ f'_0(c) \in [l_0, u_0] \rightarrow f'_0(c) + l_1 \approx f'_1(c) + l_0$.

Proof. For simplicity assume $f_0 = \texttt{idx}(a_0, .)$ and $f_1 = \texttt{idx}(a_1, .)$. However, the proof works in an analogous way for `count`. We proceed by constructing `idx`' and `count`' and then showing properties (i)-(vi) and $\forall c : \texttt{Coin}.\ f'_0(c) \in [l_0, u_0] \rightarrow f'_0(c) + l_1 = f'_1(c) + l_0$ hold.
The function $\texttt{count}' \triangleq \texttt{count}$. We construct `idx`' the following way.

$$\forall a, c.\ (a \not\approx a_1 \vee \texttt{idx}(a_1, c) \notin [l_1, u_1]) \text{ let } \texttt{idx}'(a, c) \triangleq \texttt{idx}(a, c) \quad \text{and}$$
$$\forall c.\ \texttt{idx}(a_1, c) \in [l_1, u_1] \text{ we define } \texttt{idx}'(a_1, c) \triangleq \texttt{idx}(a_0, c) - l_0 + l_1\ .$$

With these definitions the properties (i), (ii), (vi) and $\forall c : \texttt{Coin}.\ f'_0(c) \in [l_0, u_0] \rightarrow f'_0(c) + l_1 = f'_1(c) + l_0$ obviously hold.

To show property (v) we first fix a coin c and assume $\texttt{idx}'(a_0, c) \in [l_0, u_0]$. By definition of `idx`' we know that also $\texttt{idx}(a_0, c) \in [l_0, u_0]$ and then using (v) we get $\texttt{idx}(a_1, c) \in [l_1, u_1]$. Again by definition of `idx`', we have $\texttt{idx}'(a_1, c) = \texttt{idx}(a_0, c) - l_0 + l_1$. Since $\texttt{idx}(a_0, c) \in [l_0, u_0]$, it follows $\texttt{idx}'(a_1, c) = \texttt{idx}(a_0, c) - l_0 + l_1 \in [l_1, u_0 - l_0 + l_1]$. From property (v) and the bijectivity of the $\texttt{idx}(a, .)$ functions, it follows that the intervals have the same size and thus $u_0 - l_0 = u_1 - l_1$. Therefore we end up having $\texttt{idx}'(a_1, c) \in [l_1, u_1]$. For the other direction of the equivalence, we fix c and assume $\texttt{idx}'(a_0, c) \notin [l_0, u_0]$. Then $\texttt{idx}(a_0, c) \notin [l_0, u_0]$ and using (v) we get $\texttt{idx}(a_1, c) \notin [l_1, u_1]$ and thus $\texttt{idx}'(a_1, c) = \texttt{idx}(a_1, c) \notin [l_1, u_1]$. This concludes the proof of (v). Note that (v) implies $\texttt{idx}'(a_1, c) \in [l_1, u_1]$ if and only if $\texttt{idx}(a_1, c) \in [l_1, u_1]$.

The proof of property (iii) only requires showing $\texttt{idx}'(a_1, .)$ is bijective. To show injectivity, assume $\texttt{idx}'(a_1, c) = \texttt{idx}'(a_1, d)$. where $c \neq d$. Then clearly $\texttt{idx}(a_1, c) \neq \texttt{idx}(a_1, d)$. Thus, one of $\texttt{idx}(a_1, c), \texttt{idx}(a_1, d) \in [l_1, u_1]$, since otherwise $\texttt{idx}'(a_1, c) = \texttt{idx}(a_1, c)$ and $\texttt{idx}'(a_1, d) = \texttt{idx}(a_1, d)$. From the implication of (v), we know that $\texttt{idx}'(a_1, c) = \texttt{idx}'(a_1, d) \in [l_1, u_1]$ and thus $\texttt{idx}(a_1, c), \texttt{idx}(a_1, d) \in [l_1, u_1]$. Therefore $\texttt{idx}(a_0, c) - l_0 + l_1 = \texttt{idx}'(a_1, c) = \texttt{idx}'(a_1, d) = \texttt{idx}(a_0, d) - l_0 + l_1$ and thus $\texttt{idx}(a_0, c) = \texttt{idx}(a_0, d)$ which is a contradiction to the bijectivity of `idx`.

For surjectivity of $\texttt{idx}'(a_1, .)$ let $n \in \mathbb{N}^+$ be arbitrary. Assume $n \notin [l_1, u_1]$ first. Then by surjectivity of $\texttt{idx}(a_1, .)$ it follows that there is a c such that $\texttt{idx}(a_1, c) = n$ and as

$n \notin [l_1, u_1]$, we have $\texttt{idx}'(a_1, c) = n$. Assume now $n \in [l_1, u_1]$, then by surjectivity of $\texttt{idx}(a_0, .)$, there exists c such that $\texttt{idx}(a_0, c) = n - l_1 + l_0$. With the same reasoning as above it follows $n - l_1 + l_0 \in [l_0, u_0]$ and therefore we conclude $\texttt{idx}'(a_1, c) = \texttt{idx}(a_0, c) - l_0 + l_1 = n$. This completes the surjectivity proof.

Finally, we have to prove (iv). Once we have shown $\texttt{idx}(a, c) \leq \texttt{bal}(a)$ iff $\texttt{idx}'(a, c) \leq \texttt{bal}(a)$ for all a and for all c the property follows immediately, since $\texttt{count}' = \texttt{count}$. For all a, c with one of $a \neq a_1$ or $\texttt{idx}(a_1, c) \notin [l_1, u_1]$ the equivalence follows from the definition of $\texttt{idx}'$. Consider now a_1 with a c such that $\texttt{idx}(a_1, c) \in [l_1, u_1]$. Using the implication of (v), we get $\texttt{idx}'(a_1, c) \in [l_1, u_1]$. Now using property (vi), we know that either $u_1 \leq \texttt{bal}(a_1)$, in which case $\texttt{idx}(a_1, c)$, $\texttt{idx}'(a_1, c) \leq \texttt{bal}(a_1)$, or $\texttt{bal}(a_1) < l_1$ which implies $\texttt{idx}(a_1, c)$, $\texttt{idx}'(a_1, c) > \texttt{bal}(a_1)$. This concludes the proof of property (iv) and thus of the theorem. □

Based on the above, we derive three different explicit SL encodings to be used in automated reasoning about smart transitions. We respectively denote these explicit SL encodings by `int`, `nat` and `id`, and describe them next.

6.6.2 Experiments

Benchmarks. In our experiments, we consider four smart transitions $\texttt{mint}_1$, $\texttt{mint}_n$, $\texttt{transferFrom}_1$ and $\texttt{transferFrom}_n$, respectively denoting minting and transferring one and n coins. These transitions capture the main operations of linear integer arithmetic. In particular, $\texttt{mint}_n$ implements the smart transition of our running example from Figure 6.1.

For each of the four smart transitions, we implement four SL encodings: the implicit SL encoding `uf` from Section 6.5.1 using only uninterpreted functions and three explicit encodings `int`, `nat` and `id` as variants of Section 6.5.3. We also consider three additional arithmetic benchmarks using `int`, which are not directly motivated by smart contracts. Together with variants of `int` and `nat` presented in the sequel, our benchmark set contains 31 examples altogether, with each example being formalized in the SMT-LIB input syntax [BFT16]. In addition to our encodings, we also proved consistency of the axioms used in our encodings.

