Formal Analysis of Graphical Security Models

Aslanyan, Zaruhi; Nielson, Flemming; Probst, Christian W.

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Formal Analysis of Graphical Security Models

Zaruhi Aslanyan

Kongens Lyngby 2016
PhD-2016-421

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

May not music be described as the mathematics of the sense, mathematics as music of the reason? The musician feels mathematics, the mathematician thinks music: music the dream, mathematics the working life.

James Joseph Sylvester

Summary

The increasing usage of computer-based systems in almost every aspect of our daily life makes more and more dangerous the threat posed by potential attackers, and more and more rewarding a successful attack. Moreover, the complexity of these systems is also increasing, including physical devices, software components and human actors interacting with each other to form so-called socio-technical systems. The importance of socio-technical systems to modern societies requires verifying their security properties formally, while their inherent complexity makes manual analyses impracticable.

Graphical models for security offer an unrivalled opportunity to describe socio-technical systems, for they allow to represent different aspects like human behaviour, computation and physical phenomena in an abstract yet uniform manner. Moreover, these models can be assigned a formal semantics, thereby allowing formal verification of their properties. Finally, their appealing graphical notations enable to communicate security concerns in an understandable way also to non-experts, often in charge of the decision making.

This dissertation argues that automated techniques can be developed on graphical security models to evaluate qualitative and quantitative security properties of socio-technical systems and to synthesise optimal attack and defence strategies.

In support to this claim we develop analysis techniques for widely-used graphical security models such as attack trees and attack-defence trees. Our analyses cope with the optimisation of multiple parameters of an attack and defence scenario. Improving on the literature, in case of conflicting parameters such as probability and cost we compute the set of optimal solutions in terms of Pareto efficiency. Moreover, we investigate the relation between attack and attack-defence trees and stochastic models in a verification-oriented setting, with the aim of levering the great many mature tools and analysis techniques developed for instance in the area of games.

Resumè

Den stigende brug af computerbaserede systemer i næsten alle aspekter af vores daglige liv gør truslen fra potentielle angribere mere og mere farlig, og et vellykket angreb stadig mere givende. Desuden er kompleksiteten af disse systemer også stigende, herunder de fysiske enheder, softwarekomponenterne og menneskene der interagerer med systemerne i såkaldte socio-tekniske systemer. Vigtigheden af socio-tekniske systemer i det moderne samfund gør at der kræves formel verifikation af deres sikkerhedsegenskaber, mens deres iboende kompleksitet gør manuel analyse umulig.

Grafiske modeller for sikkerhed tilbyder en uovertruffen mulighed for at beskrive socio-tekniske systemer. De giver mulighed for at repræsentere forskellige aspekter som menneskelig opførsel, beregning og fysiske fænomener på en abstrakt men stadig ensartet måde. Desuden kan disse modeller tildeles en formel semantik, og derved muligøre formel verifikation af deres egenskaber. Endelig gør deres intuitive grafiske notationer det muligt at kommunikere sikkerhedsproblemer på en forståelig måde, også til ikke-eksperter der ofte er ansvarlige for beslutningsprocessen.

Denne afhandling argumenterer for, at der kan udvikles automatiserede teknikker på grafiske sikkerhedsmodeller til at evaluere kvalitative og kvantitative sikkerheds egenskaber af socio-tekniske systemer og til at syntetisere optimale angrebs- og forsvarsstrategier.

For at understøtte denne indsigt har vi udviklet analyseteknikker til udbredte grafiske sikkerhedsmodeller, såsom angrebstræer og kombinerede angrebs- og forsvarstræer. Vores analyse håndterer optimering af flere parametre af et angrebs- og forsvarsscenario. I tilfælde af modstridende parametre som sandsynlighed og omkostninger, forbedrer vi resultaterne i den eksisterende litteratur, da vi beregner det optimale sæt af løsninger i form af Pareto effektivitet. Desuden undersøger vi forholdet mellem angrebstræer og kombinerede angrebs- og forsvarstræer og stokastiske modeller i en verifikations-orienteret ramme, med det formål at udnytte de mange veludviklede værktøjer og teknikker fra stokastiske

spil.

Preface

This thesis was prepared at DTU Compute, the Department of Applied Mathematics and Computer Science of the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree in Computer Science.

The Ph.D. study has been carried out under the supervision of Professor Flemming Nielson and Professor Christian W. Probst in the period from July 2013 to June 2016. Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TRESPASS). This publication reflects only the authors' views and the Union is not liable for any use that may be made of the information contained herein.

Most of the work behind this dissertation has been carried out independently and I take full responsibility for its content. A substantial part of the scientific work reported in this thesis is based on joint work with my supervisors [AN14, AN15, AINP15] and on a collaboration with Dr. David Parker [ANP16] (University of Birmingham, United Kingdom). The relevance of those publications to this thesis shall be clarified in each chapter.

Beyond what mere bibliography tells, the work presented in this dissertation greatly benefited from a prolonged and fruitful interaction with a number of excellent researchers affiliated with the Technical University of Denmark, Aalborg University, University of Luxembourg, University of Hamburg, and the University of Twente, all involved in the TREsPASS research project. Similarly, during my study at the Technical University of Denmark I had the opportunity to discuss with and learn from a great many exquisite researchers that visited the Technical University of Denmark.

Kongens Lyngby, June 2016
Zaruhi Aslanyan

Acknowledgements

First and foremost, I would like to thank my supervisors Flemming Nielson and Christian W. Probst, whose expertise, enthusiasm and support made this work possible. I am truly grateful for their guidance and for engaging in countless fruitful discussions that taught me more than just research. It is impossible to express my gratitude to them.

I should like to thank David Parker, who hosted me in the Computer Security group at Birmingham University. His passion and ever-friendly nature facilitated the practical side of my stay, and his knowledge and curiosity have been key to bring home one of the results in this thesis. It was a pleasure working with him.

I am grateful to the members of my thesis assessment committee, Alberto Lluch Lafuente, Rene Rydhof Hansen, and Axel Legay for accepting to read and review this manuscript, for their valuable comments on the dissertation as well as during the examination.

I would like to thank current and former members of the Language-Based Technology section (now Formal Methods for Safe and Secure Systems) at DTU. Lijun for supervising me during my first period at DTU. Ender and Kebin for being senior examples for my studies. Marieta and Alessandro for colouring my working days and for being always there with any kind of supplies, funny jokes and popcorn included. Marieta and Jose for being unforgettable office mates. Laust and Lars for pretending a genuine interest in my work, always coming up with comments on my presentations that would have filled hundreds of slides, and of course for maintaining the coffee machine. Cathrin, Hanne, Sebastian, Jan, Omar, Ximeng, Hugo, Michal, Nataliya, Andreas for creating a motivating and friendly working environment which allowed both for scientific discussions and for social activities.

My most affectionate thoughts go to my parents, my sister and my aunt for their love, support and encouragement. They did not understand my work but

they shared the pain and joy together with me. Finally, I am forever indebted to my husband Roberto for his full support, invaluable discussions, endless patient. This work is dedicated to him, without whom this dissertation would have taken longer time and would have been thinner.

Contents

List of Tables

Chapter 1

Introduction

> Life was simple before World War II.
> After that, we had systems.
>
> Grace Hopper [Sch04]

1.1 Challenge

The technological evolution of computers from the early 1950s has impacted their internal structure and organisation, but even more the way and the extent they interact with each other to form computer networks. Distributed computing has found a great many applications and governs our daily life, being used in various areas such as education, financial systems, entertainment, and counting. Beyond the Internet, a further increase in complexity is witnessed in recent developments where classical IT networks are coupled with devices interacting with the physical environment, giving rise to so-called *Cyber-Physical Systems*, exploited for example in the domains of traffic control, health care, and environmental monitoring.

The pervasive role of computers in our life includes supporting processes that manage private information and public-concern infrastructure, from personal correspondence via e-mail and bank accounts to military equipment. In turn, such a wide application is increasing the number of people interested in attacking such systems, thereby making their security a chief concern both for private citizens and public institutions. These circumstances have motivated an ever-

growing interest in the verification of security properties in the last couple of decades. Moreover, due to the size of the systems in scope, any practicable verification technique must be automated in some respect.

Unsurprisingly, the challenges of programming computer behaviour have increased with the shift in paradigm from centralised systems to distributed networks. In a nutshell, the problem consists in ensuring that the processes executed by the system establish the desired outcome, that is, the execution fulfils a specification enjoying given properties. Typical properties include functional properties and non-functional properties such as performance, safety, and security. This is the essence of the problem of system verification: *given a formal model of a system and a property of interest, establishing whether or not the model enjoys the property.*

To make the picture even more intricate, we shall remember that computer systems do not operate in isolation. They are part of the organisation consisting in the combination of the environment they are deployed in, its physical infrastructure, and human actors, resulting in a whole referred to as a *socio-technical organisation.* When it comes to security, all these layers concur to increase the attack surface. In particular, human behaviour proves difficult to formalise and therefore to analyse, while it is clear that a system proven secure without considering the staff who operates it gives little guarantees of proving secure while in operation. Abstracting away the human component in the model does not eradicate from the real world the threats connected to human behaviour.

The inherent complexity of socio-technical organisations impacts not only the modelling, with the need of formalising human behaviour, but also the verification concerns, pushing for moving from *qualitative to quantitative* queries. The cases where a complex system is error-free and its correctness can be unconditionally guaranteed are rare, if there exists one at all. Nonetheless, the answer that a system is insecure does not reveal much information. It is much more informative to investigate *to what extent* the system is secure, in terms of likelihood of an attack, cost to a given attacker profile, and so on.

Moving from qualitative to quantitative properties, however, demands to reason carefully about the objectives a given property formalises. Single-objective queries are simple to understand and are solved by computing the best solution of a given system of constraints. However, in many real-life scenarios the investigation of a single objective is rarely of interest, as conflicting constraints and drivers typically arise. The saying that "security comes at a price" hints at the core of the problem. In queries with multiple objectives, in particular with conflicting objectives, there is no best solution but rather a set of efficient solutions, known as *Pareto frontier.*

One chief example of conflicting objectives is indeed a pillar of the problem of security, that is, the struggle between attackers and defenders. The relevance of some socio-technical systems to a great many stakeholders led to equip them with security mechanisms to counter a number known threats. Hence, a comprehensive framework should allow to model the competing behaviour of attackers

and defenders as part of the system, and should support natively the analysis of their conflicting objectives.

In order to tame the complexity of a socio-technical organisation, graphical models prove useful tools as they enable to reason over a visual representation of a problem. They allow to explore the scenario of interest in an intuitive way, and therefore they are suitable to convey information to non-experts, often at the top of the decision-making chain. In the realm of security, tools like *attack trees* and *attack-defence trees* have found industrial application, where quantitative reasoning plays a crucial role. They allow to model in an abstract yet effective way human behaviour, as well as technical threats of the organisation. Moreover, representing interaction between the attacker and the defender, attack trees and attack-defence trees can be seen as one- and two-player games, respectively. While there is a growing interest in such graphical models for security, this is a young research area if compared to computer security construed as protocol verification, cryptography, and the like.

The thesis of this dissertation is that graphical models for security of socio-technical organisations can be assigned formal syntax and semantics supporting the development of automated verification techniques for the analysis of security properties, both from a qualitative and a quantitative perspective. Such formal models, analyses, and properties encompass both attacker and defender behaviour, and can be relied upon to synthesise optimal strategies, thereby drawing attention to vulnerable fragments of the overall system.

1.2 Contribution

In order to tackle the challenge of formal verification of graphical models for security, we study a number of verification techniques on attack and attack-defence trees.

Attack trees are one of the most popular graphical security models. They are an effective tool for identifying and reasoning about security threats of an attack scenario in a graphical manner, showing how an overall goal (attack) can be described as a propositional combination of sub-goals (basic actions). Attack trees combine visual features with formal semantics, which makes them appealing both to scientists, security professionals, and non-experts. Moreover, further extensions of attack trees for capturing the defender's behaviour have been explored. Attack-defence trees allow to study attack-defence scenarios and represent the possible attacker actions and the corresponding countermeasures a defender can undertake.

A scenario where an attacker bribes an employee of an organisation is described by the attack-defence tree presented in Figure 1.1. The root represents the overall goal of the attacker. The leaves represent basic actions that the attacker or the defender can perform to achieve his/her goal, represented by

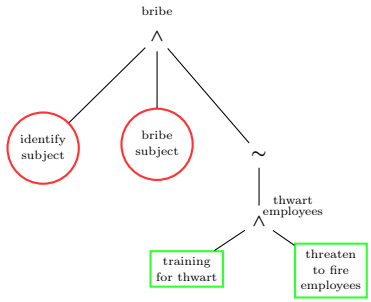


Figure 1.1: An attack-defence tree for bribing an employee.

round or square boxes, respectively. The internal nodes show how these actions can be combined in order to achieve the overall goal.

The analysis of attack and attack-defence trees consists in the evaluation of the scenario they depict from a qualitative or a quantitative perspective. In a qualitative setting, we would enquire whether the overall goal described by the tree is achievable, e.g., whether there exists an attack in the scenario. In a quantitative setting, security attributes are associated with the basic actions (leaves), providing the basics for various types of quantitative analyses, e.g., the likelihood and cost of the attacks in the scenario.

We start this dissertation with a study of attack and attack-defence trees, represented with a simple context-free grammar. As many real-life scenarios require multiple attributes for security analyses, we augment tree models with multiple attributes for the basic actions, like *probability* and *cost*, and present evaluation techniques that address *multi-parameter optimisation*, improving on the existing literature. Moreover, in case of conflicting objectives our techniques compute the set of optimal solutions, defined in terms of Pareto efficiency. We define two evaluation techniques; a natural but expensive *semantic* evaluation, and an *algorithmic* evaluation which enjoys a dramatic improvement in complexity. The evaluations are defined as formal semantics for trees represented in our syntax. We finally show the equivalence of two evaluations under certain conditions.

In the second part of this dissertation, we continue with a deeper treatment of attack-defence scenarios, building on existing results that show a connection between attack-defence trees and two-player games. Indeed, attack-defence trees can be seen as interactions between an attacker and a defender who compete in order to achieve conflicting objectives. While existing literature explores the theoretical connection between these two concepts, we investigate the applicability of game verification results to attack-defence scenarios. This opens up to take advantage of a mature line of research including the support of a number of tools for automated verification of games.

In particular, we translate attack-defence trees into Stochastic Two-player

Games (STGs), and evaluate their security properties by means of PRISM-games, a state-of-the-art model checker for stochastic games. The framework we develop formalises the notion of strategy for a player as a decision tree with an appealing graphical notation, which recalls the distinguishing trait of graphical models.

In our investigation we identified a limitation in the logics used to express security properties in one- and two-player games. In particular, Probabilistic Computation Tree Logic with Rewards (rPCTL) and Probabilistic Alternating-Time Temporal Logic (rPATL) do not evaluate the cost of an execution without considering the probability, i.e., the *exact cost* of an execution. Hence, in the third and last part of this dissertation, we extend rPCTL to overcome such limitation. We investigate both cost-related properties as well as multi-objective properties concerning probability *and* exact cost. The extended logic can answer the questions such as "what is the minimum cost of a successful attack?", "what is the maximum probability of cheap attacks?", or "is the cost of attacks with probability 0.7 within given cost bounds?". We propose a model checking algorithm for the new operators we introduce. We carry out these developments in the simpler setting of attack trees, exploring their connection with Markov Decision Processes (MDPs).

1.3 Synopsis

In this section, we present the organisation of the dissertation. For each part that we have presented in Sect. 1.2 we give a brief account of its chapters and their connection.

Part I: Evaluation of Attack and Defence Scenarios
The first part of the dissertation studies the evaluation of attack trees and attack-defence trees.

Chapter 2 reviews existing literature on graphical models for analysing security scenarios, in particular focusing on attack trees and their extension with countermeasures, attack-defence trees. The chapter discusses the existing evaluation techniques on tree models and highlights their limitations.

Chapter 3 introduces the formal syntax of attack trees which we use throughout the dissertation. Focusing on the optimisation of attack trees with multiple parameters for basic actions, the chapter presents evaluation techniques that optimise all parameters at once, levering the concept of Pareto efficiency. The chapter is based on [AN14, AINP15].

Chapter 4 builds on Ch. 3 by extending the model of attack trees with defender actions and presenting evaluation techniques for attack-defence trees with multiple parameters. It defines the formal syntax of attack-defence trees by introducing the novel changing player operator. The chapter is based on [AN15].

Part II: From Attack-Defence Trees to Security Games
The second part of the dissertation discusses the connection between attack-defence trees and Stochastic Two-player Games.

Chapter 5 provides background material on probabilistic models, in particular Discrete-Time Markov Chains and stochastic games, and their evaluation by means of probabilistic model checking.

Chapter 6 proposes a framework for a formal quantitative evaluation of attack-defence scenarios with dependent actions, presented by an extended model of attack-defence trees with sequential operators. The evaluation of attack-defence trees exploits a game-theoretic approach. The chapter presents the translation from attack-defence trees to Stochastic Two-player Games, and employs probabilistic model checking to analyse the model. The chapter is based on [ANP16].

Part III: The Logic erPCTL for Attack Trees
The third part of the dissertation explores the relation between attack trees and probabilistic models and presents the extended logic with exact cost operators.

Chapter 7 gives preliminaries on Markov Decision Processes and the probabilistic temporal logic rPCTL for formalising the properties of MDPs. Moreover, the chapter discusses a model checking algorithm for MDPs. Finally, we present an evaluation technique for attack trees through Discrete-Time Markov Chains.