SL Encodings and Relaxations. Our explicit SL encoding `int` uses linear integer arithmetic, whereas `nat` and `id` are based on natural numbers. As naturals are not a built-in theory in SMT-LIB, we assert the axioms of Presburger arithmetic directly in the encodings of `nat` and `id`.

In our `id` encodings, inductive datatypes are additionally used to order coins. There exists one linked list of all coins for `count` and one for each $\texttt{idx}(a, .)$, $a \in \texttt{Address}$. Additionally, there exists a "null" coin, which is the first element of every list and is not owned by any address. As shown in Figure 6.4, the numbering of each coin is defined by its position in the respective list. This way surjectivity for `count` and

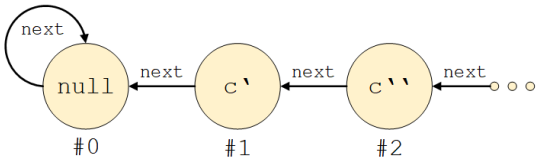


Figure 6.4: Linked Lists in `id`.

`idx` can respectively be asserted by the formulas $\exists c : \texttt{Coin}.\ \texttt{count}(c) \approx \texttt{sum}$ and $\forall a : \texttt{Address}.\ \exists c : \texttt{Coin}.\ \texttt{idx}(a, c) \approx \texttt{bal}(a)$. However, asserting surjectivity for `int` and `nat` cannot be achieved without quantifying over $\mathbb{N}^+$. Such quantification would drastically affect the performance of automated reasoners in (fragments of) first-order logics. As a remedy, within the default encodings of `int` and `nat`, we only consider relevant instances of surjectivity.

Further, we consider variations of `int` and `nat` by asserting proper surjectivity to the relevant intervals of `idx` and `count` (denoted as *surj*) and/or adding the `total` constants mentioned in Section 6.5.3 (denoted as *with* `total`*, no* `total`) . These variations of `int` and `nat` are implemented for $\texttt{mint}_1$ and $\texttt{transferFrom}_1$.

Experimental Setting. We evaluated our benchmark set of 31 examples using SMT solvers Z3 [DMB08] and CVC4 [BCD+11], as well as the first-order theorem prover Vampire [KV13]. Our experiments were run on a standard machine with an Intel Core i5-6200U CPU (2.30GHz, 2.40GHz) and 8 GB RAM. The time is given in seconds and we ran all experiments with a time limit of 300s. Time out is indicated by the symbol $\times$. The default parameters were used for each solver, unless stated otherwise in the corresponding tables, indicated by the superscripts $*$, †, and ‡. The precise calls of the solvers are listed in Table 6.3. Examples of the encodings can be found in Figures 6.5 to 6.7[4].

Experimental Analysis. We first report on our experiments using different variations of `int` and `nat`. Table 6.4 shows that asserting complete surjectivity for `int` and `nat` is computationally hard and indeed significantly affects the performance of automated reasoners. Thus, for the following experiments only relevant instances of surjectivity, such as $\exists c : \texttt{Coin}.\ \texttt{count}(c) = \texttt{sum}$ were asserted in `int` and `nat`. Table 6.4 also illustrates the instability of using the `total` constant. Some tasks seem to be easier, even though their reasoning difficulty increased strictly by adding additional constants.

Our most important experimental findings are shown in Table 6.5, demonstrating that *our* SL *encodings are suitable for automated reasoners. Thanks to our explicit* SL *encodings, each solver can certify every smart transition in at least one encoding.* Our explicit SL encodings are more relevant than the implicit encoding `uf` as we can express and compare any two non-negative integer sums, whereas for `uf` handling arbitrary values n can only

[4]All encodings are available at `https://github.com/SoRaTu/SmartSums`.

Z3	`z3 -smt2 <file-name>`
CVC4	`cvc4 -lang=smtlib2.6 -full-saturate-quant <file-name>`
Vampire	`vampire -input_syntax smtlib2 <file-name>` except for the cases with superscripts $*$, † or ‡: $*$ `vampire -input_syntax smtlib2 <file-name> -forced_options "aac=none:add=large:afp=40000:afq=1.2:amm=off:anc=none:bd=off:fsr=off:gsp=input_only:inw=on:irw=on:lma=on:nm=64:nwc=1:sos=on:sp=occurrence:tha=off:updr=off:awr=5:s=1011:sa=discount:ind=math"` † `vampire -input_syntax smtlib2 -thsq on -thsqd 6 -thsqc 6 -thsqr 10,1 <file-name>` ‡ options from $*$ to prove the property in `ind_property_id.smt`, options from † to prove the actual property and its lemmas, if any

Table 6.3: Precise Calls of the Solvers in Tables 6.4 to 6.6.

$\texttt{mint}_1$				$\texttt{transferFrom}_1$			
no `total`	**Z3**	**CVC4**	**Vampire**	no `total`	**Z3**	**CVC4**	**Vampire**
`nat`	0.02	×	0.92	`nat`	×	×	15.35
`nat` surj.	×	×	×	`nat` surj.	100.03	×	×
`int`	0.02	0.03	×	`int`	0.02	0.07	×
`int` surj.	×	5.96	×	`int` surj.	1.02	×	×
with `total`	**Z3**	**CVC4**	**Vampire**	with `total`	**Z3**	**CVC4**	**Vampire**
`nat`	0.03	×	2.92	`nat`	0.28	×	22.54
`nat` surj.	0.11	×	×	`nat` surj.	38.24	×	×
`int`	0.02	0.03	×	`int`	0.02	0.10	×
`int` surj.	3.81	5.95	×	`int` surj.	×	6.56	×

Table 6.4: Results of $\texttt{mint}_1$ and $\texttt{transferFrom}_1$ using `nat` and `int`, with/without the `total` Constants and with/without Surjectivity.

be done by iterating over the $\texttt{mint}_1$ (or $\texttt{transferFrom}_1$) transition. This iteration requires inductive reasoning, which currently only Vampire could do [HHK+20], as indicated by the superscript $*$. Nevertheless, the transactions $\texttt{mint}_1$, $\texttt{transferFrom}_1$, which involve only one coin in `uf`, require no inductive reasoning as the actual sum is not considered; each of our solvers can certify these examples.

We note that the tasks $\texttt{mint}_n$ and $\texttt{transferFrom}_n$ from Table 6.5 yield a huge search space when using their explicit SL encodings within automated reasoners. We split these tasks into proving intermediate lemmas and proved each of these lemmas independently, by the respective solver. In particular, we used one lemma for $\texttt{mint}_n$ and four lemmas for $\texttt{transferFrom}_n$. In our experiments, we only used the recent theory reasoning framework of Vampire with split queues [GS20] and indicate our results by superscript †.

We further remark that our explicit SL encoding `id` using inductive datatypes also

Encoding	Task							
	$\texttt{mint}_1$		$\texttt{transferFrom}_1$		$\texttt{mint}_n$		$\texttt{transferFrom}_n$	
uf	Z3:	0.01	Z3:	0.02	Z3:	×	Z3:	×
	CVC4:	0.02	CVC4:	0.03	CVC4:	×	CVC4:	×
	Vampire:	0.18	Vampire:	0.19	Vampire:	0.35*	Vampire:	0.44*
nat	Z3:	0.02	Z3:	×	Z3:	×	Z3:	×
	CVC4:	×	CVC4:	×	CVC4:	×	CVC4:	×
	Vampire:	0.92	Vampire:	15.35	Vampire:	$23.23^{\dagger}$	Vampire:	$228.22^{\dagger}$
int	Z3:	0.02	Z3:	0.02	Z3:	0.03	Z3:	0.11
	CVC4:	0.03	CVC4:	0.07	CVC4:	0.05	CVC4:	0.35
	Vampire:	×	Vampire:	×	Vampire:	×	Vampire:	×
id	Z3:	×	Z3:	×	Z3:	×	Z3:	×
	CVC4:	×	CVC4:	×	CVC4:	×	CVC4:	×
	Vampire:	$7.36^{\ddagger}$	Vampire:	$17.16^{\ddagger}$	Vampire:	$23.52^{\ddagger}$	Vampire:	×

Table 6.5: Smart Transitions using Implicit/Explicit SL Encodings.

Task		Time	
Transition	Impact		
$\texttt{new-bal}(a_0) = \texttt{old-bal}(a_0) + 3$ $\texttt{new-bal}(a_1) = \texttt{old-bal}(a_1) - 3$	$\texttt{new-sum} = \texttt{old-sum}$	Z3: CVC4: Vampire:	0.20 1.28 ×
$\texttt{new-bal}(a_0) = \texttt{old-bal}(a_0) + 4$ $\texttt{new-bal}(a_1) = \texttt{old-bal}(a_1) - 2$	$\texttt{new-sum} = \texttt{old-sum} + 2$	Z3: CVC4: Vampire:	0.58 7.14 ×
$\texttt{new-bal}(a_0) = \texttt{old-bal}(a_0) + 5$ $\texttt{new-bal}(a_1) = \texttt{old-bal}(a_1) - 3$ $\texttt{new-bal}(a_2) = \texttt{old-bal}(a_2) - 1$	$\texttt{new-sum} = \texttt{old-sum} + 1$	Z3: CVC4: Vampire:	1.52 155.20 ×

Table 6.6: Arithmetic Reasoning in the Explicit SL Encoding `int`.

requires inductive reasoning about smart transitions and beyond. The need for induction explains why SMT solvers failed to prove our `id` benchmarks, as shown in Table 6.5. We note that Vampire found a proof using built-in induction [HHK+20] and theory-specific reasoning [GS20], as indicated by superscript ‡.

We conclude by showing the generality of our approach beyond smart transitions. It in fact enables fully automated reasoning about any two summations $\sum_{i \in I} g(i)$, $\sum_{i \in I} h(i)$ of non-negative integer values $g(i)$, $h(i)$ $(i \in I)$ over a mutual finite set I. The examples of Table 6.6 affirm this claim.

6.7 Related work

Smart Contract Safety. Formal verification of smart contracts is an emerging hot topic because of the value of the assets stored in smart contracts, e.g. the DeFi software [Com20]. Due to the nature of the blockchain, bugs in smart contracts are irreversible and thus the demand for provably bug-free smart contracts is high.

The K interactive framework has been used to verify safety of a smart contract, e.g.

```
(set-logic UFLIA)
(declare-sort Coin 0 )
(declare-sort Address 0)
(declare-fun old-sum () Int)
(declare-fun new-sum () Int)
(declare-fun old-total () Int)
(declare-fun new-total () Int)
(declare-fun a0 () Address)
(declare-fun old-bal (Address) Int)
(declare-fun new-bal (Address) Int)
(declare-fun count (Coin) Int)
(declare-fun ind (Coin Address) Int)
; axioms on sum and count
(assert (<= 0 old-sum))                                    ; sum non-negative
(assert (<= 0 new-sum))                                    ; sum non-negative
(assert (forall ((C Coin)) (< 0 (count C)) ) )               ; count positive
(assert (forall ((C Coin) (D Coin))
          (=> (= (count C) (count D)) (= C D) )))          ; count injective
(assert (forall ((N Int)) (=>
          (and (< 0 N) (or (<= N old-sum) (<= N new-sum)) )
          (exists ((C Coin)) (= (count C) N) ))))          ; count surjective
; axioms on bal and find
(assert (forall ((A Address)) (<= 0 (old-bal A)) ))      ; bal non-negative
(assert (forall ((A Address)) (<= 0 (new-bal A)) ))      ; bal non-negative
(assert (forall ((C Coin)(A Address)) (< 0 (ind C A)) ))    ; ind positive
(assert (forall ((C Coin) (D Coin) (A Address))
          (=> (= (ind C A) (ind D A)) (= C D) )))      ; ind(A,.) injective
(assert (forall ((N Int) (A Address)) (=>
          (and (< 0 N) (or (<= N (new-bal A)) (<= N (old-bal A))))
          (exists ((C Coin)) (= (ind C A) N) ))))     ; ind(A,.) subjective
; axioms between sum and bal
(assert (forall ((C Coin)) (=
          (exists ((A Address)) (<= (ind C A) (old-bal A)) )
          (<= (count C) old-sum) )))              ; find<=bal iff count<=sum
(assert (forall ((C Coin)) (=
          (exists ((A Address)) (<= (ind C A) (new-bal A)) )
          (<= (count C) new-sum) )))              ; find<=bal iff count<=sum
(assert (forall ((A Address)(B Address)(C Coin)) (=>
          (and (<= (ind C A) (old-bal A)) (<= (ind C B) (old-bal B)) )
          (= A B) )))                                    ; only once find<=bal
(assert (forall ((A Address)(B Address)(C Coin)) (=>
          (and (<= (ind C A) (new-bal A)) (<= (ind C B) (new-bal B)) )
          (= A B) )))                                    ; only once find<=bal
; transition and expected impact
(assert (and (= (new-bal a0) (+ (old-bal a0) 1)) (forall ((A Address))
          (=> (distinct A a0) (= (old-bal A) (new-bal A)) ))))     ; mint1
(assert (= (+ old-sum 1) new-sum) )                       ; expected impact
; invariants
(assert (= old-sum old-total) )                             ; pre-invariant
(assert (distinct new-sum new-total) )            ; negated post-invariant
(check-sat)
```

Figure 6.5: Encoding of `mint1`, Full Surjectivity, with `total`, `int` Version.

```
(set-logic UFLIA)
(declare-sort Coin 0 )
(declare-sort Address 0)
(declare-fun act (Coin Int) Bool )
(declare-fun hc (Address Coin Int) Bool)
(declare-fun induct (Int) Bool)
(declare-const a1 Address)
(declare-const a2 Address)
(declare-const n Int)
; inductive predicate definition
(assert (forall ((I Int))  (= (induct I)
         (and (forall ((C Coin))
                      (= (exists ((A Address)) (hc A C I)) (act C I)) ))
              (forall ((A Address) (B Address) (C Coin))
                      (=> (and (hc A C I) (hc B C I)) (= A B)) )))))
; pre-invariants
(assert (forall ((C Coin)) (= (exists ((A Address)) (hc A C 0))
         (act C 0) )))                  ; inactive coins and at least one
(assert (forall ((A Address)(B Address)(C Coin))
         (=> (and (hc A C 0) (hc B C 0)) (= A B) )))       ; at most one
; transition
(assert (forall ((I Int)) (=> (<= 0 I)
         (and (forall ((D Coin)) (= (act D I) (act D (+ I 1)) ))
              (exists ((C Coin)) (and
               (hc a1 C I) (not (hc a2 C I))
               (not (hc a1 C (+ I 1))) (hc a2 C (+ I 1))
               (forall ((D Coin) (A Address))
                (=> (or (distinct C D)
                        (and (distinct A a1) (distinct A a2)) )
                    (= (hc A D (+ I 1)) (hc A D I)) ))))))))
; negated post-invariant
(assert (and (<= 0 n) (not (induct n)) ))
(check-sat)
```