Chapter 8 extends rPCTL with new operators allowing to specify cost-related properties. Investigating the probability and cost properties of an attack scenario, the chapter proposes a quantitative evaluation of attack scenarios through Markov Decision Processes resorting to model checking techniques. The chapter is based on [AN17].

Chapter 9 presents some concluding remarks and highlights research directions that are open for future work.

On the whole, Chs. 2, 5 and 7 give the essential background to position this dissertation, while the main contribution of the thesis is presented in Chs. 3, 4, 6 and 8.

Part I

Evaluation of Attack and Defence Scenarios

As we have discussed in the introduction, ensuring security of socio-technical organisations is as challenging as crucial in today's world. The distributed nature of these systems increases the attack surface including cyber, physical and "social" targets. Hence, automated techniques that support the risk assessment process are required. Finally, the system under investigation and the result of the analysis must be communicated to end users, who often are not security experts.

To cope with the challenge of developing automated risk assessment techniques and produce accessible results, various formal graphical models have been studied. Graphical models prove useful tools for presenting both technical and social threats and conveying security information to non-experts.

Models like attack trees and attack-defence trees are a promising approach for representing threat scenarios and possible countermeasures in a concise and intuitive manner. An attack tree represents attacks in an attack scenario, while an attack-defence tree describes the interaction between an attacker and a defender in an attack-defence scenario. They are evaluated by assigning parameters to the leaves, such as probability or cost of attack or defence. In case of multiple parameters most analytical methods optimise one parameter at a time, e.g., maximising the probability *or* minimising the cost of an attack. Such methods may lead to sub-optimal solutions when optimising conflicting objectives, e.g., maximising probability *while* minimising cost.

In order to tackle this challenge and evaluate complex attack and attack-defence scenarios with more than one parameter, we devise automated techniques that optimise all parameters at once, thus computing different aspects of an attack and handling multiple objectives. Moreover, in the case of conflicting objectives our techniques compute the set of all optimal solutions, defined in terms of Pareto efficiency. The developments are carried out on a new and general formalism for attack trees and attack-defence trees.

Part I is organised as follows. Ch. 2 reviews existing literature on security modelling and risk assessment of attack and defence scenarios. Our techniques for evaluating attack trees with multiple parameters are presented in Ch. 3. Ch. 4 extends the developments of Ch. 3 to attack-defence trees.

Chapter 2

Preliminaries : Graphical Models for Security Analyses

Formal graphical models are useful tools for security analysis. They allow to convey information to non-experts in an intuitive, visual and user-friendly way, meanwhile attracting the attention of security experts and scientists as they can be assigned a formal semantics. In turn, formal semantics support the development of automated verification techniques for evaluating both qualitative and quantitative security aspects of a system. Moreover, both the strengths of graphical security models, that is, visualisation and verification, are supported by various tools that facilitate risk management and threat assessment of complex attack and defence scenarios.

Different graphical approaches for evaluating the properties of attack and defence scenarios have been studied. In this work we will focus on the formalisms based on undirected acyclic graphs, in particular on attack and attack-defence trees.

A historical overview on existing graph-based approaches for security threats is given by Piètre-Cambacédès and Boussou [PB10]. Moreover, Kordy et al. summarise the existing methodologies for analysing attack and defence scenarios in [KPS14a].

This chapter gives a review of the literature on attack and defence graphical modelling approaches. We start with a short presentation of the first graphical approaches in Sect. 2.1. Sect. 2.2 gives a review of the attack modelling tech-

niques, while the modelling approaches for attacks and defences are described in Sect. 2.3. Finally, the concept of Pareto efficiency, heavily exploited in our developments, is discussed in Sect. 2.4.

2.1 First Graphical Models

The first graphical model aimed at representing graphically the safety and the reliability of a system was introduced in the 1960's, leading to so-called *fault trees*. Fault trees represent a system failure in terms of the failure of its components [VRHG81]. The root of the tree represents the main failure of the system, while the basic actions (the leaves of the tree) correspond to the component failure. Fault trees have been used in various domains such as aerospace, nuclear power and electrical power (for a list of references refer to the bibliography of [MAV+13]).

Fault trees inspired a similar approach to security. In 1991, Weiss used trees in security analysis and presented *threat-logic trees* as the first graphical attack-modelling technique [Wei91]. They are used to identify the potential threats to the system. The model follows top down approach breaking down the high-level threat corresponding to the root of the tree into smaller threats that comprise an attack scenarios. However, threat-logic went without much notice and few years later, in 1998, Salter et al. [SSSW98] and a year later Schneier [Sch99] introduced *attack trees* retracing the core idea of threat trees. Even though attack trees were first mentioned by Salter et al., however, the term is often credited to Schneier.

2.2 Attack Modelling Techniques

2.2.1 Attack Trees

In 1999, Schneier introduced attack trees as a tool to evaluate the security of complex systems in a structured, hierarchical way. Attack trees allow to analyse the possible attack scenarios and reason about the security of the whole system in a methodical way, by splitting a complex goal into sub-goals and basic attacks [Sch99].

Basics. The root of an attack tree represents the overall goal of the attacker. The leaves represent the basic actions that an attacker might perform in order to achieve his/her goal. The internal nodes represent the sub-goals of the tree and refine conjunctively or disjunctively the sub-trees or actions. A conjunctively-refined node is satisfied if all of its children are satisfied. A disjunctively-refined node is satisfied if at least one of its children is satisfied.

Figure 2.1 displays a simplistic attack tree, where the goal is to force a card-holder to pay. The root of the tree is labelled with the main goal. The goal

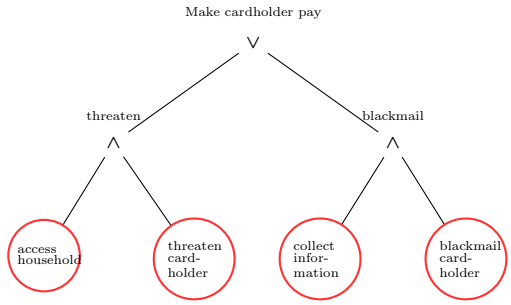


Figure 2.1: Attack tree for forcing the cardholder to pay.

is further divided into two possible sub-goals: threatening and blackmailing. As the main goal is satisfied if at least one of sub-goals is satisfied, the root is a disjunctively-refined node. In order to succeed in threatening, the attacker should access the household and threaten the cardholder, while the attacker needs to collect information and blackmail the cardholder for a successful blackmailing.

The attack tree model have been further extended. In particular, Mauw and Oostdijk [MO05] give a formal foundation of attack trees. They define a formal semantics of attack trees in terms of attack suites, which are collections of possible basic actions.

Verification. In order to analyse attack trees, different kind of attributes (such as probability or cost) are assigned to the leaves of a tree. By having values for the basic actions the value for the root can be computed. The standard method for evaluating attack trees is a bottom-up algorithm, where the values are propagated from the leaves up to the root. During the years, researchers have proposed various evaluation methods and analysed different security aspects, such as feasibility of an attack, attack cost and probability of success.

The representation of an attack scenario through an attack tree and its evaluation provides the overall picture of the security of the system under study. In order to make this picture realistic, some knowledge about the attacker is required, as different attackers have different levels of skills, budget, etc. This knowledge can be applied in the modelling stage of attack trees, for example identifying the likelihood of basic actions on account of the attacker skills. Moreover, as mentioned by Schneier: "The characteristics of your attacker determine which parts of the attack tree you have to worry about". Thus, the result of an attack tree analysis can be further refined on the basis of the attacker profile. For instance, knowing the attacker's budget, the expensive attacks can be eliminated from the result.

Attack trees are supported by various tools for modelling and evaluating attack scenarios, both academic tools such as SeaMonster [Mel10, MGE+08], and

commercial software like SecurITree [Ame] from Amaneza and AttackTree+ [Iso] from Isograph. Different tools implement different analyses. For example, it seems that SecurITree is centred around the notion of risk of an attack, while AttackTrees+ also reasons about probability.

Attack trees have been applied to the evaluation of various case studies. For example, the security of the Estonian and the USA e-voting systems have been studied in [BM07], while the evaluation of the border getaway protocol with attack tree techniques is presented in [CCF04]. Moreover, in [BFM04, TLG07] attack trees are used to identify possible attacks in Supervisory Controls And Data Acquisition (SCADA) systems, and to suggest possible improvements based on the identified vulnerabilities.

Generation. Historically, attack trees are constructed manually with the help of experts. The construction starts by identifying the main attack goals of the scenario, which are further decomposed into smaller sub-goals until reaching the basic attacks. This process can be repeated by modifying or adding sub-goals and basic actions to the tree. However, the manual construction of attack trees for large systems is error-prone and complex. Various approaches have been proposed for the automated generation of attack trees from formal specifications of attack scenarios, for example, automated deduction of attack trees from a process-algebraic specification [VNN14], attack tree generation by policy invalidation [IPHK15b, IPHK15a] and industry-based process-oriented construction and maintenance of big attack trees [Pau14].

Attack graphs. Attack graphs have been introduced at the same time as attack trees by Philips and Swiler [PS98]. They are mostly used for network security analysis, and this could have favoured the usage of graphs as opposed to trees, for network models are naturally represented with (cyclic) graphs. The nodes of the graph represent possible attack states, while edges represent a change of state due to an attacker's action.

Based on Philips and Swiler's model further studies on attack graphs have been carried out. Sheyner et al. [SHJ+02, JSW02] developed a model-checking based approach for automated generation and analysis of attack graphs, where attack graphs are characterised as counter-example to safety properties. A toolkit for generating and exploring attack graphs is developed on [SW04], while a logic-based approach for representing and generating attack graphs is presented in [OBM06, Sah08]. Ammann et al [AWK02] proposed more compact representation of attack graphs relying on explicit assumption of monotonicity of the attacker actions.

2.2.2 Extensions of Attack Trees

Attack trees discussed so far have conjunctive (denoted by AND) and disjunctive (denoted by OR) refinements for internal nodes. Further extensions of this

model with other type of refinements have been studied.

Yager [Yag06] proposed an extension of attack trees called OWA trees. In these extended trees AND and OR nodes are replaced with an Ordered Weighted Averaging (OWA) node. The new node corresponds to quantifiers such as some, half of, most, etc. OWA trees are suitable to model uncertainty and to reason about the scenario when the exact number of satisfied sub-goals is unknown.

For modelling the situations when exactly one out of n children is necessary for a successful attack, attack trees have been extended with XOR node [MDTG07]. A XOR node is satisfiable when exactly one of its children is satisfiable. An attack tree with a XOR node is used to model spam over internet telephony attack scenarios [MDTG07].

Several studies propose attack tree models with an order on the performance of basic actions. For modelling the sequential-dependencies of attacks and ordering them, new nodes such as *priority-AND*, *k-out-of-n* and *sequential enforcing* have been studied in [Kha09]. Similarly, attack trees have been extended with a sequential conjunction operator that considers a precise order on the execution of the basic actions in the tree [LL11]. Jhawar et al. [JKM$^+$15] give a formal semantics of attack trees with sequential conjunction calling the model SAND attack trees.

It is worth noticing that the literature mentioned so far takes a static view on the security model, where time does not play any role. Other models that naturally allow to consider time have been investigated in connection to security analyses, such as timed automata and Continuous-Time Markov Chains. However, this is the case of well-established general-purpose models, verification techniques and tools applied to security problems and not the case of modelling and analysing graphically appealing objects. We will briefly discuss these approaches in Ch. 6 in connection to our study of games.

2.2.3 Multi-Parameter Attack Trees

Basic attack trees first introduced in the literature enriched basic actions with a single parameter of interest, such as probability, cost, time, and so on. More recent works have lift this limitation looking at multi-parameter trees. This extension, however, makes the analysis of attack trees much more challenging as it leads to multi-parameter optimisation problems.

Buldas et al. [BLP$^+$06] proposed a risk-analysis based method for security evaluation of attack trees against a gain-oriented attacker, i.e., an attacker who chooses only profitable attacks. The resulting model is called *multi-parameter attack trees*. In the model the authors consider multiple security parameters of the system such as gain of the attacker, probability of success, probability of getting caught and expected penalties. On the basis of these parameters this method computes the overall value of an attack in a bottom-up fashion. The computation considers precise values for the parameters. An extension of this method which considers an interval estimation for the parameters is proposed

in [JW07].

The method has some limitations. The value for the gain parameter is constant throughout the tree computation. Moreover, the computation of disjunctive nodes is based on local decisions, i.e., the outcome value of disjunctive nodes is only based on the outcome value of their children.

Jürgenson and Willemson [JW08] proposed a new computation model for multi-parameter attack trees. Instead of using a bottom-up approach for the computation, the authors evaluate the outcome value for each attack suite. An attack suite is a set of basic actions leading to a successful attack. The analysis is done by considering all attack suites and computing the outcome for each of them. In the end, the attack suite with the highest outcome is considered. As the computation considers all possible attack suites, the complexity is exponential.

To improve the complexity of the computation, the authors propose an approximation technique based on genetic algorithms [JW10]. The genetic algorithm essentially computes a lower bound to the attacker's outcome. The complexity of their genetic algorithm is $\mathcal{O}(n^4)$, where n is the number of leaves.

Another extension of the multi-parameter attack tree model [JW09] enriches the attacker's decision-making process with a temporal order. A *serial model* for attack tree computation considers basic actions (attacks) in a given order and computes the outcome of an attack according to the defined order on the basic attack set.

2.3 Attack and Defence Modelling Techniques

While attack trees focus on evaluating attack scenarios, other tree-like representation also incorporate countermeasures.

2.3.1 Defence Trees

Defence trees, introduced by Bistarelli et al., constitute the first approach towards combining attacker's and defender's behaviour in tree-like models [BFP06]. They are an extension of attack trees with defender actions, where each attacker basic action (leaf of the tree) is associated with a set of corresponding countermeasures. Thus, defender actions occur only at the leaves of the tree and not in the internal nodes, as opposed to attacks (an internal node represents a sub-goal of the attacker).

Defence trees are used to evaluate effectiveness and economic profitability of countermeasures. These analyses are based on two economic indexes; return on investment (ROI), which is used to determine cost-effective countermeasures, and return on attack (ROA), which is use to determine the best attack strategies. The computation of ROI and ROA is based on parameters such as cost, impact, etc.

Further extensions of the model into game theory have been studied. The authors proposed a game-theoretical reasoning for evaluating defence trees in [BDP06]. The analysis selects the best strategies for the attacker and the defender in terms of Nash equilibrium.

2.3.2 Protection Trees

Edge et al. [EDRM, ERG+07] proposed a methodology for allocating appropriate protections against specified attacks such that the success probability of the defender is maximised. *Protection trees* have the same structure of attack trees with the difference that nodes represent protections. A protection tree is generated from an existing attack tree by placing protections against corresponding attacks. Thus, as a result we obtain two trees, an attack tree and corresponding protection tree. The root of the protection tree directly corresponds to the root of the associated attack tree, however this is not true for the rest nodes of protection and attack trees, for protections and attacks are not necessarily mapped one-to-one.

The evaluation of both trees are done by using bottom-up algorithms and considering parameters such as probability, cost or impact. The formalism has been applied to various case studies, including the U.S. Department of Homeland Security [EDRM], online banking [ERG+07] and an RFID network [IEMR10].

2.3.3 Attack Countermeasure Trees

Another extension of attack trees with defender actions is *attack countermeasure trees* proposed by Roy et al. [RKT10b, RKT10a, RKT12]. The countermeasures in the tree are presented in the form of detection and mitigation events, and can be applied at any node of the tree as oppose to defence trees, discussed in Sect. 2.3.1. Moreover, together with conjunction and disjunction nodes, the model is extended with a k-out-of-n node.

For evaluating attack countermeasure trees qualitative and probabilistic analyses have been developed. The qualitative analysis of an attack countermeasure tree exploits minimum cut sets to determine the possible ways of attacking and defending a system and to identify the system's most critical components. The probability analysis is used to evaluate security aspects such as probability, cost, impact, Birnbaum's importance measure and risk, as well as ROI and ROA indexes from defence trees.

2.3.4 Attack-Response Trees

Attack-response trees [ZKSY14] are another extension of attack trees with response (countermeasure) actions. In attack-response trees an internal node or a leaf is tagged with the corresponding response, and the root of the tree is evaluated by considering the set of attacks and responses.

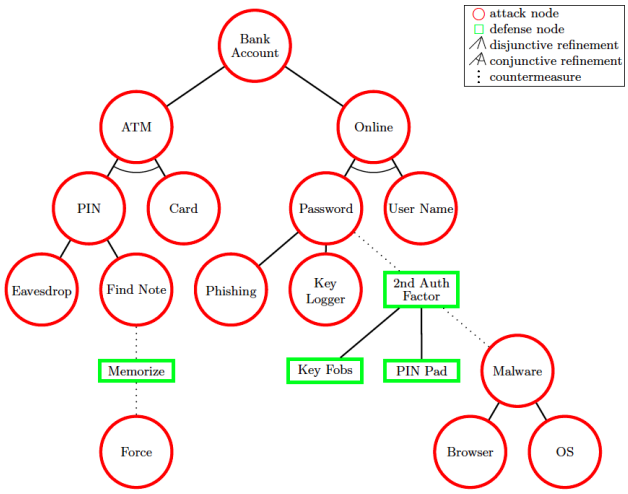


Figure 2.2: An attack-defence tree for attacking bank accounts.

Attack-response trees are used to describe and analyse intrusion and response scenarios. They have been developed as a part of an automated game-theoretic intrusion response and recovery engine, which identifies the optimal response actions from partially observable competitive Markov Decision Processes constructed from attack-response trees.

2.3.5 Attack-Defence Trees

Kordy et al. [KMRS10, KMRS14] formalised attack-defence trees as an intuitive model for presenting attacks and countermeasures in a single view. Attack-defence trees are extensions of attack trees with countermeasures. They illustrate in a graphical way the possible actions an attacker can perform in order to attain a given goal, and the feasible countermeasures a defender can undertake to counter such actions. They can be seen as an interaction between two players: an attacker and a defender.

The graphical representation of an attack-defence tree is similar to the one of an attack tree. The root node represents the main goal of one of the players. The leaves represent the basic actions and the intermediate nodes refine subtrees either conjunctively or disjunctively. Moreover, each node might have at most one child of the opposite type. Figure 2.2 illustrates an example of attack-defence tree for attacking bank accounts.

For evaluating attack-defence trees the typical bottom-up approach of attack trees is extended, i.e., the evaluation assigns values to the parameters of the leaves and the tree is traversed from the leaves to the root. The evaluation considers the basic actions to be independent. Attack-defence trees are inter-

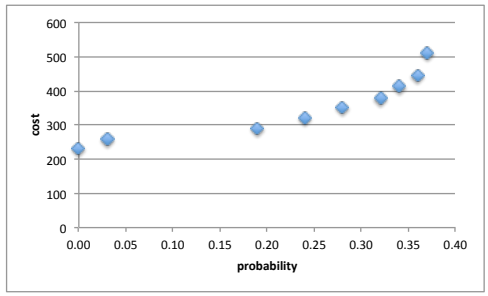


Figure 2.3: Pareto-efficient solutions for the attack tree t in Ch. 3.

preted with various semantics to answer questions such as the vulnerability of the system to an attack or the minimum cost of an attack. Most evaluations [KMRS10, KPS11, KPS14b] analyse a specific aspect of a scenario and do not consider trees with multiple parameters.

Further developments on attack-defence trees have been explored. The relationship between attack-defence trees and games has been studied in [KMMS10]. The work shows that attack-defence trees with satisfiability attributes and two-player binary zero-sum extensive form games have equivalent expressive power. Moreover, [KPS14b] combines the tree methodology with Bayesian networks for analysing probabilistic measures of attack-defence trees with dependent actions.

Attack-defence trees have been used to evaluate a real-life RFID good management system [BKMS12]. This case study resulted also in guidelines for the use of parameters for basic actions in attack-defence trees. The attack-defence tree methodology is supported by a tool called ADTool [KKMS13], which allows to model and analyse attack-defence scenarios.

2.4 Pareto Efficiency

Any analysis of attack or attack-defence models starts from identifying the parameters of interest that characterise the basic actions. As we have already put forward, real-life scenarios would often require to consider multiple parameters, leading to the challenge of dealing with conflicting aspects.

Throughout this dissertation, we will use probability of success and cost of performing an action as a paradigmatic example of conflicting goals, for any rational player would like to maximise one while minimising the other. The challenge arises as in the context of multi-objective optimisation with conflicting objectives there is no one best solution but rather a set of incomparable ones.

In order to formalise this intuition, we resort to the notion of Pareto efficiency. The concept was developed by the economist Vilfredo Pareto and have been widely used in various fields such as economy, engineering and the life sci-

ences. A solution is called Pareto-efficient if it is not dominated by any other solution in the objective space [LGCM10, MG13]. The set of Pareto-efficient solutions is known as Pareto frontier.

Figure 3.3 displays the Pareto frontier obtained in the study of an example in Ch. 3. Each point corresponds to one solution, characterised by a probability and a cost, on the x and y axis, respectively. Observe that each solution in the Pareto frontier has either higher probability or lower cost than other points in the frontier, hence it is not worse or better than the others and might be appealing to a given player profile. On the other hand, observe that all points that are not on the frontier, are either not a solution or are solutions not worth being considered by any rational player, as they can be improved in either component.

Chapter 3

Pareto Efficient Solutions of Attack Trees

Securing socio-technical systems against the threat of attackers is a crucial problem, which becomes increasingly difficult as attacks become more sophisticated and systems more complex. This necessitates a thorough investigation of the possible attack scenarios of a system. As we have discussed in Sect. 2.2, attack trees are a powerful tool for modelling security threats of a system and representing attack scenarios in an intuitive manner.

Attack trees are typically analysed by assigning values to the basic actions and propagating them from the leaves to the root of the tree, so that the value at the root describes the entire attack scenario. Most attack tree analyses consider attack trees with one parameter and therefore optimise one particular aspect of a scenario, such as likelihood of success *or* difficulty of a hack, for instance in terms of time or cost of an attack. Moreover, even in those attack tree models enhanced with multiple parameters, values are propagated from the leaves to the root applying local decision strategies, i.e., in each step of the evaluation the optimisation is made with respect to one parameter. In case of conflicting objectives, however, this approach may yield sup-optimal results.

In order to overcome this limitation and analyse complex attack scenarios, we present evaluation techniques that consider attack trees where basic actions are assigned more than one parameter, such as likelihood of success *and* cost. Our evaluation techniques optimise all parameters at once. In order to cope with the optimisation of multiple conflicting objectives in partially-ordered sets we compute the set of optimal solutions, defined in terms of Pareto efficiency. In this way we can for example maximise the likelihood of possible attacks while

minimising their cost.

Our developments are carried out on a new language-based formalism for attack trees. We study the problem in the settings of a Boolean and a probabilistic semantics for attack trees. For each semantics, we first consider the problem of feasibility of the attack, and then we extend our techniques to compute optimal attacks in presence of probabilities and multiple costs. Moreover, for each case, we first define the solution considering all possible choices of the attacker, obtaining a natural but exponential characterisation. Then, we improve on the complexity devising an algorithmic evaluation that is linear in the size of the tree and yet sound for an expressive sub-class of models. We illustrate our developments on a home-payment system case study and we discuss a more complex example that we have investigated through a proof-of-concept implementation of our analysis.

We introduce our formalism for attack trees and provide evaluation techniques for feasibility queries in Sect. 3.1. We extend the model with a single cost per action and present the computation of minimum cost in Sect. 3.2. Sect. 3.3 extend the single cost model to multiple costs. Finally, the proofs are reported in Appendix A. This chapter is mainly based on [AN14, AINP15].

3.1 Formal Model of Attack Trees

In this section, we present our formalism for attack trees. We start by defining the syntax and the terminology used throughout the chapter. Then, we describe the evaluation techniques for investigating the feasibility of attacks both in Boolean and probabilistic settings. The Boolean case is thoroughly explained in Sect. 3.1.2. The developments are generalised to the probabilistic setting in Sect. 3.1.3.

3.1.1 Attack Trees

An attack tree is a graphical representation of an attack scenario. The root of the tree represents the main goal of the attacker. The leaves of the tree represent the basic actions that the attacker can perform in order to achieve his/her goal. The internal nodes show how the basic actions can be combined. For the sake of simplifying the technical developments, we assume that the actions are independent.

The abstract syntax of an attack tree t is presented in Table 3.1. A tree is either a leaf or the application of a tree operator to one or two sub-trees. A leaf a is a basic action of the attacker. We denote the set of basic actions by Act and the set of attack trees by $Tree$.

The special leaves `true` and `false` represent a trivially-successful and a trivially-failed action, respectively.

Table 3.1: The syntax of an attack tree.

$$t \quad ::= \quad a \quad | \quad \&_{\wedge}(t_1, t_2) \quad | \quad \&_{\vee}(t_1, t_2) \quad | \quad \&_{\neg}(t) \quad | \quad \&_{\texttt{true}} \quad | \quad \&_{\texttt{false}}$$

As standard in the literature, tree operators include conjunction and disjunction, while we introduce negation. The conjunction operator $t = \&_{\wedge}(t_1, t_2)$ requires that the goals of t_1, t_2 are achieved in order for the goal of t to be achieved. The disjunction operator $t = \&_{\vee}(t_1, t_2)$ requires that the goal of at least one sub-tree is achieved in order for the goal of t to be achieved.

The negation operator $t = \&_{\neg}(t')$ requires that the goal of the sub-tree t' is not achieved in order for the goal of t to be achieved. This operator negates the goal of t' and it is handy for modelling purposes: even if basic actions can be defined with a negative flavour, the possibility to negate an entire sub-tree enhance its human-readability. Moreover, it is sometimes simpler to define cost and probability of success in case of occurrence of an event as opposed to estimate these values for enforcing its absence. For instance, cutting a communication wire might be unrecoverable, and after having cut a wire a player might not be able to communicate with a given device.

It is worthwhile observing that the operators that we have discussed are tree operators and do not necessarily correspond to their propositional counterparts. Hence the choice of & in the syntax.

Polarity-consistent tree. We introduce the notion of polarity-consistency, to be exploited in the technical developments. We say that an action a occurs negatively in a tree, if a is under an odd number of negations. Otherwise, we say that an action a occurs positively. Such *polarities* are denoted with the symbols $-$ and $+$, respectively.

Definition 3.1 An attack tree t is polarity-consistent iff there is no action that occurs both positively and negatively in t.

A sufficient (but not necessary) condition for polarity-consistency is that all actions are "uniformly good" or "uniformly bad" for the proponent. If t is a polarity-consistent tree, then the polarity of each action is uniquely determined.

Example. We demonstrate our developments on the example of a home-payment system, inspired by [IPHK15a] and studied in the project TREsPASS [The14]. A home-payment system allows people, who may have difficulties leaving their home, to manage some kind of services, e.g., care-taking or rent. The payment services are performed using the remote control of a television box by means of a contact-less payment card to authenticate users and to provide

payment capabilities. The card-holder owns a card with a password for initiating transfers. We assume that the cardholder is an elderly person who stays at home most of the time and has a trusted person who visits and helps him/her from time to time.

The attack scenario that we consider is to steal money from the card-holder by forcing him/her to pay for fake services. This goal is the root of the corresponding attack tree. In order to steal money from the card-holder, the attacker can "blackmail" or "threaten". In order to succeed in blackmailing the attacker has to collect necessary information and blackmail the card-holder. For a successful threatening the attacker should threaten the cardholder and access the household. The latter is achievable in different ways. The complete corresponding attack tree is given in Figure 3.1, where we label leaves for ease of reference. For the sake of clarity, in the figures we denote internal node operators with their subscript symbols. The syntactic term of the full tree is:

$$t = \&_\vee(\&_\wedge(\&_\vee(ip, \&_\vee(\&_\vee(\&_\wedge(\&_\vee(rc, rt), it), itt), itp)), tc), \&_\wedge(ci, bc))$$

3.1.2 Semantics in the Boolean Case

In this section we formalise the semantic of attack trees in the Boolean setting. We evaluate the feasibility of an attack tree and investigate questions such as "is there a successful attack?" by determining possible values at the root. We associate with each basic action a value from the Boolean set $\mathbb{B}$, where true (tt) corresponds to performing and false (ff) corresponds to not performing the action. We consider $\mathbb{B}$ to be ordered such that $\max\{tt, ff\} = tt$ and $\min\{tt, ff\} = ff$. In this setting, the value true at the root denotes that the attacker has at least one combination of basic actions that leads to the overall attack goal.

We define a Boolean assignment of basic actions as follows: a Boolean assignment m is an arbitrary function that assigns a value $b \in \mathbb{B}$ to each basic action $a \in Act$; $m : Act \to \mathbb{B}$. We say that the attack is successful if the Boolean assignment evaluates the tree to *true*.

It is worthwhile noticing that in the following the conjunction, disjunction and negation operators are interpreted as the corresponding Boolean operators, and in this setting the Boolean laws, i.e., commutativity, associativity, distributivity and idempotency hold. Hence, the following two trees, $t_1 = \&_\wedge(a, \&_\vee(b, c))$ and $t_2 = \&_\vee(\&_\wedge(a, b), \&_\wedge(a, c))$ are equivalent, meaning that the same assignment evaluates t_1 and t_2 identically.

We present two evaluation techniques for feasibility analysis of an attack scenario, termed semantic and algorithmic evaluations.

The *semantic evaluation* $M(t)$ of an attack tree $t \in Tree$ is presented in Table 3.2. The evaluation of a tree t is performed over all possible Boolean assignments of values to the basic actions of t. It computes the minimum and maximum success values of an attack.

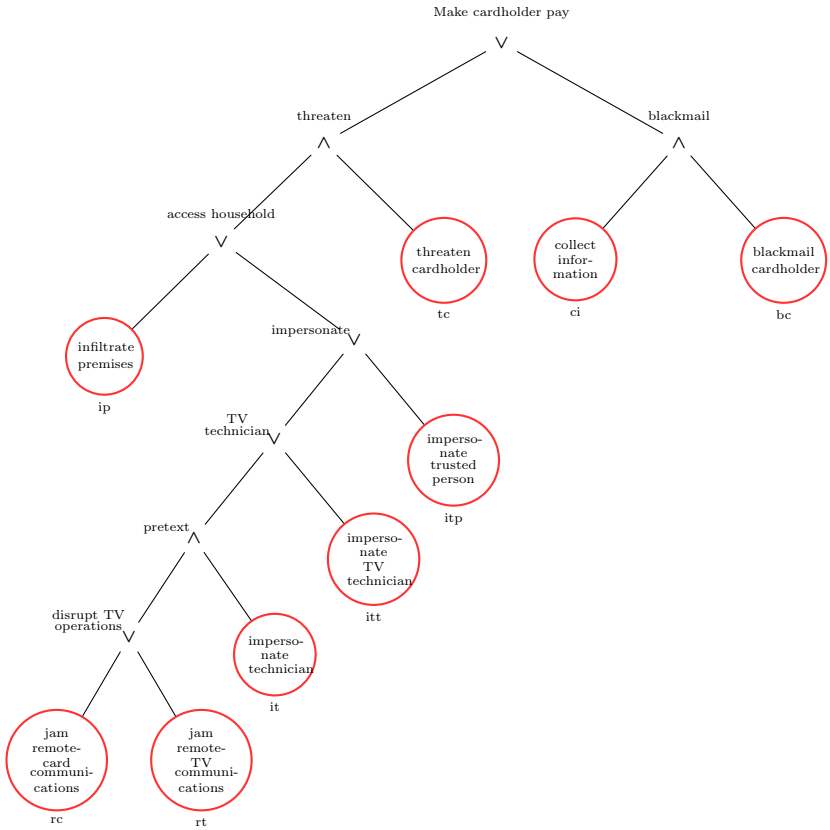


Figure 3.1: Attack tree for forcing the cardholder to pay.

The call $\mathcal{B}[\![t]\!]m$ of the recursive function $\mathcal{B}[\![\cdot]\!]m$, displayed in the second part of Table 3.2, analyses the tree t over the Boolean assignment m. Observe that the negation operator changes the goal of the attacker in terms of optimisation objectives.

Observe that the minimum value gives information about all attackers, e.g., if any attacker can attack the system, while the maximum value gives information about some attackers, e.g., if there exists an attacker that can attack the system. The result of the evaluation is interpreted as follows:

- If both the minimum and the maximum values of t are *false*, $M(t) = (\mathit{ff}, \mathit{ff})$, then the system is always secure despite the attacker's actions.
- If the minimum value of t is *false* and the maximum value of t is *true*, $M(t) = (\mathit{ff}, \mathit{tt})$, then the system is vulnerable. In other words, there exist a set of actions (a Boolean assignment m) such that an attack on the

Table 3.2: The Boolean semantic evaluation of an attack tree.

$$M(t) = (\min\{\mathcal{B}[\![t]\!]m \mid m \text{ Boolean assignment}\}, \max\{\mathcal{B}[\![t]\!]m \mid m \text{ Boolean assignment}\})$$

$\mathcal{B}[\![a]\!]m$	$=$	$m(a)$
$\mathcal{B}[\![\&_\wedge(t_1,t_2)]\!]m$	$=$	$\mathcal{B}[\![t_1]\!]m \wedge \mathcal{B}[\![t_2]\!]m$
$\mathcal{B}[\![\&_\vee(t_1,t_2)]\!]m$	$=$	$\mathcal{B}[\![t_1]\!]m \vee \mathcal{B}[\![t_2]\!]m$
$\mathcal{B}[\![\&_\neg(t)]\!]m$	$=$	$\neg\mathcal{B}[\![t]\!]m$
$\mathcal{B}[\![\&_{\texttt{true}}]\!]m$	$=$	tt
$\mathcal{B}[\![\&_{\texttt{false}}]\!]m$	$=$	ff

system leads to the overall goal.

- If both the minimum and the maximum values of t are *true*, $M(t) = (tt, tt)$, the system is flawed. In other words, despite the attacker's actions (for all Boolean assignment m) an attack on the system is always successful.

The semantic evaluation explores all possible Boolean assignments to the leaves of the tree and establishes our reference standard for the evaluation. The drawback is its exponential complexity, as the satisfiability problem is NP-complete. In the following, we present a technique with lower complexity and we define restrictions on attack trees under which the new technique is sound with respect to the semantic evaluation.

The *algorithmic evaluation* $INT(t)$ of an attack tree $t \in Tree$ is defined in Table 3.3. The evaluation computes the pair of minimum and maximum success values by propagating the values of basic actions up to the root of the tree applying the rules presented in Table 3.3. The rule (Case-BA) assigns the minimum and maximum values to basic actions. The rules (Case-and) and (Case-or) define the computation for conjunction and disjunction in the standard manner, respectively. The rule (Case-neg) swaps the minimum and the maximum values before applying negation, as negation changes the goal of the attacker. The last two rules (Case-tt) and (Case-ff) assign true and false values to a trivially-successful and a trivially-failed action, respectively.