Figure 6.6: Encoding of `transferFromN`, `uf` Version.

```
(set-logic UFLIA)
(declare-sort Coin 0 )
(declare-sort Address 0)
(declare-fun old-sum () Int)
(declare-fun new-sum () Int)
(declare-fun a0 () Address)
(declare-fun a1 () Address)
(declare-fun a2 () Address)
(declare-fun old-bal (Address) Int)
(declare-fun new-bal (Address) Int)
(declare-fun count (Coin) Int)
(declare-fun ind (Coin Address) Int)
```

```
13 ; axioms on sum and count
14 (assert (<= 0 old-sum))                                  ; sum non-negative
15 (assert (<= 0 new-sum))                                  ; sum non-negative
16 (assert (forall ((C Coin)) (< 0 (count C)) ) )            ; count positive
17 (assert (forall ((C Coin) (D Coin))
18          (=> (= (count C) (count D)) (= C D) )))       ; count injective
19 (assert (exists ((C Coin)) (= (count C) old-sum) ))
20 (assert (exists ((C Coin)) (= (count C) (+ old-sum 1)) ))
21 (assert (exists ((C Coin)) (= (count C) new-sum) ))
22                                        ; count instances of surjectivity
23 ;  axioms on bal and find
24 (assert (forall ((A Address)) (<= 0 (old-bal A)) ))  ; bal non-negative
25 (assert (forall ((A Address)) (<= 0 (new-bal A)) ))  ; bal non-negative
26 (assert (forall ((C Coin)(A Address)) (< 0 (ind C A)) )) ; ind positive
27 (assert (forall ((C Coin) (D Coin) (A Address))
28          (=> (= (ind C A) (ind D A)) (= C D) )))   ; ind(A,.) injective
29 (assert (exists ((C Coin)) (= (ind C a0) (old-bal a0)) ))
30 (assert (exists ((C Coin)) (= (ind C a0) (+(old-bal a0) 1))))
31 (assert (exists ((C Coin)) (= (ind C a0) (+(old-bal a0) 2))))
32 (assert (exists ((C Coin)) (= (ind C a0) (+(old-bal a0) 3))))
33 (assert (exists ((C Coin)) (= (ind C a0) (+(old-bal a0) 4))))
34 (assert (exists ((C Coin)) (= (ind C a0) (+(old-bal a0) 5))))
35 (assert (exists ((C Coin)) (= (ind C a1) (new-bal a1)) ))
36 (assert (exists ((C Coin)) (= (ind C a1) (+(new-bal a1) 1))))
37 (assert (exists ((C Coin)) (= (ind C a1) (+(new-bal a1) 2))))
38 (assert (exists ((C Coin)) (= (ind C a1) (+(new-bal a1) 3))))
39 (assert (exists ((C Coin)) (= (ind C a2) (new-bal a2)) ))
40 (assert (exists ((C Coin)) (= (ind C a2) (+(new-bal a2) 1))))
41                                     ; find(A,.) instances of surjectivity
42 ; axioms between sum and bal
43 (assert (forall ((C Coin)) (=
44         (exists ((A Address)) (<= (ind C A) (old-bal A)) )
45          (<= (count C) old-sum) )))          ; find<=bal iff count<=sum
46 (assert (forall ((C Coin)) (=
47          (exists ((A Address)) (<= (ind C A) (new-bal A)) )
48          (<= (count C) new-sum) )))          ; find<=bal iff count<=sum
49 (assert (forall ((A Address)(B Address)(C Coin)) (=>
50          (and (<= (ind C A) (old-bal A)) (<= (ind C B) (old-bal B)) )
51          (= A B) )))                                ; only once find<=bal
52 (assert (forall ((A Address)(B Address)(C Coin)) (=>
53          (and (<= (ind C A) (new-bal A)) (<= (ind C B) (new-bal B)) )
54          (= A B) )))                                ; only once find<=bal
55 ; transition and negated impact
56 (assert (and (forall ((A Address)) (=>
57                (and (distinct A a0) (distinct A a1) (distinct A a2))
58                (= (old-bal A) (new-bal A)) ))))
59          (= (new-bal a0) (+ (old-bal a0) 5))
60          (= (old-bal a1) (+ (new-bal a1) 3))
61          (= (old-bal a2) (+ (new-bal a2) 1))      ; plus5 minus3 minus1
62 (assert (distinct (+ old-sum 1) new-sum) )             ; negated impact
63 (check-sat)
```

Figure 6.7: Encoding of $\texttt{bal}(a_0) + 5$, $\texttt{bal}(a_1) - 3$, $\texttt{bal}(a_2) - 1$.

in [PZR20]. Isabelle [Nip12] was also shown to be useful in manual, interactive verification of smart contracts [Hir17]. We, however, focus on automated approaches.

There are also efforts to perform deductive verification of smart contracts both on the source level in languages such as Solidity [WLC+19, Alt19, HJ19] and Move [ZCQ+20], as well as on the Ethereum virtual machine (EVM) level [Cer, SGSM20]. This work improves the effectiveness of these approaches by developing techniques for automatically reasoning about unbounded sums. This way, we believe we support a more semantic-based verification of smart contracts.

Our approach differs from works using ghost variables [HJ19], since we do not manually update the "ghost state". Instead, the verifier needs only to reason about the local changes, and the aggregate state is maintained by the axioms. That means other approaches assume (a) the local changes and (b) the impact on ghost variables (sum), whereas we only assume (a) and automatically prove $a \Rightarrow b$. This way, we reduce the user-guidance in providing and proving (b).

Our work complements approaches that verify smart contracts as finite state machines [WLC+19] and methods, like ZEUS [KGDS18], using symbolic model checking and abstract interpretation to verify generic safety properties for smart contracts.

The work in [SFM+21] provides an extensive evaluation of ERC-20 and ERC-721 tokens. ERC-721 extends ERC-20 with ownership functions, one of which being "approve". It enables transactions on another party's behalf. This is independent of our ability to express sums in first-order logic, as the transaction's initiator is irrelevant to its effect.

Reasoning about Financial Applications. Recently, the Imandra prover introduced an automated reasoning framework for financial applications [PCI+20, Pas18, PI17]. Similarly to our approach, these works use SMT procedures to verify and/or generate counterexamples to safety properties of low- and high-level algorithms. In particular, results of [PCI+20, Pas18, PI17] include examples of verifying ranking orders in matching logics of exchanges, proving high-level properties such as transitivity and anti-symmetry of such orders. In contrast, we focus on verifying properties relating local changes in balances to changes of the global state (the sum). Moreover, our encodings enable automated reasoning both in SMT solving and first-order theorem proving.

Automated Aggregate Reasoning. The theory of first-order logic with aggregate operators has been thoroughly studied in [HLNW01, Lib99]. Though proven to be strictly more expressive than first-order logic, both in the case of general aggregates as well as simple counting logics, in this work we present a practical way to encode a weakened version of aggregates (specifically sums) in first-order logic. Our encoding (as in Section 6.5) works by expressing particular sums of interest, harnessing domain knowledge to avoid the need of general aggregate operators.