The two evaluation techniques we presented might lead to different results. For example, consider the attack tree $t = \&_\wedge(a, \&_\neg(a))$, where $a \in Act$. The result of the semantic evaluation is $M(t) = (ff, ff)$, while the result of the algorithmic evaluation is $INT(t) = (ff, tt)$. Observe that t is not polarity-consistent as a occurs both positively and negatively in t. In fact, to show that the two techniques are equivalent it is sufficient to restrict to polarity-consistent trees.

Table 3.3: The Boolean algorithmic evaluation of an attack tree.

$INT(a)$	$=$	(ff, tt)	(Case-BA)
$INT(\&_\wedge(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1,2\}$ $in\ (min_1 \wedge min_2, max_1 \wedge max_2)$	(Case-and)
$INT(\&_\vee(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1,2\}$ $in\ (min_1 \vee min_2, max_1 \vee max_2)$	(Case-or)
$INT(\&_\neg(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (\neg max, \neg min)$	(Case-neg)
$INT(\&_{\texttt{true}})$	$=$	(tt, tt)	(Case-tt)
$INT(\&_{\texttt{false}})$	$=$	$(\mathit{ff}, \mathit{ff})$	(Case-ff)

Theorem 3.2 *Let $t \in Tree$ be a polarity-consistent attack tree. Then*

$$M(t) = INT(t)$$

The complexity of the two evaluations is as follows. The semantic evaluation considers all possible Boolean attacks m, thus it is exponential in the number of leaves of t. The algorithmic evaluation consists in a bottom-up traversal over the tree, and thus it is linear in the size of t. For polarity-consistent trees the algorithmic evaluation offers a dramatic improvement in complexity.

3.1.3 Semantics in the Probabilistic Case

We extend the developments from the Boolean case to the probabilistic case. In this setting, we consider the interval $[0, 1]$, where 0 and 1 correspond to the failure and to the success of an attack, respectively. We investigate questions such as "what is the maximum probability of an attack?" or "how vulnerable is the system to an attack?".

We assume that each basic action $a \in Act$ has two associated success probabilities; success probability $p_1(a)$ in case of not performing a, and success probability $p_2(a)$ in case of performing a, such that $p_1(a) \leq p_2(a)$. For instance, an attacker might succeed to disable a security camera with a given probability p_2, or the security camera might be disabled due to some external conditions with a given probability p_1, which for the attacker will be the probability of succeeding without performing the action. We consider the Boolean assignment m as defined in the previous section, $m : Act \to \mathbb{B}$, and assume that an action a has a probability of success $p_1(a)$ if $m(a)$ is *false* and has a probability of success $p_2(a)$ if $m(a)$ is *true*. Choosing $p_1(a) = 0$ and $p_2(a) = 1$ coincides with the Boolean case.

Table 3.4: The probabilistic semantic evaluation of an attack tree.

$$M(t) = (\min\{\mathcal{P}[\![t]\!]m \mid m \text{ Boolean assignment}\}, \max\{\mathcal{P}[\![t]\!]m \mid m \text{ Boolean assignment}\})$$

$$\mathcal{P}[\![a]\!]m = \begin{cases} p_1(a) & \text{if} \quad m(a) = \mathit{ff} \\ p_2(a) & \text{if} \quad m(a) = \mathit{tt} \end{cases}$$

$$\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m = \mathcal{P}[\![t_1]\!]m \cdot \mathcal{P}[\![t_2]\!]m$$

$$\mathcal{P}[\![\&_\vee(t_1, t_2)]\!]m = 1 - (1 - \mathcal{P}[\![t_1]\!]m) \cdot (1 - \mathcal{P}[\![t_2]\!]m)$$

$$\mathcal{P}[\![\&_\neg(t)]\!]m = 1 - \mathcal{P}[\![t]\!]m$$

$$\mathcal{P}[\![\&_{\mathtt{true}}]\!]m = 1$$

$$\mathcal{P}[\![\&_{\mathtt{false}}]\!]m = 0$$

In the probabilistic setting the operators are interpreted differently than in the Boolean setting. They satisfy commutativity (e.g., $\&_\wedge(a, b)$ and $\&_\wedge(b, a)$ are evaluated identically by the same assignment) and associativity (e.g., $\&_\wedge(a, \&_\wedge(b, c))$ and $\&_\wedge(\&_\wedge(a, b), c)$ are evaluated identically by the same assignment). However, distributivity and idempotency do not hold in this setting. Thus, the following two trees, $t_1 = \&_\wedge(a, \&_\vee(b, c))$ and $t_2 = \&_\vee(\&_\wedge(a, b), \&_\wedge(a, c))$, are not equivalent in the probabilistic setting. The core reason why, is that in the probabilistic setting multiple occurrences of the same action a are governed by the corresponding probability distribution, but the outcome of each occurrence is given by an independent sampling because each occurrence is attempted independently. This feature allows to abstract details away in the model and group basic actions in families sharing the same probabilistic behaviour. For instance, the basic action "break door" might be used for all doors in the model that share similar characteristics.

Similarly to the Boolean setting, we first present an evaluation technique that describes the analysis in a natural way, and then we formulate a technique with lower complexity by determining the necessary restrictions on the model.

The *algorithmic evaluation* $M(t)$ for an attack tree $t \in \mathit{Tree}$ is presented in Table 3.4. The evaluation considers all possible Boolean assignments and from these infers the probability values. The analysis $\mathcal{P}[\![t]\!]m$ of the tree t, displayed in the second part of Table 3.4, is performed recursively. The evaluation computes the pair of minimum and maximum success probabilities. The maximum success probability shows the existence of an attack with that probability. We say that the system is p-vulnerable if there exists an attack with success probability p.

The *algorithmic evaluation* $\mathit{INT}(t)$ of an attack tree $t \in \mathit{Tree}$ is given in Table 3.5. Similarly to the semantic evaluation, it computes the minimum and the maximum success probabilities. The evaluation associates the probability

Table 3.5: The probabilistic algorithmic evaluation of an attack-defence tree.

$INT(a)$	$=$	$(p_1(a), p_2(a))$	(Case-BA)
$INT(\&_\wedge(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1,2\}$ $in\ (min_1 \cdot min_2, max_1 \cdot max_2)$	(Case-and)
$INT(\&_\vee(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1,2\}$ $in\ (1-(1-min_1)\cdot(1-min_2),$ $1-(1-max_1)\cdot(1-max_2))$	(Case-or)
$INT(\&_\neg(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (1-max, 1-min)$	(Case-neg)
$INT(\&_{\texttt{true}})$	$=$	$(1,1)$	(Case-tt)
$INT(\&_{\texttt{false}})$	$=$	$(0,0)$	(Case-ff)

values with each basic action and traverses the tree using a bottom-up approach. For the negation operator, similarly to the Boolean case, the values are swapped before applying negation.

Analogously to the previous section, we shall restrict to polarity-consistent trees in order to show the equivalence of the two evaluations.

Theorem 3.3 *Let $t \in Tree$ be a polarity-consistent attack tree. Then*

$$M(t) = INT(t)$$

Here as well, the imposed polarity-consistent restriction allows to scale from an exponential to a linear complexity.

Example. Consider the polarity-consistent attack tree t presented in Figure 3.1. We apply the algorithmic evaluation for computing the maximum probability of the attack scenario. The possible probability values for basic actions are listed in Table 3.6 (the last column is for later reference).

The result of the algorithmic computation is $INT(t) = (0, 0.34)$, that is, the system is 0.34 vulnerable. The details of the computation with the values for the internal nodes are presented in Appendix B.1.

3.2 Attack Trees with Cost

In the previous section we presented the feasibility and probability evaluations of an attack tree. In this section, we extend the model of attack trees with a

Table 3.6: The values of probability and cost for the basic actions of the tree t, displayed in Figure 3.1

Label	Name of the nodes	p_1	p_2	c
ip	infiltrate premises	0.1	0.25	120
rc	jam remote-card communications	0	0.65	100
rt	jam remote-TV communications	0	0.65	100
it	impersonate technician	0	0.6	60
itt	impersonate TV technician	0	0.45	60
itp	impersonate trusted person	0	0.3	80
tc	threaten cardholder	0	0.3	30
ci	collect information	0.1	0.55	50
bc	blackmail cardholder	0	0.2	30

single cost for basic actions, and investigate cost-related questions. We start by presenting evaluation of the minimum cost in the Boolean setting in Sect. 3.2.1. Sect. 3.2.2 generalises the developments to the probabilistic setting.

3.2.1 Cost in the Boolean Case

In the following, the semantic and algorithmic evaluations are extended with a single cost. Hence, we associate with each basic action a pair of Boolean and cost values and consider the set $D = \mathbb{B} \times \mathbb{Q}_{\geq 0}$. We investigate questions such as "what is the minimum cost of an attack?" or "how much does it cost to protect a system in a given scenario?".

We assume that each basic action $a \in Act$ has a cost of not performing a, which we set to 0, and a cost $c(a) \in \mathbb{Q}_{\geq 0}$ of performing a. The actions are associated with two costs in order to link the model to the Boolean setting in a seamless manner.

Associating a pair of success and cost values with each basic action leads to multi-parameter optimisation. Moreover, having the goal of the attacker to maximise the success value and minimise the cost, rises an issue of optimising more than one conflicting parameters at the same time. In order to handle multi-parameter optimisation with incomparable values we employ to the notion of Pareto efficiency. We define two functions for computing the set of Pareto-efficient solutions.

We assume that the goal of the attacker is to maximise the success value while minimising the cost of an attack. In order to compute the set of pairs of efficient solutions, where we want to maximise the first argument while minimising the second, we define function MR^{+-}. The function computes the set of all pairs that have higher value for the first argument *or* lower value for the second argument with respect to the other pairs in the set.

$$\begin{aligned} MR^{+-}(Z) &= \{(x,y) \in Z \mid \forall (x',y') \in Z : x' \sqsupseteq x \wedge y' \sqsubseteq y \Rightarrow x' = x \wedge y' = y\} \\ &= \{(x,y) \in Z \mid \forall (x',y') \in Z : (x \sqsupseteq x' \vee y \sqsubset y') \ \wedge \ (x \sqsupset x' \vee y \sqsubseteq y')\} \end{aligned}$$

where $Z \subseteq D$.

Table 3.7: The Boolean semantic evaluation of an attack tree with cost.

$$M(t,A) = (MR^{--}(\{\ (\mathcal{B}[\![t]\!]m, \mathfrak{b}) \mid cost(m,A) \leq \mathfrak{b}\}),\ MR^{+-}(\{\ (\mathcal{B}[\![t]\!]m, \mathfrak{b}) \mid cost(m,A) \leq \mathfrak{b}\}))$$

$$cost(m,A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m(a) = ff \\ c(a), & \text{if } m(a) = tt \end{cases}$$

Note that the sign "+" indicates the maximisation and the sign "−" indicates the minimisation, and their position refer to the parameter of the maximisation/minimisation.

As we mentioned in Sect. 3.1.1, the negation operator changes the goal of the player. For instance, if the goal is to maximise the success value and minimise the cost, then under negation the goal is to minimise the success value and minimise the cost. Observe that the objective for the cost remains the same under negation, as we assume to deal with rational players. Thus, we define a dual function MR^{--} that computes the set of all pairs that have lower value for both arguments with respect to the other pairs in the set.

$$\begin{aligned} MR^{--}(Z) &= \{(x,y) \in Z \mid \forall (x',y') \in Z : x' \sqsubseteq x \wedge y' \sqsubseteq y \Rightarrow x' = x \wedge y' = y\} \\ &= \{(x,y) \in Z \mid \forall (x',y') \in Z : (x \sqsubseteq x' \vee y \sqsubset y')\ \wedge\ (x \sqsubset x' \vee y \sqsubseteq y')\} \end{aligned}$$

where $Z \subseteq D$.

Note, that the first sign "−" in MR^{--} corresponds to the min operator in Table 3.2, and the sign "+" in MR^{+-} corresponds to the max operator.

Similarly to the previous cases, we present two evaluation techniques and investigate their equivalence condition.

The *semantic evaluation* $M(t,A)$ of an attack tree $t \in \mathit{Tree}$ and a set of basic actions A is defined in Table 3.7. It computes a pair of sets of Pareto-efficient solutions MR^{--} and MR^{+-}. The first set corresponds to the solutions that have lower success values and lower cost if compared to other solutions, while the second set corresponds to the solutions that have higher success value and lower cost if compared to other solutions.

The analysis $\mathcal{B}[\![t]\!]m$ computes the success value for each Boolean assignment m. The cost is represented with the concept of a budget for associating with each success value the corresponding budget of the attacker and the Boolean assignment m. The budget $\mathfrak{b} \in \mathbb{Q}_{\geq 0}$ takes values from 0 to infinity in an increasing manner. For a given budget $\mathfrak{b}$ we take m such that the cost of the attacker for m is not greater than $\mathfrak{b}$, and the corresponding success value for m is computed. The cost for a given m is computed with the function *cost*, defined in Table 3.7.

The *algorithmic evaluation* $INT(t)$ for an attack tree $t \in \mathit{Tree}$ is given in Table 3.8. Similarly, it computes a pair of sets MR^{--} and MR^{+-} by travers-

Table 3.8: The Boolean algorithmic evaluation of an attack tree with cost.

$$
\begin{array}{lcll}
INT(a) & = & (MR^{--}(\{(\mathit{ff},0),(tt,c(a))\}), & \\
 & & \ MR^{+-}(\{(\mathit{ff},0),(tt,c(a))\})) & \text{(Case-BA)} \\
 & & & \\
INT(\&_{\wedge}(t_1,t_2)) & = & \mathit{let}\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & \mathit{in}\ (MR^{--}(\{(b \wedge b', c+c') \mid (b,c) \in V_1, (b',c') \in V_2\}), & \\
 & & \quad MR^{+-}(\{(b \wedge b', c+c') \mid (b,c) \in W_1, (b',c') \in W_2\})) & \\
 & & & \text{(Case-and)} \\
 & & & \\
INT(\&_{\vee}(t_1,t_2)) & = & \mathit{let}\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & \mathit{in}\ (MR^{--}(\{(b \vee b', c+c') \mid (b,c) \in V_1, (b',c') \in V_2\}), & \\
 & & \quad MR^{+-}(\{(b \vee b', c+c') \mid (b,c) \in W_1, (b',c') \in W_2\})) & \\
 & & & \text{(Case-or)} \\
 & & & \\
INT(\&_{\neg}(t)) & = & \mathit{let}\ (V,W) = INT(t) & \\
 & & \mathit{in}\ (MR^{--}(\{(\neg b, c) \mid (b,c) \in W\}), & \\
 & & \quad MR^{+-}(\{(\neg b, c) \mid (b,c) \in V\})) & \text{(Case-neg)} \\
 & & & \\
INT(\&_{\texttt{true}}) & = & (\{(tt,0)\},\{(tt,0)\}) & \text{(Case-tt)} \\
 & & & \\
INT(\&_{\texttt{false}}) & = & (\{(\mathit{ff},0)\},\{(\mathit{ff},0)\}) & \text{(Case-ff)}
\end{array}
$$

ing the tree from the leaves to the root following the rules given in Table 3.8. The rule (Case-BA) defines the sets MR^{--} and MR^{+-} for the basic actions. The rules (Case-and) and (Case-or) represent the evaluation of conjunction and disjunction, respectively, where the success values are computed through the Boolean evaluation of the corresponding operator and the costs are summed. The rule (Case-neg) for the negation operator swaps the pairs before applying the evaluation. The evaluation of the success value simply negates it, as in Sect. 3.1.2, while cost remains unchanged. The rules (Case-tt) and (Case-ff) are used to evaluate trivially-successful and trivially-failed actions, where the cost is equal to 0 as these actions are independent from the attacker. The functions MR^{--} and MR^{+-} are applied to each rule leading to a reduction of the size of the sets in each step.

We denote by $\mathit{yield}(t) \subseteq \mathit{Act}$ the set of actions that correspond to the leaves of t. Consider the polarity-consistent tree $t = \&_{\wedge}(\&_{\vee}(a,b),\&_{\vee}(a,d))$, where $c(a) = 2$, $c(b) = 1$ and $c(d) = 3$. The result of the semantic evaluation is $M(t,A) = (\{(\mathit{ff},0)\},\{(\mathit{ff},0),(tt,2)\})$, while the result of the algorithmic evaluation is $INT(t) = (\{(\mathit{ff},0)\},\{(\mathit{ff},0),(tt,3)\})$. As it becomes apparent from

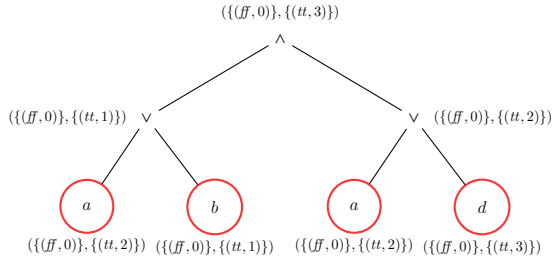


Figure 3.2: The Boolean algorithmic evaluation with cost of the attack tree $t = \&_\wedge(\&_\vee(a, b), \&_\vee(a, d))$.

Figure. 3.2, the algorithmic evaluation chooses b to satisfy the sub-tree $\&_\vee(a, b)$ as $c(b) < c(a)$ and a in the sub-tree $\&_\vee(a, d)$ as $c(a) < c(d)$. However, the choice of b and a are local to their sub-trees and yield sub-optimal solution once combined. The reason why is to be sought in the repetition of a in distinct sub-trees. On the contrary, in the semantic evaluation the assignment $m(a) = tt$ is enough to satisfy the whole tree yielding cost 2. Hence, the polarity-consistent restriction in no longer adequate for the model of attack trees with cost. To show the equivalence of the two evaluations we define the following restriction, inspired by Girard's linear logic [Gir95].

Definition 3.4 An attack tree t is linear iff no action occurs twice in t.

The notion of linearity is stronger than polarity-consistency, as the latter does not forbid to have multiple occurrences of the same action with the same polarity.

Theorem 3.5 *Let $t \in Tree$ be a linear attack tree. Then*

$$M(t, yield(t)) = INT(t)$$

The complexity of the two evaluation techniques is defined with respect to the number of set operations. The semantic evaluation is exponential, while the algorithmic evaluation is linear. Thus, the linearity restriction leads to a dramatic improvement in the complexity.

3.2.2 Cost in the Probabilistic Case

In this section we study the model of attack trees with a single cost in the probabilistic setting by generalising the Boolean case. We focus on the differences and omit redundant details.

We consider set $D = [0, 1] \times \mathbb{Q}_{\geq 0}$ and investigate questions such as "what is the maximum probability of an attack while minimising the cost?". In order to

Table 3.9: The probabilistic semantic evaluation of an attack tree with cost.

$$\begin{array}{rcl} M(t,A) & = & (MR^{--}(\{\ (\mathcal{P}[\![t]\!]m, \mathfrak{b}) \mid cost(m,A) \leq \mathfrak{b}\}), \\ & & MR^{+-}(\{\ (\mathcal{P}[\![t]\!]m, \mathfrak{b}) \mid cost(m,A) \leq \mathfrak{b}\})) \end{array}$$

$$cost(m,A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m(a) = \mathit{ff} \\ c(a), & \text{if } m(a) = \mathit{tt} \end{cases}$$

link to the model to the probabilistic setting, we assume that each basic action $a \in Act$ has a cost of not performing a, which we set to 0 in this work, and a cost $c(a) \in \mathbb{Q}_{\geq 0}$ of performing a.

Similarly to the Boolean case, the consideration of probability and cost and the goal of maximising probability while minimising cost lead to a multi-parameter optimisation problem with incomparable values. We define two evaluation techniques by considering the functions MR^{--} and MR^{+-}.

The *semantic evaluation* $M(t, A)$ of an attack tree $t \in \mathit{Tree}$ and a set of basic actions A is given in Table 3.9. It computes the pair of sets of Pareto-efficient solutions. The evaluation follows the corresponding one presented in Sect. 3.2.1, with the difference that the computation of the success value is based on the probabilistic analysis $\mathcal{P}[\![t]\!]m$ instead of the Boolean analysis $\mathcal{B}[\![t]\!]m$.

The *algorithmic evaluation* $INT(t)$ of an attack tree $t \in \mathit{Tree}$ is presented in Table 3.10. The computation is performed in a bottom-up manner according to the similar rules given in the Boolean setting, where the computation of the probabilistic values are performed according to probabilistic evaluation, discussed in Sect. 3.1.3.

In order to show the equivalence of the two evaluation techniques, we resort again to a linearity restriction on trees.

Theorem 3.6 *Let $t \in \mathit{Tree}$ be a linear attack tree. Then*

$$M(t, \mathit{yield}(t)) = INT(t)$$

Here again the restriction allows to scale from an exponential to a linear complexity.

Example. Consider the linear attack tree t presented in Figure 3.1, and the values of probability and cost given in Table 3.6.

We apply the above mentioned algorithmic evaluation for computing attacks with maximum probability of success and minimum cost. We get the following result at the root of the tree:

Table 3.10: The probabilistic algorithmic evaluation of an attack tree with cost.

$$
\begin{aligned}
INT(a) &= (MR^{--}(\{(p_1(a),0),(p_2(a),c(a))\}),\\
&\quad MR^{+-}(\{(p_1(a),0),(p_2(a),c(a))\})) && \text{(Case-BA)}\\
INT(\&_\wedge(t_1,t_2)) &= let\ (V_i,W_i) = INT(t_i), i \in \{1,2\}\\
&\quad in\ (MR^{--}(\{(p\cdot p', c+c') \mid (p,c)\in V_1, (p',c')\in V_2\}),\\
&\quad\quad MR^{+-}(\{(p\cdot p', c+c') \mid (p,c)\in W_1, (p',c')\in W_2\})) && \text{(Case-and)}\\
INT(\&_\vee(t_1,t_2)) &= let\ (V_i,W_i) = INT(t_i), i \in \{1,2\}\\
&\quad in\ (MR^{--}(\{(1-(1-p)(1-p'), c+c') \mid\\
&\quad\quad (p,c)\in V_1, (p',c')\in V_2\}),\\
&\quad\quad MR^{+-}(\{(1-(1-p)(1-p'), c+c') \mid\\
&\quad\quad (p,c)\in W_1, (p',c')\in W_2\})) && \text{(Case-or)}\\
INT(\&_\neg(t)) &= let\ (V,W) = INT(t)\\
&\quad in\ (MR^{--}(\{(1-p,c) \mid (p,c)\in W\}),\\
&\quad\quad MR^{+-}(\{(1-p,c) \mid (p,c)\in V\})) && \text{(Case-neg)}\\
INT(\&_{\texttt{true}}) &= (\{(1,0)\},\{(1,0)\}) && \text{(Case-tt)}\\
INT(\&_{\texttt{false}}) &= (\{(0,0)\},\{(0,0)\}) && \text{(Case-ff)}
\end{aligned}
$$

$$
\begin{aligned}
INT(t) = (&\{(0,0)\},\{(0,0),(0.03,30),(0.05,60),(0.11,80),(0.15,90),\\
&(0.17,120),(0.24,170),(0.28,250),(0.3,330),\\
&(0.31,410),(0.33,510),(0.34,630)\})
\end{aligned}
$$

Figure 3.3 represents the Pareto frontier corresponding to the overall result of the evaluation. In the figure each point describes a possible optimal solution with a probability of success with the corresponding cost. The probability of success of an attack ranges from 0 to 0.34, while the corresponding cost, representing the necessary resources to be spent, ranges from 0 to 630. The detailed evaluation on the tree t is given in Appendix B.2.

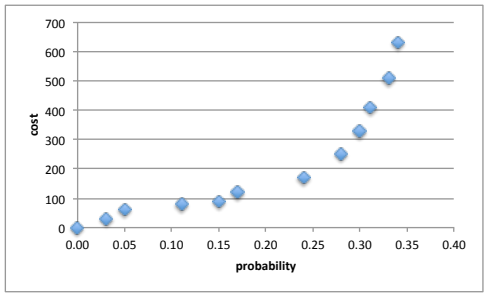


Figure 3.3: Pareto-efficient solutions for the attack tree t.

3.3 Attack Trees with Multiple Costs

We extend further the model of attack trees by considering multiple costs for basic actions. In the following, we study the extended model in the probability setting as it generalises the Boolean one.

Building on top of the developments of Sect. 3.2.2 we consider the set $D = [0,1] \times \mathbb{Q}^n_{\geq 0}$. We assume that each basic action $a \in Act$ has two vectors of n associated costs: a vector of 0's denoting the costs incurred for not performing a, and a vector $\gamma : Act \to \mathbb{Q}^n_{\geq 0}$ denoting the costs incurred for performing a. We write $\gamma = (c_1(a), \cdots, c_n(a))$, $c_i : Act \to \mathbb{Q}_{\geq 0}$, $i \in [1,n]$. When adding costs we resort to point-wise summation of vectors.

Similarly to the model with a single cost, the investigation of probability and multiple costs leads to a multi-parameter optimisation challenge with conflicting parameters. In order to generalise the functions MR^{--} and MR^{+-} to deal with multiple costs we introduce polarity modifications of the comparison operators as follows: $\sqsupseteq^+$ is $\sqsupseteq$, $\sqsupset^+$ is $\sqsupset$, $\sqsupseteq^-$ is $\sqsubseteq$ and $\sqsupset^-$ is $\sqsubset$. The sign "+" corresponds to the maximisation of the parameters and keeps the operator as it is, while the sign "−" corresponds to the minimisation of the parameters, therefore it changes the operator.

The general format of the optimisation function MR is defined as follows, where $s_i \in \{+,-\}$ and $Z \subseteq D$:

$$\begin{aligned} MR^{s_0,\cdots,s_n}(Z) &= \{(x_0,\cdots,x_n) \in Z \mid \forall (x'_0,\cdots,x'_n) \in Z : \\ &\quad x'_0 \sqsupseteq^{s_0} x_0 \wedge \cdots \wedge x'_n \sqsupseteq^{s_n} x_n \Rightarrow x'_0 = x_0 \wedge \cdots \wedge x'_n = x_n\} \\ &= \{(x_0,\cdots,x_n) \in Z \mid \forall (x'_0,\cdots,x'_n) \in Z : \\ &\quad ((x_0 \sqsupseteq^{s_0} x'_0) \vee (x_1 \sqsupset^{s_1} x'_1) \vee \cdots \vee (x_n \sqsupset^{s_n} x'_n)) \wedge \cdots \\ &\quad \cdots \wedge ((x_0 \sqsupset^{s_0} x'_0) \vee (x_1 \sqsupset^{s_1} x'_1) \vee \cdots \vee (x_n \sqsupseteq^{s_n} x'_n))\} \end{aligned}$$

The function $MR^{s_0,\cdots,s_n}$ computes the efficient solutions for multiple parameters by maximising the parameter values if $s_i = +$ and minimising them if $s_i = -$. Note that each $\sqsupseteq^{s_i}$ is in fact a total order (on [0,1] or $\mathbb{Q}_{\geq 0}$) and hence

Table 3.11: The probabilistic semantic evaluation of an attack tree with multiple cost.

$$
\begin{array}{rcl}
M^*(t,A) & = & (MR^{--\cdots-}(\{\ (\mathcal{P}[\![t]\!]m, \mathfrak{b}^1, \cdots \mathfrak{b}^n) \mid cost_i(m,A) \leq \mathfrak{b}^i\}), \\
 & & MR^{+-\cdots-}(\{\ (\mathcal{P}[\![t]\!]m, \mathfrak{b}^1 \cdots \mathfrak{b}^n) \mid cost_i(m,A) \leq \mathfrak{b}^i\}))
\end{array}
$$

$$
cost_i(m,A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m(a) = \mathit{ff} \\ c_i(a), & \text{if } m(a) = \mathit{tt} \end{cases}
$$

$\neg(x_i' \sqsupseteq^{s_i} x_i)$ is equivalent to $x_i \sqsupset^{s_i} x_i'$. Observe that with this notation we get MR^{--} if we take $n = 1$ and $s_0 = s_1 = -$, and we get MR^{+-} if we take $n - 1$ and $s_0 = +, s_1 = -$.

Lest to surprise the reader, we present again two evaluation techniques.

The *semantic evaluation* $M^*(t, A)$ of an attack tree $t \in \mathit{Tree}$ and a set of basic actions A is given in Table 3.11. The evaluation closely follows the corresponding one described in Sect. 3.2.2. Similarly, the cost is represented with a budget $\mathfrak{b}^i \in \mathbb{R}_{\geq 0}$, where for a given budget $\mathfrak{b}^i$ the corresponding cost $c_i(a)$ from the vector γ is considered. The result of the evaluation is a pair of the set of Pareto-efficient solutions, computed through the functions $MR^{--\cdots-}$ and $MR^{+-\cdots-}$.

The *algorithmic evaluation* $INT^*(t)$ of an attack tree $t \in \mathit{Tree}$ is presented in Table 3.12. Similarly to the evaluation of the model with a single cost, it traverses the tree from the leaves to the root according to the rules. The rules extend the ones presented in Table 3.10 with multiple costs.

The rule (Case-BA) defines the Pareto sets for the basic actions. The rules (Case-and) and (Case-or) compute all possible sets from two sub-trees by evaluating probabilities based on the operator and summing the corresponding costs, and then applying the function $MR^{\cdots}$ in order to obtain the optimal solutions. This is sound due to the point-wise ordering of arguments of $MR^{\cdots}$. Similarly to the previous cases, the negation rule (Case-neg) swaps the sets before applying the negation. The rules (Case-tt) and (Case-ff) are for trivially-successful and trivially-failed actions, and have all costs equal to 0.

Analogously to the previous section, the linearity restriction is sufficient in order to show the equivalence of the two evaluations.

Theorem 3.7 *Let $t \in \mathit{Tree}$ be a linear attack tree. Then*

$$M^*(t, \mathit{yield}(t)) = INT^*(t)$$

Table 3.12: The probabilistic algorithmic evaluation of an attack tree with multiple cost.

$$
\begin{array}{lcll}
INT^*(a) & = & (MR^{--\cdots-}(\{(p_1(a),0,\cdots,0),(p_2(a),c_1(a),\cdots,c_n(a))\}), & \\
 & & \quad MR^{+-\cdots-}(\{(p_1(a),0,\cdots,0),(p_2(a),c_1(a),\cdots,c_n(a))\})) & \\
 & & & \text{(Case-BA)} \\
INT^*(\&_\wedge(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & in\ (MR^{--\cdots-}(\{(p\cdot p', c_1+c_1', \cdots, c_n+c_n') \mid & \\
 & & \qquad (p,c_1,\cdots,c_n) \in V_1, (p',c_1',\cdots,c_n') \in V_2\}), & \\
 & & \quad MR^{+-\cdots-}(\{(p\cdot p', c_1+c_1', \cdots, c_n+c_n') \mid & \\
 & & \qquad (p,c_1,\cdots,c_n) \in W_1, (p',c_1',\cdots,c_n') \in W_2\})) & \\
 & & & \text{(Case-and)} \\
INT^*(\&_\vee(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & in\ (MR^{--\cdots-}(\{(1-(1-p)(1-p'), c_1+c_1', \cdots, c_n+c_n') \mid & \\
 & & \qquad (p,c_1,\cdots,c_n) \in V_1, (p',c_1',\cdots,c_n') \in V_2\}), & \\
 & & \quad MR^{+-\cdots-}(\{(1-(1-p)(1-p'), c_1+c_1', \cdots, c_n+c_n') \mid & \\
 & & \qquad (p,c_1,\cdots,c_n) \in W_1, (p',c_1',\cdots,c_n') \in W_2\})) & \\
 & & & \text{(Case-or)} \\
INT^*(\&_\neg(t)) & = & let\ (V,W) = INT(t) & \\
 & & in\ (MR^{--\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in W\}), & \\
 & & \quad MR^{+-\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in V\})) & \\
 & & & \text{(Case-neg)} \\
INT^*(\&_{\texttt{true}}) & = & (\{(1,0,\cdots,0)\},\{(1,0,\cdots,0)\}) & \text{(Case-tt)} \\
INT^*(\&_{\texttt{false}}) & = & (\{(0,0,\cdots,0)\},\{(0,0,\cdots,0)\}) & \text{(Case-ff)}
\end{array}
$$

3.4 Attack Tree Evaluator Tool

We applied our approach to a real-life scenario of cloud environment studied in the project TREsPASS [The14]. The International Traders LTD (ITL) company is a commodity trader that buys and sells oil and gas related investments around the world for an international set of clients. The key company asset is a list of international clients with their private data. Much of their trading is automated and they have a large IT department.

The company owns an office with a co-located data-center, illustrated in Figure 3.4. Both the office and the data-center can be accessed either through the door or the window. The data-center can also be accessed from the office through an internal door.

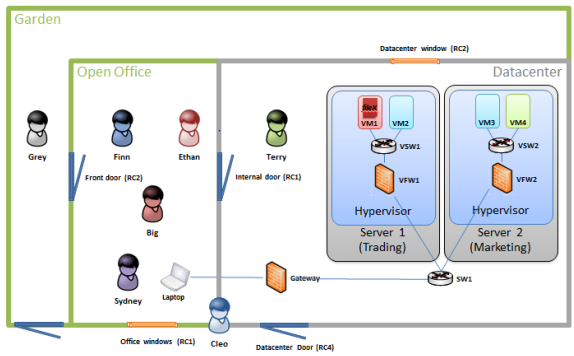


Figure 3.4: Simplified scenario of a private cloud environment with various actors.

The cloud infrastructure is running on two physical servers, both located in the data-center. On each server, two virtual machines, a virtual switch and a virtual firewall are running on top of a Hypervisor. These virtual components are connected to physical network components, namely switch SW1 and a Gateway. Through this connection it is possible to reach the physical and virtual infrastructure from the Laptop. The sensitive document FileX is located in the storage of the virtual machine VM1.

We consider an attacker whose goal is to obtain the sensitive document FileX. The attacker is Ethan, who is trying to obtain the FileX through Terry. Due to the large size of the attack tree corresponding to the scenario, we do not represent it here. However, we illustrate a small part of it, presented in Figure 3.5. For obtaining the FileX from Terry attacker has to get Terry's ID card and the pin. The attacker gets the ID card though threatening, blackmailing, bribing, social engineering or acquiring trust. Similarly the attacker can obtain the pin of the ID card.

We apply the algorithmic evaluation to the example considering probability and cost values to basic actions. The result of the evaluation at the root is presented in Figure 3.6. As we can see from the chart, the probability of an attack ranges from 0 to 0.73 and the cost ranges from 0 to 9250. For each solution, represented in the frontier, we can say that the system is (p, c)-vulnerable.