Previous works [KNR05, BREO+19] in the field of higher-order reasoning do not directly discuss aggregates. The work of [KNR05] extends Presburger arithmetic with Boolean

algebra for finite, unbounded sets of uninterpreted elements. This includes a way to express the set cardinalities and to compare them against integer variables, but does not support uninterpreted functions, such as the balance functions we use throughout our approach.

The SMT-based framework of [BREO+19] takes a different, white-box approach, modifying the inner workings of SMT solvers to support higher-order logic. We, on the other hand, treat theorem provers and SMT solvers as black-boxes, constructing first-order formulas that are tailored to their capabilities. This allows us to use any off-the-shelf SMT solver.

In [DDC10], an SMT module for the theory of FO(Agg) is presented, which can be used in all DPLL-based SAT, SMT and ASP solvers. However, FO(Agg) only provides a way to express functions that have sets or similar constructs as inputs, but not to verify their semantic behavior.

6.8 Conclusion

We present a methodology for reasoning about unbounded sums in the context of *smart transitions*, that is transitions that occur in smart contracts modeling transactions. Our sum logic SL and its usage of sum constants, instead of fully-fledged sum operators, turns out to be most appropriate for the setting of smart contracts. We show that SL has decidable fragments (Section 6.4.1), as well as undecidable ones (Section 6.4.2). Using two phases to first implicitly encode SL in first-order logic (Section 6.5.1), and then explicitly encode it (Section 6.5.3), allows us to use off-the-shelf automated reasoners in new ways, and automatically verify the semantic correctness of smart transitions.

Showing the (un)decidability of the SL fragment with two sets of uninterpreted functions and sums is an interesting step for further work, as this fragment supports encoding smart transition systems. Another interesting direction of future work is to apply our approach to different aggregates, such as minimum and maximum and to reason about under which conditions these values stay above/below certain thresholds. A slightly modified setting of our SL axioms can already handle min/max aggregates in a basic way, namely by using $\geq$ and $\leq$ instead of equality and dropping the injectivity/surjectivity (respectively) axioms of the counting mechanisms.

Summation upon multidimensional arrays in various ways is yet another direction of future research. Our approach supports the summation over all values in all dimensions by adding the required number of parameters to the predicate `idx` and by adapting the axioms accordingly.

CHAPTER 7

Summary and Outlook

In this thesis, we addressed two aspects of blockchain security. First and foremost, we developed a comprehensive framework for game-theoretic security analysis, combining extensive form games, symbolic payoffs, and automated reasoning techniques to rigorously model and evaluate a protocol's incentive structures. Additionally, this dissertation extends to implementation security, proposing methods for formalizing and verifying unbounded summations in smart contracts. Thereby enabling both theoretical and practical advancements in blockchain security.

Game-Theoretic Security. This thesis introduced a novel approach to modeling blockchain protocols for game-theoretic security analysis by leveraging extensive form games with symbolic payoffs. By defining game-theoretic security based on weak immunity, collusion resilience, and practicality, we provide a rigorous framework for assessing the incentive structures of blockchain protocols. The proof-of-concept application to Bitcoin's Lightning Network demonstrates the ability to model and analyze real-world components with mathematical precision. These models are the first to accurately capture relevant aspects, such as arbitrary deviations, and establish formal criteria for when these protocols are secure. Our approach sets the stage for further game-theoretic modeling of blockchain protocols, in particular through offering clear criteria for when incentive structures should be deemed secure.

Significant advancements in the automated analysis of game-theoretic security were made in this thesis. We developed a framework based on SMT solving, which enables the automatic analysis of game models with symbolic payoffs. This approach allows for the efficient and rigorous evaluation of complex game models through a sound and complete encoding of game-theoretic concepts into first-order logic. Additionally, the introduction of methods for automatically computing weakest preconditions, counterexamples, and witness strategies further enhances the utility of the framework, rendering it an effective approach for security analysis in blockchain protocols.

The novel techniques for SMT-based automated game-theoretic security analysis were implemented in the newly developed tool CheckMate. Our experiments, evaluated on multiple benchmarks, demonstrate the scalability and effectiveness of CheckMate, with results obtained in seconds for most models. They, however, also show their limits. Our tool was not able to analyze a model with over 500,000 nodes. To the best of our knowledge, our tool is the first to allow for automated game-theoretic security analysis and the first to support symbolic payoffs.

The method for compositional reasoning over the game-theoretic security properties that we established allowed us to push the limits of automated game-theoretic analysis beyond game models of 100,000,000 nodes. Thus, making it possible for complex real-world blockchain protocols to have verified incentive structures. To this end, we leveraged divide-and-conquer techniques, defined compositional variations of game-theoretic security properties, and proved their equivalence to the original definitions. This enables the independent analysis of subgames, significantly reducing computational and storage requirements. We further implemented these concepts in CheckMate2.0, the next generation of our framework. It demonstrates substantial runtime improvements and allows iterative and memory-efficient analysis of large-scale game models. This compositional approach not only overcomes practical barriers to analyzing massive game models but also paves the way for more modular game-theoretic security evaluations.

Implementation Security. Concerning reasoning about unbounded summations in smart contracts, we generalized Presburger arithmetic to express properties about summations and provided a formal framework for verifying the correctness of smart contract operations, such as maintaining the total supply of tokens during transactions. Our approach explores the decidability of this extended logic and offers sound encodings into first-order logic for practical verification. Our experimental evaluation on 31 new benchmarks demonstrated the encodings' applicability to real-world smart contract properties. This line of work aims to enhance the implementation security in blockchain systems by offering a framework for reasoning about unbounded summations in a formal manner.

Outlook. Using the results of this thesis as a basis, two major directions of future research arise. During the modeling of protocols, it became apparent that many decentralized finance applications rely on chance in addition to the users' choices. While we still believe that modeling the protocols as extensive form games was the right decision, we are convinced that an enhancement of EFGs to support chance would be invaluable. Such an extension is relevant to applications like decentralized gambling but also to blockchain protocols that depend on price fluctuations in various markets. We think the compositional approach to game-theoretic security implemented in CheckMate2.0 can be leveraged to accomplish said enhancement.

The other most evident avenue of further work is the automated modeling of a blockchain protocol as a game. The current overhead of modeling is the main bottleneck that keeps

game-theoretic security analysis from being widely used. The main challenge, in our opinion, is the lack of formal descriptions of blockchain protocols. Without a rigorous specification, automated modeling is severely hindered. Nonetheless, there is a class of protocols that does have precise documentation: In a smart contract, it is clearly stated which capabilities each user has, as code is law. We, therefore, believe automating the modeling of a smart contract as a game is a promising direction to improve the usability of game-theoretic security.

Further, it would be interesting to investigate the applicability of our results to other areas. Specifically, one could explore whether game-theoretic analyses are also meaningful and beneficial in other research fields. More generally, it would be worthwhile to study whether the verification of cryptographic security, implementation security, and game-theoretic security can all be symbiotically combined in a unified framework. Such an approach would enable users to formally model their protocol only once.

Overview of Generative AI Tools Used

I used ChatGPT-4 to rephrase some of my sentences more formally or clearer throughout Chapter 1 and Chapter 7. Additionally, I used it to provide synonyms.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AEE+21] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostakova, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. In *AsiaCrypt*, pages 635–664, 2021.

[AES25] Salam Al-E'mari and Yousef Sanjalawe. A Review of Reentrancy Attack in Ethereum Smart Contracts. In Giancarlo Fortino, Akshi Kumar, Abhishek Swaroop, and Pancham Shukla, editors, *Proceedings of Third International Conference on Computing and Communication Networks*, pages 53–70, Singapore, 2025. Springer Nature Singapore.

[AHWW20] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. Ride the Lightning: The Game Theory of Payment Channels. In *FC*, pages 264–283, 2020.

[AKWZ21] Zeta Avarikioti, Eleftherios Kokoris Kogias, Roger Wattenhofer, and Dionysis Zindros. Brick: Asynchronous Incentive-Compatible Payment Channels. In *FC*, pages 209–230, 2021.

[AL09] Gilad Asharov and Yehuda Lindell. Utility Dependence in Correct and Fair Rational Secret Sharing. Cryptology ePrint Archive, 2009.

[ALS+18] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. Towards Secure and Efficient Payment Channels. *arXiv preprint*, 2018.

[Alt19] Leonardo Alt. Solidity's SMTChecker can Automatically find Real Bugs, 2019.

[ALW20] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In *FC*, pages 346–366, 2020.

[AME+21] Lukas Aumayr, Matteo Maffei, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostakova, and Pedro Moreno-Sanchez. Bitcoin-Compatible Virtual Channels. In *SP*, pages 901–918, 2021.

[Ami] Nils Amiet. Polynonce: A Tale of a Novel ECDSA Attack and Bitcoin Tears. https://research.kudelskisecurity.com/2023/03/06/polynonce-a-tale-of-a-novel-ecdsa-attack-and-bitcoin-tears/.

[AMKM21] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Blitz: Secure Multi-Hop Payments Without Two-Phase Commits. In *Usenix Security*, pages 4043–4060, 2021.

[ASW19] Georgia Avarikioti, Rolf Scheuner, and Roger Wattenhofer. Payment Networks as Creation Games. In *CBT*, pages 195–210, 2019.

[Aut25] The Solidity Authors. Solidity Documentation. available online, 2025. https://docs.soliditylang.org/en/v0.8.24/.

[BCD+11] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, pages 171–177, 2011.

[BFT16] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). www.SMT-LIB.org, 2016.

[BGM+18] Christian Badertscher, Juan Garay, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. But why does it Work? A Rational Protocol Design Treatment of Bitcoin. In *Eurocrypt*, pages 34–65, 2018.

[BHZ23] Joe Bolt, Jules Hedges, and Philipp Zahn. Bayesian Open Games. *Compositionality*, 5:9, October 2023.

[BKK+23a] Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. In *Proceedings of 30th ACM SIGSAC Conference on Computer and Communications Security*, pages 1407–1421, New York, NY, USA, 2023.

[BKK+23b] Lea Salome Brugger, Laura Kovács, Anja Petković Komel, Sophie Rain, and Michael Rawson. CheckMate: Automated Game-Theoretic Security Reasoning. EasyChair Preprint no. 10853, 2023.

[BKK+25] Ivana Bocevska, Anja Petković Komel, Laura Kovács, Sophie Rain, and Michael Rawson. Divide and Conquer: a Compositional Approach to Game-Theoretic Security. EasyChair Preprint no. 15785, 2025.

[Bla14] Bruno Blanchet. Automatic Verification of Security Protocols in the Symbolic Model: The Verifier ProVerif. In *Foundations of Security Analysis and Design*, pages 54–87, Cham, 2014. Springer.

[BMR20] Prabal Banerjee, Subhra Mazumdar, and Sushmita Ruj. Griefing-Penalty: Countermeasure for Griefing Attack in Bitcoin-compatible PCNs. *CoRR*, abs/2005.09327, 2020.

[BN24] Nikolaj S. Bjørner and Lev Nachmanson. Arithmetic Solving in Z3. In *CAV*, pages 26–41, Cham, 2024. Springer.

[BREO+19] Haniel Barbosa, Andrew Reynolds, Daniel El Ouraoui, Cesare Tinelli, and Clark Barrett. Extending SMT Solvers to Higher-Order Logic. In *CADE*, pages 35–54, 2019.

[BT18] Clark Barrett and Cesare Tinelli. Satisfiability Modulo Theories. In *Handbook of Model Checking*, pages 305–343. Springer, UC Berkeley, 2018.

[But14] Vitalik Buterin. A Next-Generation Smart Contract and Decentralized Application Platform, 2014. https://ethereum.org/content/whitepaper/whitepaper-pdf/Ethereum_Whitepaper_-_Buterin_2014.pdf.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally Composable Security with Global Setup. In *TCC*, pages 61–85, 2007.

[Cer] Certora. The Certora Prover. https://docs.certora.com/en/latest/.

[CGP19] Krishnendu Chatterjee, Amir Goharshady, and Arash Pourdamghani. Probabilistic Smart Contracts: Secure Randomness on the Blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 403–412, Seoul, South Korea, 2019. IEEE.

[CGV18] Krishnendu Chatterjee, Amir Kafshdar Goharshady, and Yaron Velner. Quantitative Analysis of Smart Contracts. In *ESOP*, pages 739–767, 2018.

[CMVSM18] Anamika Chauhan, Om Malviya, Madhav Verma, and Tejinder Singh Mor. Blockchain and Scalability. In *QRS-C*, pages 122–128, 2018.

[Col75] George E Collins. Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decompostion. In *Automata Theory and Formal Languages*, pages 134–183, Berlin, Heidelberg, 1975. Springer.

[Com20] Concourse Open Community. Concourse Open Community: DeFi Pulse, 2020. https://defipulse.com/.

[CPR19] Xi Chen, Christos Papadimitriou, and Tim Roughgarden. An Axiomatic Approach to Block Rewards. In *AFT*, pages 124–131, 2019.

[DDC10] Marc Denecker and Broes De Cat. DPLL (Agg): An efficient SMT Module for Aggregates. In *Logic and Search*, 2010.

[DDM06a] Bruno Dutertre and Leonardo De Moura. A Fast Linear-Arithmetic Solver for DPLL (T). In *CAV*, pages 81–94, Berlin, Heidelberg, 2006. Springer.

[DDM06b] Bruno Dutertre and Leonardo De Moura. The Yices SMT Solver, 2006. Tool paper at http://yices.csl.sri.com/tool-paper.pdf.

[DEF+19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-Party Virtual State Channels. In *EuroCrypt*, pages 625–656, 2019.

[DEFM19] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual Payment Hubs over Cryptocurrencies. In *SP*, pages 106–123, 2019.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In *TACAS*, pages 337–340, Berlin, Heidelberg, 2008. Springer.

[DTH+17] Cuong T. Do, Nguyen H. Tran, Choongseon Hong, Charles A. Kamhoua, Kevin A. Kwiat, Erik Blasch, Shaolei Ren, Niki Pissinou, and Sundaraja Sitharama Iyengar. Game Theory for Cyber Security and Privacy. *ACM Comput. Surv.*, 50(2), 2017.

[DW15] Christian Decker and Roger Wattenhofer. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *SSS*, pages 3–18, 2015.

[Eme90] E Allen Emerson. Temporal and Modal Logic. In *Formal Models and Semantics*, pages 995–1072. Elsevier, 1990.

[ERE20] Oğuzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. How to Profit from Payments Channels. In *FC*, pages 284–303, 2020.

[ERI+21a] Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing up Smart Transitions. In *Proceedings of 33rd International Conference on Computer Aided Verification,*, pages 317–340, Cham, Switzerland, 2021.