The result can be further analysed. For example, given a cost threshold for an attacker we can eliminate the attacks that are outside of the cost threshold, or we can rank the most successful attacks based on the threshold.

Implementation. A proof-of-concept implementation of the algorithmic evaluation techniques has been developed in Java and is available at

`http://www2.compute.dtu.dk/~zaas/ATE.zip`

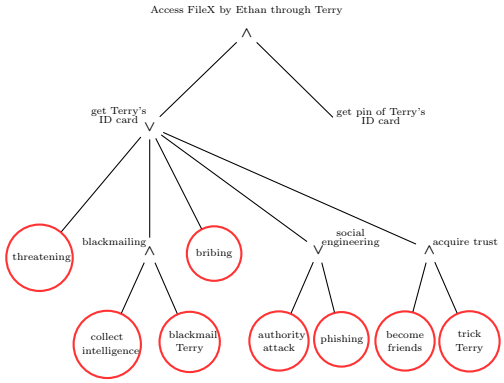


Figure 3.5: Attack tree for obtaining FileX.

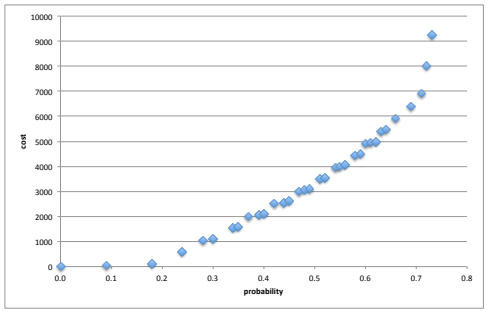


Figure 3.6: Pareto-efficient solutions for the attack tree t.

The *Attack Tree Evaluator* addresses multi-parameter optimisation of attack trees in terms of Pareto efficiency. It takes as input an attack tree in .xml format and values such as probability of success and cost for the basic actions.

If only the probabilistic values are provided for the basic actions, the tool computes the maximum success probability. It also provides the set of basic actions for that attack. If instead probability and cost values are provided, the tool computes the set of optimal solutions by means of Pareto efficiency. It presents the Pareto sets of pairs of maximum probability and minimum cost, with the corresponding sets of basic actions. The overall structure of the tool is presented in Figure 3.7.

3.5 Concluding Remarks

Attack trees are a widely-used tool to analyse threat scenarios and describe in a formal way the security of a system. Most analyses consider one-parameter

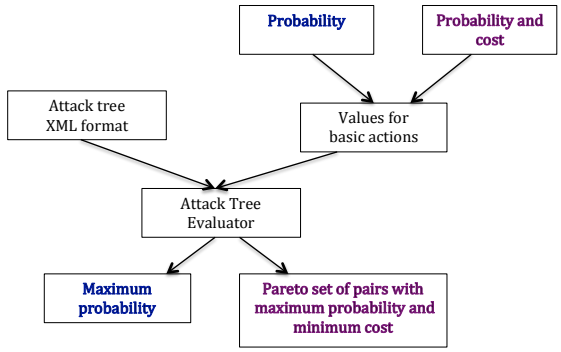


Figure 3.7: Structure of the tool Attack Tree Evaluator.

attack trees and analyse a particular aspect of an attack, such as feasibility or cost. Moreover, in attack trees with multiple parameters, values characterising basic actions are propagated to the root relying on local decision strategies. In case of conflicting objectives, however, this approach may yield sup-optimal results.

To overcome this limitation, we devised automated techniques that quantify attack scenarios with sets of Pareto optimal solutions in the case of multiple conflicting objectives. We studied the problem in both the Boolean and the probabilistic settings. In each setting we presented two evaluation techniques, a top-down evaluation with exponential complexity and a bottom-up evaluation with linear complexity. We showed the equivalence of two evaluation techniques under some condition. Similarly, we extended the developments to encompass single and multiple costs to basic actions, and presented the corresponding evaluation techniques.

The method for evaluating attack trees with multiple costs can seamlessly encompass costs but also rewards. Contrary to costs, a rational attacker would seek reward maximisation, which can be evaluated in our framework by changing the symbols s_i in the function $MR^{s_0,\cdots,s_n}$ accordingly. Moreover, the model can also cope with symbolic costs/rewards or with a combination of these.

We illustrated our evaluation techniques on an example from the TREsPASS project, where the analysis allows to identify the most promising attacks.

In order to investigate the security properties of an attack scenario described as an attack tree, the attack tree model needs first be produced. The automated generation of attack trees is one active research strand, briefly surveyed in Ch. 2. Some of the examples we presented have been generated automatically by tools developed in the context of the TREsPASS project, a European research project for technology-supported risk estimation by predictive assessment of socio-technical security. At a very high level, the end to end process

starts with the generation of an attack tree for the scenario under study, whose security properties are then analysed with dedicated tools and finally presented to the end user.

Once an attack tree is available, another key component in the evaluation of the corresponding attack scenario consists in the association of basic actions with parameters such as probability and cost, which allow to phrase questions beyond Boolean security. Providing realistic estimates for the parameters that describe basic actions is a research topic in itself. Several techniques have been applied to this problem, including quantification of information leakage [BLMW13] and resorting to domain expert knowledge.

Chapter 4

Pareto Efficient Solutions of Attack-Defence Trees

In the previous chapter we have developed evaluation techniques of attack scenarios represented by attack trees. However, attack trees allow to model and analyse only the attacker's behaviour and do not consider possible defences undertaken to avoid the attacks. In this chapter we extend the model of an attack tree with the defender's action.

As we have discussed in Ch. 2, attack-defence trees are extensions of attack trees with countermeasures. They illustrate in a graphical way the possible actions an attacker can perform in order to attain a given goal, and the feasible countermeasures a defender can undertake to counter such actions. Attack-defence trees are used for analysing attack-defence scenarios. As for attack trees, typical evaluations assign values to the parameters of the leaves and traverse a tree from the leaves to the root.

The remarks on existing analyses we made on attack trees in Ch. 3 also apply to attack-defence trees. Most analyses focus on one specific aspect of the system, such as feasibility or cost of an attack or a defence. They do not consider multiple parameters and the subsequent need for optimising all of them at once. As we have seen, multiple parameters may lead to Pareto problems, where there is a set of optimal solutions.

In order to address multi-parameter optimisation of attack-defence trees, we present evaluation techniques that characterise the leaves of a tree with more than one parameter, such as the success probability *and* the cost of an attack. Our techniques compute different aspects of the scenario and handle multiple parameters, thus optimising all of them at once. Multi-parameter optimisation

becomes necessary in case of conflicting objectives, that we handle by computing the set of optimal solutions, defined in terms of Pareto efficiency.

Building on Ch 3, our developments are performed on a new language-based formalism for attack-defence trees, and retrace the same structure of our investigation of attack trees. We study attack-defence tree analysis in Boolean and probabilistic settings. For each setting, we first consider the problem of feasibility of the attack or the defence, and then we extend our techniques to compute optimal attacks or defences in presence of probabilities and multiple costs. Moreover, for each case, we first define the solution considering all possible player interactions, obtaining a natural but exponential characterisation. Then, we improve on the complexity devising an algorithmic evaluation that is linear and yet sound for an expressive sub-class of models.

The chapter is organised as follows. In Sect. 4.1 we introduce our formalism for attack-defence trees and provide evaluation techniques for feasibility queries. Sect. 4.2 extends the model with a single cost and presents evaluation techniques for computing minimum cost. We extend the single cost model to multiple costs in Sect. 4.3. The evaluation is demonstrated on a case study for a Radio-Frequency Identification system managing goods in a warehouse. Proofs are in Appendix C. This chapter is mainly based on [AN15].

4.1 Formal Model of Attack-Defence Trees

In the following, we extend the formalism for attack trees given in Sect. 3.1 to attack-defence trees. First, we define the syntax and the related terminology in Sect. 4.1.1. Then, we describe the evaluation techniques for investigating the feasibility of attacks and defences both in Boolean and probabilistic settings. The Boolean case is thoroughly explained in Sect. 4.1.2. The developments are generalised to the probabilistic setting in Sect. 4.1.3.

4.1.1 Attack-Defence Trees

We construe an attack-defence tree as an interaction between two players (denoted by τ), the proponent ($\tau = p$) and the opponent ($\tau = o$), in the wake of [KMRS10]. A player can be either an attacker or a defender. We associate the proponent with the player at the root, and the opponent with the opposite player. Each player has an associated goal, such as minimising or maximising the overall probability of an attack or a defence.

The root of the tree represents the main goal of an attack-defence scenario for a given player τ. The leaves represent the basic actions that a player can perform to achieve his/her goal. The internal nodes show how those actions can be combined. In order to simplify the technical developments, we assume that the players' actions are independent.

Table 4.1: The syntax of attack-defence trees and the type system for defining well-formed trees.

$$t \quad ::= \quad a \;\mid\; \&_{\wedge}(t_1, t_2) \;\mid\; \&_{\vee}(t_1, t_2) \;\mid\; \&_{\neg}(t) \;\mid\; \&_{\sim}(t) \;\mid\; \&_{\texttt{true}} \;\mid\; \&_{\texttt{false}}$$

$$\vdash a : p \;\text{ if } a \in Act_p \qquad\qquad \vdash a : o \;\text{ if } a \in Act_o$$

$$\frac{\vdash t_1 : \tau \;\; \vdash t_2 : \tau}{\vdash \&_{\wedge}(t_1, t_2) : \tau} \qquad \frac{\vdash t_1 : \tau \;\; \vdash t_2 : \tau}{\vdash \&_{\vee}(t_1, t_2) : \tau} \qquad \frac{\vdash t : \tau}{\vdash \&_{\neg}(t) : \tau} \qquad \frac{\vdash t : \tau}{\vdash \&_{\sim}(t) : \tau'} \; \tau' = \tau^{-1}$$

$$\vdash \&_{\texttt{true}} : \tau \qquad\qquad \vdash \&_{\texttt{false}} : \tau$$

The abstract syntax of an attack-defence tree t is presented in Table 4.1. A tree is either a leaf or the application of a tree operator to one or two sub-trees.

Based on the player type, a leaf a is either a basic action of the proponent or of the opponent. We denote the set of proponent's and opponent's basic actions by Act_p and Act_o, respectively. We assume that these two sets are disjoint, $Act_p \cap Act_o = \emptyset$. We denote by Act the set of all basic actions, $Act = Act_p \cup Act_o$.

There are two special types of leaves; $\&_{\texttt{true}}$ represents a trivially-successful action, and $\&_{\texttt{false}}$ represents a trivially-failed action.

Similarly to the formalism for attack trees, tree operators include conjunction, disjunction and negation, while we introduce a novel construct for player alternation. The conjunction, disjunction and negation operators are interpreted in the same way as in Sect. 3.1.1.

The changing player operator $t = \&_{\sim}(t')$ changes the goal of t' by changing the type of the player. Note that in this case the goal belongs to the opposite player. For instance, if t' belongs to an attacker with the corresponding goal (e.g., minimising), then the tree t belongs to a defender with the corresponding goal (e.g., maximising). Thus, the changing player operator flips the player from an attacker to a defender and vice versa, as highlighted by the side-condition of the corresponding rule, where $p^{-1} = o$ and $o^{-1} = p$.

Well-formedness. The syntax of Table 4.1 is overly liberal for it does not associate players to nodes. A simple type system, showed in the second section of the table, enforces such association defining a well-formedness condition. We denote by $Tree_\tau$ the set of well-formed attack-defence trees whose root belongs to a player τ. Based on the type of the player, we have $Tree_p$ when τ is the proponent and $Tree_o$ when τ is the opponent and we define $Tree = Tree_p \cup Tree_o$. In the following, we will refer to them as attack-defence trees or simply trees.

Polarity-consistent tree. Analogously to the model of attack trees, we introduce the notion of polarity consistency, to be exploited in the technical developments. We say that an action a occurs negatively in a tree, if a is under an odd number of negations. Otherwise, we say that an action a occurs positively. Such *polarities* are denoted with the symbols $-$ and $+$, respectively.

Definition 4.1 An attack-defence tree t is polarity-consistent iff there is no action that occurs both positively and negatively in t.

Example. Let us introduce an example that we will develop throughout the chapter. We consider a fragment of a Radio-Frequency Identification (RFID) system managing goods in a warehouse, studied in [BKMS12]. Particularly, we consider an attacker (proponent) whose goal consists in removing the RFID tags from goods with the help of an insider.

In order to attain the goal, the attacker can "bribe", "threaten", "blackmail", or "trick" the insider. For bribing a person the attacker has to "identify a corruptible subject" and "bribe the subject". The defender (opponent) can protect against bribery by "thwarting employees", which can be done by "training for security" and by "threatening to fire the employees".

In case the attacker decides instead to "trick" the insider by placing a fake tag, he/she can either "send false replacement tags" or give a "false management order" to do it. The latter can be done by "infiltrating the management" and "ordering replacement tags". To fight such attacks, the defender can provide the employees with "training for trick".

The corresponding attack-defence tree is given in Figure 4.1. We decorate internal nodes with labels to keep track of sub-goals, hence making the tree more informative and human-readable. We label the leaves to refer to them easily. The example is formalised as a tree t, displayed in Figure 1, represented by the following syntactic term:

$$\begin{aligned} t = \quad & \&_\vee(\&_\wedge(\mathrm{is}, \&_\wedge(\mathrm{bs}, \&_\sim(\&_\wedge(\mathrm{t1},\mathrm{tf})))), \\ & \&_\vee(\mathrm{t}, \&_\vee(\mathrm{b}, \&_\wedge(\&_\vee(\&_\wedge(\mathrm{st}, \&_\sim(\&_\wedge(\mathrm{at}, \&_\sim(\mathrm{ba})))), \&_\wedge(\mathrm{im},\mathrm{ot})), \&_\sim(\mathrm{t2}))))) \end{aligned}$$

In order to enable a direct comparison with the evaluation techniques in the literature, our attack-defence tree does not contain negation. We will see, however, that as far as the calculation is concerned, negation would be treated similarly to the changing player operator.

4.1.2 Semantics in the Boolean Case

As mentioned above, we construe an attack-defence tree as an interaction between the proponent and the opponent. Corresponding to the model of attack trees, in the Boolean setting the investigation of the feasibility of a scenario is related to answering questions such as "Is there a successful attack/defence?".

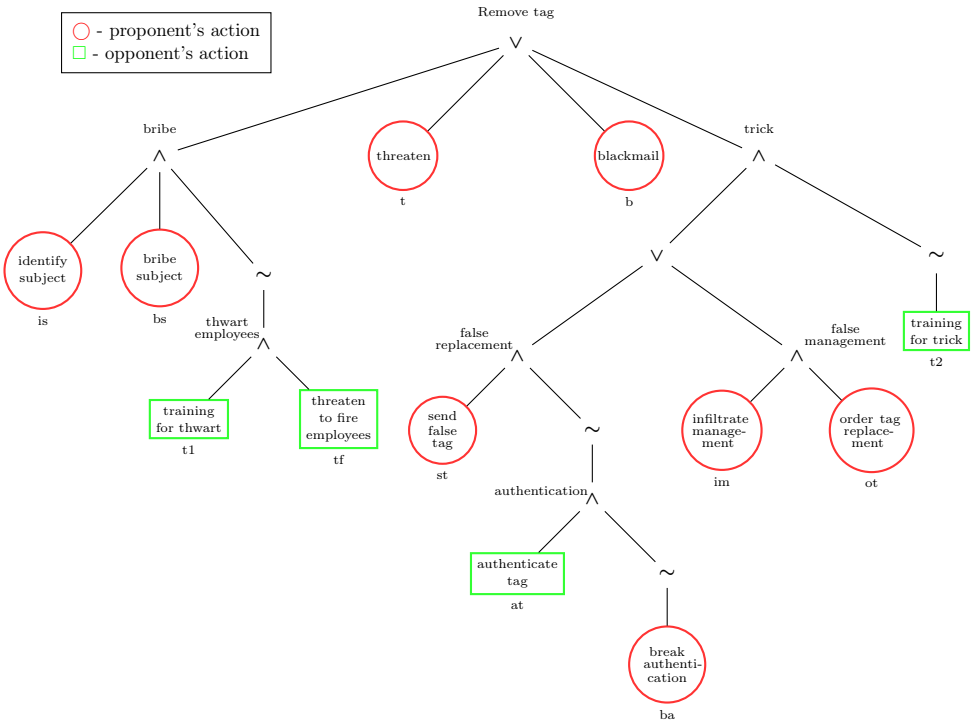


Figure 4.1: Attack-defence tree for removing RFID tags in a warehouse.

In this setting, we associate with each basic action a value from the Boolean set $\mathbb{B}$, where *true* corresponds to performing and *false* corresponds to not performing the action. We consider $\mathbb{B}$ to be totally ordered such that $\max\{tt, ff\} = tt$ and $\min\{tt, ff\} = ff$. The tree operators are interpreted as the Boolean operators, where the changing player operator is interpreted as negation.

We define a Boolean assignment of basic actions for a given player τ as follows: a Boolean assignment m_τ is an arbitrary function that assigns a value $b \in \mathbb{B}$ to each basic action $a \in Act_\tau$; $m_\tau : Act_\tau \to \mathbb{B}$. Thus, the Boolean assignment m is a pair (m_p, m_o), but we allow to write $m(a)$ as a shorthand for $m_p(a)$ when $a \in Act_p$ and $m_o(a)$ when $a \in Act_o$. We say that the main goal described by a tree succeeds if the Boolean assignment evaluates the tree to *true*.

For evaluating the feasibility of an attack-defence tree, we present two evaluation techniques, termed semantic and algorithmic evaluations, respectively.

The *semantic evaluation* $M(t)$ of an attack-defence tree $t \in Tree_p$ is presented in Table 4.2. The evaluation analyses the tree t by considering all possible Boolean assignments of values to the basic actions of t. It computes the

Table 4.2: The Boolean semantic evaluation of an attack-defence tree.

$$
\begin{array}{rcl}
M(t) & = & (\min\{\max\{\mathcal{B}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\} \mid \\
& & \qquad m_p \text{ Boolean assignment}\}, \\
& & \max\{\min\{\mathcal{B}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\} \mid \\
& & \qquad m_p \text{ Boolean assignment}\})
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{B}[\![a]\!]m & = & m(a) \\
\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!]m & = & \mathcal{B}[\![t_1]\!]m \wedge \mathcal{B}[\![t_2]\!]m \\
\mathcal{B}[\![\&_\vee(t_1, t_2)]\!]m & = & \mathcal{B}[\![t_1]\!]m \vee \mathcal{B}[\![t_2]\!]m \\
\mathcal{B}[\![\&_\neg(t)]\!]m & = & \neg\mathcal{B}[\![t]\!]m \\
\mathcal{B}[\![\&_\sim(t)]\!]m & = & \neg\mathcal{B}[\![t]\!]m \\
\mathcal{B}[\![\&_{\texttt{true}}]\!]m & = & tt \\
\mathcal{B}[\![\&_{\texttt{false}}]\!]m & = & ff
\end{array}
$$

pair of minimum and maximum success values of the proponent. If the proponent is an attacker, then it computes the minimum and the maximum values of an attack. Otherwise, it computes the minimum and maximum values of a defence. We observe that if the main goal of the scenario (represented by the root of the tree) is successful for the proponent, then it is not successful for the opponent. Similarly, if the proponent wants to maximise the success of the main goal, then the opponent wants to minimise it. Thus, the players have opposite goals. We integrate this consideration into our technique by minimising the value of t over all opponent's Boolean assignments m_o, and then maximising it over all proponent's Boolean assignments m_p. This is illustrated by the second component of $M(t)$ in Table 4.2, which computes the maximum success value of the proponent. The first component of $M(t)$ computes the minimum success value of the proponent. Therefore, the computation maximises the value over all m_o's and then minimises it over all m_p's.

The analysis $\mathcal{B}[\![t]\!]m$ of the tree t, displayed in the second part of Table 4.2, is performed recursively on the structure of t. It is worth noticing that the evaluation rules for negation and changing player unfold in the same way but are supported by a different semantic reason. In case of negation, $t = \&_\neg t'$, the player remains the same, say proponent, and his/her goal on $\&_\neg t'$ is the same goal he/she has on t, say minimise. However, minimising over $\&_\neg t'$ corresponds to maximising over t. In case of changing player, $\&_\sim t'$, the player changes and the objective changes accordingly. This is modelled again resorting to negation. Hence, both cases are treated identically but what is the consequence in one case is the reason for using negation in the other.

The result of the semantic evaluation is interpreted in the same way as discussed in Sect 3.1.2: (ff, ff) denotes a secure system, (tt, tt) a flawed system and (ff, tt) a vulnerable one – assuming the root belongs to the attacker.

Table 4.3: The Boolean algorithmic evaluation of an attack-defence tree.

$INT(a)$	$=$	$\begin{cases} (\mathit{ff}, tt) & \text{if} \quad a \in Act_p \\ (tt, \mathit{ff}) & \text{if} \quad a \in Act_o \end{cases}$	(Case-BA)
$INT(\&_\wedge(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\}$ $in\ (min_1 \wedge min_2, max_1 \wedge max_2)$	(Case-and)
$INT(\&_\vee(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\}$ $in\ (min_1 \vee min_2, max_1 \vee max_2)$	(Case-or)
$INT(\&_\neg(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (\neg max, \neg min)$	(Case-neg)
$INT(\&_\sim(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (\neg max, \neg min)$	(Case-change)
$INT(\&_{\texttt{true}})$	$=$	(tt, tt)	(Case-tt)
$INT(\&_{\texttt{false}})$	$=$	$(\mathit{ff}, \mathit{ff})$	(Case-ff)

The semantic evaluation characterises the analysis in a natural way, for it explicitly considers all the interactions interwoven in a tree in terms of assignments to the leaves. Nonetheless, it gives rise to an exponential computation already in the Boolean case, the satisfiability problem being NP-complete. Therefore, an evaluation technique that enjoy a lower complexity is needed. In particular, we face the problem of defining those restrictions on attack-defence trees under which more efficient methods are sound with respect to the semantic evaluation, our standard of reference.

The *algorithmic evaluation* $INT(t)$ of an attack-defence tree $t \in \mathit{Tree}_p$ is presented in Table 4.3. Similarly to the semantic evaluation, it computes the pair of minimum and maximum success values of the proponent. It considers the values of basic actions and propagates them up to the root. The propagation is performed according to the rules given in Table 4.3. The rule (Case-BA) assigns the minimum and the maximum success values to the actions based on the player type. Observe that, as the players have opposite goals, the success values are also opposite. The following four rules define the computation for operators. Conjunction and disjunction are treated in the standard way, hence let us focus on the negation and changing player operators. Both operators change the goal of the player. The negation operator negates the value of its sub-tree resulting in a change of optimisation goal ($min = \neg max$), while the changing player operator changes the optimisation goal by changing the player and this is modelled by negating the value of the sub-tree ($min = \neg max$). Thus,

in both rules we first swap the minimum and maximum values, and then apply negation. The last two rules are independent from the players and represent always successful and failed actions.

As for attack trees, in the model of attack-defence trees, the semantic and algorithmic evaluations might lead to different results, as we can see by considering the attack-defence tree $t = \&_{\wedge}(a, \&_{\neg}(a))$, where $a \in Act_p$. The result of the semantic evaluation is $M(t) = (ff, ff)$, while the results of the algorithmic evaluation is $INT(t) = (ff, tt)$. However, observe that t is not polarity-consistent. As a matter of fact, if we restrict to polarity-consistent trees, then the two evaluations are equivalent.

Theorem 4.2 *Let $t \in Tree$ be a polarity-consistent attack-defence tree. Then*

$$M(t) = INT(t)$$

The semantic evaluation considers all possible Boolean assignments m, thus being exponential in the size of t. The implementation of the algorithmic evaluation consists in a bottom-up traversal of t, and thus is linear in the size of the tree. Therefore, in case of polarity-consistent trees, our method offers a dramatic improvement in performance, hence in scalability.

4.1.3 Semantics in the Probabilistic Case

As we have seen in Sect. 3.1.3 for attack trees, the probabilistic setting generalises the Boolean one. In the following, we give a brief explanation of the setting and the evaluations, focusing on the novelties and omitting redundant details.

In the probabilistic setting, we consider the interval $[0, 1]$, where 0 corresponds to failure and 1 corresponds to success. The questions tackled in the probabilistic setting are, e.g., "What is the maximum probability of an attack?" or "How vulnerable is the system with respect to this attack goal?".

In this setting the tree operators are interpreted differently than the Boolean operators, and they satisfy only commutativity and associativity. This is in line with the model of attack trees.

Here as well, we assume that each basic action $a \in Act$ has two associated success probabilities; success probability $p_1(a)$ in case of not performing a, and success probability $p_2(a)$ in case of performing a, such that $p_1(a) \leq p_2(a)$. We consider the Boolean assignment m_τ as defined in the previous section, $m_\tau : Act_\tau \to \mathbb{B}$, and assume that an action a has probability of success $p_1(a)$ if $m(a)$ is *false* and has probability of success $p_2(a)$ if $m(a)$ is *true*. Choosing $p_1(a) = 0$ and $p_2(a) = 1$ coincides with the Boolean case.

The evaluation of attack-defence trees in the probabilistic setting follows the development for the Boolean setting: first, we characterise the solution to our problem in a top-down fashion, and then we investigate what restrictions on the model allow to devise an algorithmic approach with lower complexity.

Table 4.4: The probabilistic semantic evaluation of an attack-defence tree.

$$
\begin{array}{rcl}
M(t) & = & (\min\{\max\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\} \mid \\
& & \qquad m_p \text{ Boolean assignment}\}, \\
& & \max\{\min\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\} \mid \\
& & \qquad m_p \text{ Boolean assignment}\})
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{P}[\![a]\!]m & = & \begin{cases} p_2(a) & \text{if} \quad m(a) = tt \\ p_1(a) & \text{if} \quad m(a) = ff \end{cases} \\
\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m & = & \mathcal{P}[\![t_1]\!]m \cdot \mathcal{P}[\![t_2]\!]m \\
\mathcal{P}[\![\&_\vee(t_1, t_2)]\!]m & = & 1 - (1 - \mathcal{P}[\![t_1]\!]m) \cdot (1 - \mathcal{P}[\![t_2]\!]m) \\
\mathcal{P}[\![\&_\neg(t)]\!]m & = & 1 - \mathcal{P}[\![t]\!]m \\
\mathcal{P}[\![\&_\sim(t)]\!]m & = & 1 - \mathcal{P}[\![t]\!]m \\
\mathcal{P}[\![\&_{\texttt{true}}]\!]m & = & 1 \\
\mathcal{P}[\![\&_{\texttt{false}}]\!]m & = & 0
\end{array}
$$

The *semantic evaluation* $M(t)$ of an attack-defence tree $t \in \mathit{Tree}_p$ is illustrated in Table 4.4. It computes the minimum and the maximum success probability of a scenario by analysing the tree t over all Boolean assignments from which the probability values are inferred. Observe that also here the players have opposite goals, e.g., the proponent wants to maximise the overall probability of success, while the opponent wants to minimise it.

The result of the computation, when the proponent is an attacker, is interpreted as follows. The maximum success probability p shows the existence of an attack with probability p. In this case, we say that the system is p-vulnerable.

The *algorithmic evaluation* $\mathit{INT}(t)$ of an attack-defence tree $t \in \mathit{Tree}_p$ is given in Table 4.5. It traverses the tree from the leaves to the root and propagates the values of the basic actions. Similarly to the Boolean case, as the negation and changing player operators change the goal of the player, we first swap the minimum and maximum values before applying negation.

We shall restrict to polarity-consistent trees in order to show the equivalence of the two evaluations.

Theorem 4.3 *Let $t \in \mathit{Tree}$ be a polarity-consistent attack-defence tree. Then*

$$M(t) = \mathit{INT}(t)$$

Hence, in the probabilistic setting polarity-consistency allows to scale from an exponential to a linear complexity.

Example. Consider the attack-defence tree t presented in Figure 4.1. Observe that t is a polarity-consistent tree, thus we can apply the algorithmic evaluation for computing the maximum probability of success at the root. Table 4.6

Table 4.5: The probabilistic algorithmic evaluation of an attack-defence tree.

$INT(a)$	$=$	$\begin{cases} (p_1(a), p_2(a)) & \text{if} \quad a \in Act_p \\ (p_2(a), p_1(a)) & \text{if} \quad a \in Act_o \end{cases}$	(Case-BA)
$INT(\&_\wedge(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\}$ $in\ (min_1 \cdot min_2, max_1 \cdot max_2)$	(Case-and)
$INT(\&_\vee(t_1, t_2))$	$=$	$let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\}$ $in\ (1 - (1 - min_1) \cdot (1 - min_2),$ $1 - (1 - max_1) \cdot (1 - max_2))$	(Case-or)
$INT(\&_\neg(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (1 - max, 1 - min)$	(Case-neg)
$INT(\&_\sim(t))$	$=$	$let\ (min, max) = INT(t)$ $in\ (1 - max, 1 - min)$	(Case-change)
$INT(\&_{\texttt{true}})$	$=$	$(1, 1)$	(Case-tt)
$INT(\&_{\texttt{false}})$	$=$	$(0, 0)$	(Case-ff)

lists possible probability values for basic actions (the last column is for later reference).

Following the algorithmic computation we obtain $INT(t) = (0, 0.97)$, that is, the system is 0.97-vulnerable. The details of the computation with the values for the internal nodes are presented in Appendix D.1.

4.2 Attack-Defence Trees with Cost

Analogously to the model of attack trees, we extend our evaluation techniques by considering a single cost for basic actions. In this section, we evaluate the minimum cost of an attack or defence in the Boolean and probabilistic settings. Sect. 4.2.1 thoroughly explains the Boolean case, while the developments are generalised to the probabilistic setting in Sect. 4.2.2.

4.2.1 Cost in the Boolean Case

We extend the model of attack-defence trees described in Sect. 4.1.2 with a single cost for basic actions. Therefore, each basic action is associated with a pair of Boolean and cost values.

When we consider costs, we can focus on questions such as "What is the minimum cost of an attack?" or "How much does it cost to protect a system

Label	Name of the Node	p_1	p_2	c
is	identify subject	0.2	0.8	80
bs	bribe subject	0	0.7	100
t1	training for thwart	0.1	0.3	0
tf	threaten to fire employees	0.1	0.4	0
t	threaten	0	0.7	160
b	blackmail	0	0.7	150
st	send false tag	0	0.5	50
at	authenticate tag	0.1	0.7	0
ba	break authentication	0.1	0.6	85
im	infiltrate management	0	0.5	70
ot	order tag replacement	0	0.6	0
t2	training for trick	0.1	0.4	0

Table 4.6: The values of probability and cost for the basic actions of the example.

in a given scenario?". Observe that cost-related questions are player-dependent, meaning that the model is evaluated from a given player's perspective. Since we assumed that the basic actions are independent, we need to consider one player's values only. For instance, for computing the minimum cost of an attack we need only the cost of the attacker's actions and do not require the cost of the defender's actions. Thus, in our evaluation techniques we consider only the cost of the proponent's actions, and do not consider the cost of the opponent's actions.

In the following we consider the set $D = \mathbb{B} \times \mathbb{Q}_{\geq 0}$. In order to link the cost parameter to the existing model in the Boolean setting, we assume that each basic action of the proponent $a \in Act_p$ has two associated costs. One is the cost of not performing a (0 in the following), the other is the cost $c(a)$ of performing a. We set both costs of the opponent actions to 0.

As we have seen for attack trees, extending the model with costs and evaluating the pairs of success *and* cost values lead to multi-parameter optimisation. Moreover, such pairs are incomparable in case the goal of a player is to maximise one parameter while minimising the other. In order to address multi-parameter optimisation in the case of conflicting objectives, we resort to Pareto efficiency relying on the functions MR^{+-} and MR^{--} defined in Sect. 3.2.1 to only retain optimal solutions.

Following the previous developments, we present two evaluation techniques.

The *semantic evaluation* $M(t, A)$ of an attack-defence tree $t \in Tree_p$ and a set of actions A is illustrated in Table 4.7. It computes a pair, where the first argument is a set computed by the function MR^{--} and consists of all pairs that have lower success value and lower cost of the proponent actions compared to other solutions, and the second argument is the set computed by the function MR^{+-} and consists of all pairs that have higher success value and lower cost of the proponent actions compared to other solutions.

As we discussed in Sect. 4.1.2, if the proponent wants to maximise the success

Table 4.7: The Boolean semantic evaluation of an attack-defence tree with cost.

$$M(t,A) = (MR^{--}(\{\ (f_1^{m_p}(t), \mathfrak{b}_p) \mid cost(m_p, A) \leq \mathfrak{b}_p\}),\ MR^{+-}(\{\ (f_2^{m_p}(t), \mathfrak{b}_p) \mid cost(m_p, A) \leq \mathfrak{b}_p\}))$$

$$f_1^{m_p}(t) = \max\{\mathcal{B}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$f_2^{m_p}(t) = \min\{\mathcal{B}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$cost(m_p, A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m_p(a) = \mathit{ff} \\ c(a), & \text{if } m_p(a) = \mathit{tt} \end{cases}$$

of the main goal, then the opponent wants to minimise it. In other words, the players affect the computation of the success value of the main goal in opposite ways, e.g., when one wants to maximise, the other wants to minimise and vice versa. The functions MR^{--} and MR^{+-} evaluate the success values based on the goal of the proponent. In order to consider the effect of the opponent with the opposite goal, we define functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$ given in Table 4.7. The functions compute respectively the maximum and minimum success values over all Boolean assignments m_o for a given Boolean assignment m_p.

The *algorithmic evaluation* $INT(t)$ for an attack-defence tree $t \in \mathit{Tree}_p$ is given in Table 4.8. It again computes a pair, where the first argument consists of all pairs that have lower success value and cost of the proponent actions, and the second argument consists of all pairs that have higher success value and lower cost of the proponent actions. Such sets are computed in a bottom-up fashion, defined by the rules presented in Table 4.8. The rules extend the ones for the success value computation, presented in Sect. 4.1.2, Table 3.3.

The rule (Case-BA) assigns the sets MR^{--} and MR^{+-} to the basic actions. Observe that the cost of not performing the proponent's actions, as well as both costs of the opponent's actions, is 0. The rules for conjunction (Case-and) and disjunction (Case-or) use the standard computation for success values and sum the costs. The negation and changing player operators evaluate the success value as described in Sect. 4.1.2, while leaving the cost unchanged. The rules (Case-tt) and (Case-ff) correspond to trivially-successful and trivially-failed actions, which are independent from the players, and thus have a cost equal to 0. Applying the functions MR^{--} and MR^{+-} in each rule of the evaluation allows to reduce the size of the sets in each step.