[ERI+21b] Neta Elad, Sophie Rain, Neil Immerman, Laura Kovács, and Mooly Sagiv. Summing Up Smart Transitions. *arXiv preprint*, 2021.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not Enough: Bitcoin Mining is Vulnerable. In *FC*, pages 436–454, 2014.

[Ete97] Kousha Etessami. Counting Quantifiers, Successor Relations, and Logarithmic Space. In *JCSS*, pages 400–411, 1997.

[Eya15] Ittay Eyal. The Miner's Dilemma. In *IEEE S&P*, pages 89–103, 2015.

[GBC23] Amit Kumar Goel, Vibhor Singh Bisht, and Shreyashi Chaudhary. Multisignature Crypto Wallet Paper. In *2023 8th International Conference on Communication and Electronics Systems (ICCES)*, pages 476–479, 2023.

[GDM09] Yeting Ge and Leonardo De Moura. Complete Instantiation for Quantified Formulas in Satisfiability Modulo Theories. In *CAV*, pages 306–320, Berlin, Heidelberg, 2009. Springer.

[GHWZ18] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional Game Theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, page 472–481, New York, NY, USA, 2018. Association for Computing Machinery.

[GKLNF20] Neil Ghani, Clemens Kupke, Alasdair Lambert, and Fredrik Nordvall Forsberg. Compositional Game Theory with Mixed Strategies: Probabilistic Open Games Using a Distributive Law. *Electronic Proceedings in Theoretical Computer Science*, 323:95–105, September 2020.

[GLR13] Ronen Gradwohl, Noam Livne, and Alon Rosen. Sequential Rationality in Cryptographic Protocols. *ACM Trans. Econ. Comput.*, 1(1), jan 2013.

[GMSR+20] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. SoK: Layer-Two Blockchain Protocols. In *FC*, pages 201–226, 2020.

[GO94] Oded Goldreich and Yair Oren. Definitions and Properties of Zero-Knowledge Proof Systems. *J. of Cryptology*, 7(1):1–32, 1994.

[Gra10] Ronen Gradwohl. Rationality in the Full-Information Model. In *Theory of Cryptography*, pages 401–418, Berlin, Heidelberg, 2010. Springer.

[GS20] Bernhard Gleiss and Martin Suda. Layered Clause Selection for Saturation-Based Theorem Proving. In *IJCAR*, pages 34–52, 2020.

[GV20] Vijay Ganesh and Moshe Y. Vardi. On the Unreasonable Effectiveness of SAT Solvers. In *Beyond the Worst-Case Analysis of Algorithms*, 2020.

[Hal08] Joseph Y. Halpern. Beyond Nash Equilibrium: Solution Concepts for the 21st Century. In *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, PODC '08, page 1–10, New York, NY, USA, 2008. Association for Computing Machinery.

[HBS23] Sebastian Holler, Sebastian Biewer, and Clara Schneidewind. HoRStify: Sound Security Analysis of Smart Contracts. In *CSF*, pages 245–260, 2023.

[HHK+20] Márton Hajdu, Petra Hozzová, Laura Kovács, Johannes Schoisswohl, and Andrei Voronkov. Induction with Generalization in Superposition Reasoning. In *CICM*, pages 123–137, 2020.

[Hir17] Yoichi Hirai. Defining the Ethereum Virtual Machine for Interactive Theorem Provers. In *FC*, pages 520–535, 2017.

[HJ19] Ákos Hajdu and Dejan Jovanovic. solc-verify: A Modular Verifier for Solidity Smart Contracts. In *VSTTE*, pages 161–179, 2019.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In {*USENIX*} *Security*, pages 129–144, 2015.

[HLNW01] Lauri Hella, Leonid Libkin, Juha Nurmonen, and Limsoon Wong. Logics with Aggregate Operators. In *J. ACM*, pages 880–907, 2001.

[How] James Howell. An Overview of Integer Overflow Attacks. available online https://101blockchains.com/integer-overflow-attacks/.

[HSHS20] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. Scaling Blockchains: A Comprehensive Survey. *IEEE Access*, pages 125244 – 125262, 2020.

[IML05] Sergei Izmalkov, Silvio Micali, and Matt Lepinski. Rational Secure Computation and Ideal Mechanism Design. In *FOCS*, pages 585–594, Pittsburgh, 2005. IEEE.

[ISO17] ISO. The JSON Data Interchange Syntax. ISO 21778:2017, International Organization for Standardization, Geneva, Switzerland, November 2017.

[JDM12] Dejan Jovanović and Leonardo De Moura. Solving Non-Linear Arithmetic. In *IJCAR*, pages 339–354, Manchester, 2012. Springer.

[KGDS18] Sukrit Kalra, Seep Goel, Mohan Dhawan, and Subodh Sharma. ZEUS: Analyzing Safety of Smart Contracts. In *NDSS*, 2018.

[KKKT16] Aggelos Kiayias, Elias Koutsoupias, Maria Kyropoulou, and Yiannis Tselekounis. Blockchain Mining Games. In *EC*, pages 365–382, 2016.

[KKS+17] Yujin Kwon, Dohyun Kim, Yunmok Son, Eugene Vasserman, and Yongdae Kim. Be Selfish and avoid Dilemmas: Fork after Withholding (faw) Attacks on Bitcoin. In *CCS*, pages 195–209, 2017.

[KNPS20] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. PRISM-games 3.0: Stochastic Game Verification with Concurrency, Equilibria and Time. In *CAV*, pages 475–487, Cham, 2020. Springer International Publishing.

[KNPS21] Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos. Automatic Verification of Concurrent Stochastic Systems. *Formal Methods Syst. Des.*, 58(1-2):188–250, 2021.

[KNR05] Viktor Kuncak, Huu Hai Nguyen, and Martin Rinard. An Algorithm for Deciding BAPA: Boolean Algebra with Presburger Arithmetic. In *CADE*, pages 260–277, 2005.

[KNT20] Nadim Kobeissi, Georgio Nicolas, and Mukesh Tiwari. Verifpal: Cryptographic Protocol Analysis for the Real World. In *Progress in Cryptology*, pages 151–202, Cham, 2020. Springer International Publishing.

[KSHB19] Akash Khosla, Evan Schwartz, and Adrian Hope-Bailie. Connector Risk Mitigations–Interledger RFCs, 0018, 2019.

[KV13] Laura Kovács and Andrei Voronkov. First-Order Theorem Proving and Vampire. In *CAV*, pages 1–35, 2013.

[Lib99] Leonid Libkin. Logics with Counting, Auxiliary Relations, and Lower Bounds for Invariant Queries. In *LICS*, pages 316–325, 1999.

[LNW+19] Ziyao Liu, Cong Nguyen, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. A Survey on Blockchain: A Game Theoretical Perspective. *IEEE Access*, 7:47615–47643, 2019.

[LPMMS16] Mark H. Liffiton, Alessandro Previti, Ammar Malik, and Joao Marques-Silva. Fast, Flexible MUS Enumeration. *Constraints*, 21(2):223–250, 2016.

[LRKF23] Martina Landman, Sophie Rain, Laura Kovács, and Gerald Futschek. Reshaping Unplugged Computer Science Workshops for Primary School Education. In *Proceedings of 16th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives*, pages 139–151, Lausanne, Switzerland, 2023.

[LT06] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and Adversarial Behavior in Multi-Party Computation. In *Annual International Cryptology Conference*, pages 180–197, Berlin, Heidelberg, 2006. Springer.

[Mac95] D. Mackenzie. The Automation of Proof: A Historical and Sociological Exploration. *IEEE Annals of the History of Computing*, 17(3):7–29, 1995.

[MBB+19] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. Pisa: Arbitration Outsourcing for State Channels. In *AFT*, pages 16–30, 2019.

[MBS+22] Subhra Mazumdar, Prabal Banerjee, Abhinandan Sinha, Sushmita Ruj, and Bimal Roy. Strategic Analysis to Defend against Griefing Attack in Lightning Network, 2022.

[MMSK+17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and Privacy with Payment-Channel Networks. In *CCS*, page 455–471, 2017.

[MMSS+19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *Network and Distributed System Security Symposium*, San Diego, CA, USA, 2019. The Internet Society.

[MMT05] Richard Mckelvey, Andrew McLennan, and Theodore Turocy. Gambit: Software Tools for Game Theory, 2005.

[MSCB13] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In *CAV*, pages 696–701, Berlin, Heidelberg, 2013. Springer.

[Nak08] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf, 2008.

[Nip12] Tobias Nipkow. Interactive Proof: Introduction to Isabelle/HOL. In *Software Safety and Security*, 2012.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In *EuroS&P*, pages 305–320, 2016.

[NM15] Van-Hau Nguyen and Son T Mai. A New Method to Encode the At-Most-One Constraint into SAT. In *Information and Communication Technology*, pages 46–53, New York, NY, USA, 2015. ACM.

[OMA+23] Rodrigo Otoni, Matteo Marescotti, Leonardo Alt, Patrick Eugster, Antti Hyvärinen, and Natasha Sharygina. A Solicitous Approach to Smart Contract Verification. *ACM Trans. Priv. Secur.*, 26(2), mar 2023.

[OR94] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, Cambridge, USA, 1994.

[Pas18] Grant Olney Passmore. Formal Verification of Financial Algorithms with Imandra. In *FMCAD*, pages i–i, 2018.

[PCI+20] Grant O. Passmore, Simon Cruanes, Denis Ignatovich, Dave Aitken, Matt Bray, Elijah Kagan, Kostya Kanishev, Ewen Maclean, and Nicola Mometto. The Imandra Automated Reasoning System (System Description). In *IJCAR*, pages 464–471, 2020.

[PD16] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments, 2016. https://lightning.network/lightning-network-paper.pdf.

[PI17] Grant Olney Passmore and Denis Ignatovich. Formal Verification of Financial Algorithms. In *CADE*, pages 26–41, 2017.

[PR20] Catuscia Palamidessi and Marco Romanelli. Modern Applications of Game-Theoretic Principles. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory (CONCUR 2020)*, volume 171 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:9, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[Pre29] Mojzesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du I congres de Mathématiciens des Pays Slaves*, pages 92–101, 1929.

[PZR20] Daejun Park, Yi Zhang, and Grigore Rosu. End-to-End Formal Verification of Ethereum 2.0 Deposit Smart Contract. In *CAV*, pages 151–164, 2020.

[RAKM21] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. *arXiv preprint*, 2021.

[RAKM23] Sophie Rain, Georgia Avarikioti, Laura Kovács, and Matteo Maffei. Towards a Game-Theoretic Security Analysis of Off-Chain Protocols. In *Proceedings of IEEE 36th Computer Security Foundations Symposium*, pages 31–46, Los Alamitos, CA, USA, 2023.

[RBK+24] Sophie Rain, Lea Salome Brugger, Anja Petković Komel, Laura Kovács, and Michael Rawson. Scaling CheckMate for Game-Theoretic Security. In *Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning*, pages 222–231, Stockport, UK, 2024.

[Sch99] Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience series in discrete mathematics and optimization. Wiley, Hoboken, New Jersey, USA, 1999.

[Sch21] Patrick Schueffel. DeFi: Decentralized Finance - An Introduction and Overview. *Journal of Innovation Management*, 9:i, 12 2021.

[SEM18] Matthew N. O. Sadiku, Kelechi G. Eze, and Sarhan M. Musa. Smart Contracts: A Primer. In *Journal of Scientific and Engineering Research*, 2018.

[SFM+21] Jon Stephens, Kostas Ferles, Ben Mariano, Shuvendu Lahiri, and Isil Dillig. SmartPulse: Automated Checking of Temporal Properties in Smart Contracts. In *IEEE S&P*, 2021.

[SGSM20] Clara Schneidewind, Ilya Grishchenko, Markus Scherer, and Matteo Maffei. eThor: Practical and Provably Sound Static Analysis of Ethereum Smart Contracts. In *CCS*, pages 621–640, 2020.

[Smu95] Raymond M Smullyan. *First-Order Logic.* Dover Publications, New York, 1995.

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. Optimal Selfish Mining Strategies in Bitcoin. In *FC*, pages 515–532, 2016.

[Ste99] Ian Stewart. A Puzzle for Pirates. *Scientific American*, 280(5):98–99, 1999.

[SvS14] Rahul Savani and Bernhard von Stengel. Game Theory Explorer — Software for the Applied Game Theorist. *CoRR*, abs/1403.3969, 2014.

[TJS16] Jason Teutsch, Sanjay Jain, and Prateek Saxena. When Cryptocurrencies Mine their own Business. In *FC*, pages 499–514, 2016.

[TMSS20] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Fritz Schmidt, and Dominique Schröder. PayMo: Payment Channels For Monero. *IACR Cryptol. ePrint Arch.*, 2020.

[TMSS23] Erkan Tairi, Pedro Moreno-Sanchez, and Clara Schneidewind. Ledger-Locks: A Security Framework for Blockchain Protocols Based on Adaptor Signatures. In *CCS*, page 859–873, 2023.

[Vää97] Jouko A. Väänänen. Generalized Quantifiers. In *Bull. EATCS*, 1997.

[VB15] Fabian Vogelsteller and Vitalik Buterin. EIP-20: ERC-20 Token Standard, 2015.

[WCN+21] Scott Wesley, Maria Christakis, Jorge A. Navas, Richard Trefler, Valentin Wüstholz, and Arie Gurfinkel. Compositional Verification of Smart Contracts Through Communication Abstraction. In *Static Analysis: 28th International Symposium, SAS 2021, Chicago, IL, USA, October 17–19, 2021, Proceedings*, page 429–452, Berlin, Heidelberg, 2021. Springer-Verlag.

[WDF+09] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *CADE*, pages 140–145, 2009.

[WLC+19] Yuepeng Wang, Shuvendu K. Lahiri, Shuo Chen, Rong Pan, Isil Dillig, Cody Born, Immad Naseer, and Kostas Ferles. Formal Verification of Workflow Policies for Smart Contracts in Azure Blockchain. In *VSTTE*, pages 87–106, Cham, 2019. Springer International Publishing.

[Woo14] Gavin Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger. https://gavwood.com/paper.pdf, 2014.

[ZBPBS21] Paolo Zappalà, Marianna Belotti, Maria Potop-Butucaru, and Stefano Secci. Game Theoretical Framework for Analyzing Blockchains Robustness, 2021.

[ZCQ+20] Jingyi Emma Zhong, Kevin Cheang, Shaz Qadeer, Wolfgang Grieskamp, Sam Blackshear, Junkil Park, Yoni Zohar, Clark Barrett, and David L Dill. The Move Prover. In *CAV*, pages 137–150, 2020.