As we have seen on attack trees, the notion of polarity-consistency does not suffice to show the equivalence of the semantic and algorithmic evaluation when the notion of cost enters the model. The same applies to attack-defence trees, unsurprisingly. Consider the polarity-consistent tree $\&_\wedge(\&_\vee(a,b), \&_\vee(a, \&_\sim(d)))$,

Table 4.8: The Boolean algorithmic evaluation of an attack-defence tree with cost.

$$
\begin{array}{lcll}
INT(a) & = & \begin{cases} (MR^{--}(\{(\mathit{ff},0),(tt,c(a))\}), & \\ \ MR^{+-}(\{(\mathit{ff},0),(tt,c(a))\})) & \text{if} \quad a \in Act_p \\ (MR^{--}(\{(tt,0)\}), MR^{+-}(\{(\mathit{ff},0)\})) & \text{if} \quad a \in Act_o \end{cases} & \\
 & & & \text{(Case-BA)} \\
INT(\&_\wedge(t_1,t_2)) & = & \mathit{let}\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & \mathit{in}\ (MR^{--}(\{(b \wedge b', c+c') \mid (b,c) \in V_1, (b',c') \in V_2\}), & \\
 & & \quad MR^{+-}(\{(b \wedge b', c+c') \mid (b,c) \in W_1, (b',c') \in W_2\})) & \\
 & & & \text{(Case-and)} \\
INT(\&_\vee(t_1,t_2)) & = & \mathit{let}\ (V_i,W_i) = INT(t_i), i \in \{1,2\} & \\
 & & \mathit{in}\ (MR^{--}(\{(b \vee b', c+c') \mid (b,c) \in V_1, (b',c') \in V_2\}), & \\
 & & \quad MR^{+-}(\{(b \vee b', c+c') \mid (b,c) \in W_1, (b',c') \in W_2\})) & \\
 & & & \text{(Case-or)} \\
INT(\&_\neg(t)) & = & \mathit{let}\ (V,W) = INT(t) & \\
 & & \mathit{in}\ (MR^{--}(\{(\neg b, c) \mid (b,c) \in W\}), & \\
 & & \quad MR^{+-}(\{(\neg b, c) \mid (b,c) \in V\})) & \text{(Case-neg)} \\
INT(\&_\sim(t)) & = & \mathit{let}\ (V,W) = INT(t) & \\
 & & \mathit{in}\ (MR^{--}(\{(\neg b, c) \mid (b,c) \in W\}), & \\
 & & \quad MR^{+-}(\{(\neg b, c) \mid (b,c) \in V\})) & \text{(Case-change)} \\
INT(\&_{\mathtt{true}}) & = & (\{(tt,0)\},\{(tt,0)\}) & \text{(Case-tt)} \\
INT(\&_{\mathtt{false}}) & = & (\{(\mathit{ff},0)\},\{(\mathit{ff},0)\}) & \text{(Case-ff)}
\end{array}
$$

where $a, b \in Act_p, d \in Act_o$, $c(a) = 2$ and $c(b) = 1$, we see that the two evaluation techniques lead to different results. The semantic evaluation gives the result $M(t, \mathit{yield}(t)) = (\{(\mathit{ff},0)\},\{(\mathit{ff},0),(tt,2)\})$, while the algorithmic evaluation (presented in Figure. 4.2) gives the result $INT(t) = (\{(\mathit{ff},0)\},\{(\mathit{ff},0),(tt,3)\})$.

Again, to show the equivalence of the two evaluation we restrict to linear trees, i.e., trees where no action occurs twice (for any player).

Theorem 4.4 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M(t, \mathit{yield}(t)) = INT(t)$$

We measure complexity of the two evaluation methods by calculating the number of set operations. The semantic evaluation is exponential, while the algorithmic evaluation is linear and hence presents a dramatic improvement in the case of linear trees.

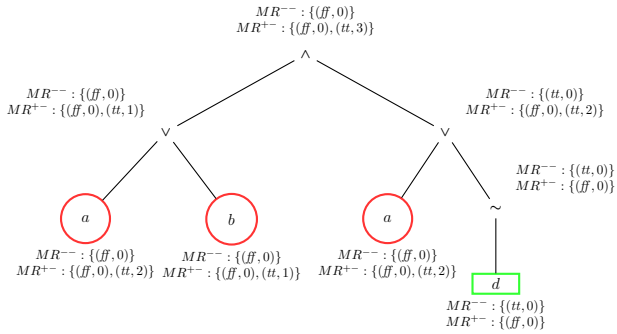


Figure 4.2: The Boolean algorithmic evaluation with cost of the attack tree $t = \&_\wedge(\&_\vee(a,b), \&_\vee(a, \&_\sim(d)))$.

4.2.2 Cost in the Probabilistic Case

In the following we briefly generalise our development to the probabilistic setting, concentrating on the differences with respect to the Boolean setting.

Note, that the same observation regarding the cost to the proponent and to the opponent applies. Here we consider the set $D = [0,1] \times \mathbb{Q}_{\geq 0}$. The cost is integrated to the basic actions in the same way. Thus, to extend the probabilistic model with costs, we assume that each basic action of the proponent player $a \in Act_p$ has two associated costs, 0 in case of not performing a, and $c(a)$ in case of performing a. We set both costs of the opponent's actions equal to 0.

As in the previous case, by considering probability *and* cost and focusing on maximising the first while minimising the other, we face a multi-parameter optimisation problem with conflicting objectives. Similarly to the Boolean setting, we provide two evaluation techniques based on Pareto efficiency by considering the functions MR^{--} and MR^{+-}. We compute the set of Pareto-efficient solutions for answering questions such as "What is the maximum probability and the minimum cost of an attack?".

The *semantic evaluation* $M(t, A)$ of an attack-defence tree $t \in Tree_p$ and a given set A is illustrated in Table 4.9. The evaluation follows the corresponding one for the Boolean case, described in the Sect. 4.2.1. The only difference is that the tree t is evaluated over the Boolean assignments by considering the probabilistic analysis $\mathcal{P}[\![t]\!]$ instead of the Boolean one $\mathcal{B}[\![t]\!]$ (and considering the corresponding probabilistic values for each action). The result of the evaluation is the pair of sets MR^{--} and MR^{+-}, corresponding to the set of Pareto-efficient solutions.

The *algorithmic evaluation* $INT(t)$ for a tree $t \in Tree_p$ is given in Table 4.10. It traverses the tree from the leaves to the root according to the rules presented in the table. The rules follow the corresponding ones of the Boolean setting.

Analogously to the previous section, we shall restrict to linear trees in order

Table 4.9: The probabilistic semantic evaluation of an attack-defence tree with cost.

$$M(t,A) = (MR^{--}(\{\ (f_1^{m_p}(t), \mathfrak{b}_p) \mid cost(m_p, A) \leq \mathfrak{b}_p\}),\ MR^{+-}(\{\ (f_2^{m_p}(t), \mathfrak{b}_p) \mid cost(m_p, A) \leq \mathfrak{b}_p\}))$$

$$f_1^{m_p}(t) = \max\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$f_2^{m_p}(t) = \min\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$cost(m_p, A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m_p(a) = \mathit{ff} \\ c(a), & \text{if } m_p(a) = \mathit{tt} \end{cases}$$

to show the equivalence of the two evaluations.

Theorem 4.5 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M(t, yield(t)) = INT(t)$$

Again, a syntactic restriction allows to develop a sound evaluation technique that is linear in the size of the tree as opposed to the exponential complexity that characterises the general case.

Example. Consider the linear attack-defence tree t presented in Figure 4.1. Table 4.6 lists possible values for probability and cost for basic actions.

In order to detect the attacks with maximum probability of success and minimum cost, we apply the algorithmic evaluation. The details of the computation are presented in Appendix D.2, while at the root we obtain:

$$INT(t) = (\{(0,0)\}, \{(0,0), (0.1,50), (0.18,70), (0.26,120), (0.7,150), (0.73,200), (0.75,220), (0.78,270), (0.91,310), (0.92,360), (0.93,430), (0.95,490), (0.96,540), (0.97,695)\})$$

The overall result of the evaluation, i.e., the set of efficient solutions for the goal representing the Pareto frontier of the problem, is displayed in Figure 4.3. The probability of successful attacks ranges from 0 to 0.97 and the corresponding cost ranges from 0 to 695. We can conclude that the system under study is (p,c)-vulnerable for all the incomparable pairs in the Pareto frontier. In particular, the attack is not attainable trivially (all pairs with probability greater than zero require a cost greater than zero).

Table 4.10: The probabilistic algorithmic evaluation of an attack-defence tree with cost.

$$
\begin{array}{lcl}
INT(a) & = & \begin{cases} (MR^{--}(\{(p_1(a),0),(p_2(a),c(a))\}), & \\ \quad MR^{+-}(\{(p_1(a),0),(p_2(a),c(a))\})) & \text{if} \quad a \in Act_p \\ (MR^{--}(\{(p_2(a),0)\}), MR^{+-}(\{(p_1(a),0)\})) & \text{if} \quad a \in Act_o \end{cases} \\
INT(\&_\wedge(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} \\
 & & in\ (MR^{--}(\{(p \cdot p', c+c') \mid (p,c) \in V_1, (p',c') \in V_2\}), \\
 & & \quad MR^{+-}(\{(p \cdot p', c+c') \mid (p,c) \in W_1, (p',c') \in W_2\})) \\
INT(\&_\vee(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} \\
 & & in\ (MR^{--}(\{(1-(1-p)(1-p'), c+c') \mid \\
 & & \qquad (p,c) \in V_1, (p',c') \in V_2\}), \\
 & & \quad MR^{+-}(\{(1-(1-p)(1-p'), c+c') \mid \\
 & & \qquad (p,c) \in W_1, (p',c') \in W_2\})) \\
INT(\&_\neg(t)) & = & let\ (V,W) = INT(t) \\
 & & in\ (MR^{--}(\{(1-p,c) \mid (p,c) \in W\}), \\
 & & MR^{+-}(\{(1-p,c) \mid (p,c) \in V\})) \\
INT(\&_\sim(t)) & = & let\ (V,W) = INT(t) \\
 & & in\ (MR^{--}(\{(1-p,c) \mid (p,c) \in W\}), \\
 & & MR^{+-}(\{(1-p,c) \mid (p,c) \in V\})) \\
INT(\&_{\texttt{true}}) & = & (\{(1,0)\},\{(1,0)\}) \\
INT(\&_{\texttt{false}}) & = & (\{(0,0)\},\{(0,0)\})
\end{array}
$$

4.3 Attack-Defence Trees with Multiple Costs

In this section we extend the model to deal with multiple costs for basic actions. Observing that the Boolean setting is a special case of the probabilistic one, in the following we describe the extended model only in the probabilistic setting, focusing on the extensions with respect to a single cost model. The developments retrace those for the model of attack trees presented in Sect. 3.3.

We consider the set $D = [0,1] \times \mathbb{Q}^n_{\geq 0}$. We assume that each basic action of the proponent $a \in Act_p$ has two vectors of n associated costs: a vector of 0's in case of not performing a, and a vector $\gamma : Act_p \to \mathbb{Q}^n_{\geq 0}$ in case of performing a. We denote $\gamma = (c_1(a), \cdots, c_n(a))$, $c_i : Act_p \to \mathbb{Q}_{\geq 0}$, $i \in [1,n]$. Here as well, when adding costs we resort to point-wise summation of vectors. We set the

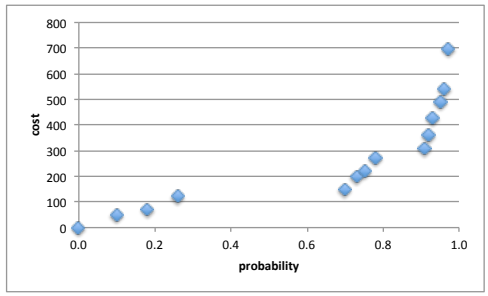


Figure 4.3: Pareto-efficient solutions for the attack-defence tree t.

cost of the opponent's actions to vectors of 0's.

The functions MR^{--} and MR^{+-} are extended as in Sect. 3.3 by introducing polarity modifications of the comparison operators as follows: $\sqsupseteq^{+}$ is $\sqsupseteq$, $\sqsupset^{+}$ is $\sqsupset$, $\sqsupseteq^{-}$ is $\sqsubseteq$ and $\sqsupset^{-}$ is $\sqsubset$. The sign "+" corresponds to the maximisation of the parameters and keeps the operator as it is, while the sign "−" corresponds to the minimisation of the parameters, therefore it changes the operator.

We define a general frontier function, where $s_i \in \{+,-\}$ and $Z \subseteq D$, as follows:

$$\begin{aligned} MR^{s_0,\cdots,s_n}(Z) &= \{(x_0,\cdots,x_n) \in Z \mid \forall (x'_0,\cdots,x'_n) \in Z : \\ &\quad x'_0 \sqsupseteq^{s_0} x_0 \wedge \cdots \wedge x'_n \sqsupseteq^{s_n} x_n \Rightarrow x'_0 = x_0 \wedge \cdots \wedge x'_n = x_n\} \\ &= \{(x_0,\cdots,x_n) \in Z \mid \forall (x'_0,\cdots,x'_n) \in Z : \\ &\quad ((x_0 \sqsupseteq^{s_0} x'_0) \vee (x_1 \sqsupset^{s_1} x'_1) \vee \cdots \vee (x_n \sqsupset^{s_n} x'_n)) \wedge \cdots \\ &\quad \cdots \wedge ((x_0 \sqsupset^{s_0} x'_0) \vee (x_1 \sqsupset^{s_1} x'_1) \vee \cdots \vee (x_n \sqsupseteq^{s_n} x'_n))\} \end{aligned}$$

As in Sect. 3.3, the function $MR^{s_0,\cdots,s_n}$ computes the efficient solutions for multiple parameters by maximising the parameter values if $s_i = +$ and minimising it if $s_i = -$. Each $\sqsupseteq^{s_i}$ is in fact a total order (on [0,1] or $\mathbb{Q}_{\geq 0}$) and hence $\neg(x'_i \sqsupseteq^{s_i} x_i)$ is equivalent to $x_i \sqsupset^{s_i} x'_i$. Observe that with this notation we get MR^{--} when we take $n = 1$ and $s_0 = s_1 = -$, and we get MR^{+-} when we take $n - 1$ and $s_0 = +, s_1 = -$.

The definitions of the semantic evaluation M^* and algorithmic evaluations INT^* closely follow that of the corresponding ones in Sects. 4.2 and 3.3. The two evaluations are given in Table 4.11 and 4.12, respectively.

We show the equivalence of two evaluation techniques by restricting to linear trees.

Theorem 4.6 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M^*(t, yield(t)) = INT^*(t)$$

Table 4.11: The probabilistic semantic evaluation of a tree with multiple cost.

$$\begin{aligned} M^*(t,A) \quad = \quad & (MR^{-\cdots-}(\{\ (f_1^{m_p}(t), \mathfrak{b}_p^1, \cdots, \mathfrak{b}_p^n) \mid cost_i(m_p, A) \leq \mathfrak{b}_p^i\}), \\ & MR^{+\cdots-}(\{\ (f_2^{m_p}(t), \mathfrak{b}_p^1, \cdots, \mathfrak{b}_p^n) \mid cost_i(m_p, A) \leq \mathfrak{b}_p^i\})) \end{aligned}$$

$$f_1^{m_p}(t) = \max\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$f_2^{m_p}(t) = \min\{\mathcal{P}[\![t]\!](m_p, m_o) \mid m_o \text{ Boolean assignment}\}$$

$$cost_i(m_p, A) = \sum_{a \in A} \begin{cases} 0, & \text{if } m_p(a) = \mathit{ff} \\ c_i(a), & \text{if } m_p(a) = \mathit{tt} \end{cases}$$

4.4 Concluding Remarks

The growing centrality of technology requires a thorough investigation of the security properties of complex systems with respect to cyber and physical attacks, as well as consideration of possible defences undertaken to counter such attacks.

Attack-defence trees are a useful tool to study attack-defence scenarios and present the interaction between an attacker and a defender in an intuitive way. Moreover, such models are relied on to develop quantitative analyses of attacks and defences. Many evaluation methods consider one-parameter trees or analyse multi-parameter trees focusing on one specific aspect of the scenario, such as probability of success or cost. Nonetheless, in case of multi-parameter models, conflicting objectives may lead to incomparable values, which require to optimise all parameters at once on pain of computing sub-optimal solutions.

In order to tackle this issue, we have presented evaluation techniques for multi-parameter attack-defence trees that optimise all parameters at once, levering the concept of Pareto efficiency. Our developments have been carried out extending the language-based formalism of Ch. 3 to attack-defence trees, introducing a novel operator for player alternation. In the extended language, the interaction between an attacker and a defender is made explicit by associating a player to each node thanks to a simple type system. We have called *proponent* the player at the root and *opponent* the other player.

We have extended the analyses first introduced in Ch. 3 on attack trees to analyse attack-defence scenarios, investigating aspects such as the feasibility and the cost of an attack or a defence. For each case we have illustrated the natural semantic evaluation technique as well as an algorithmic evaluation which enjoys a dramatic improvement in complexity, and we have proven under which conditions the latter can be relied on in place of the former. Both methods characterise the goal of the scenario with a set of Pareto-efficient solutions.

Table 4.12: The probabilistic algorithmic evaluation of a tree with multiple cost.

$$
\begin{array}{lcl}
INT^*(a) & = & \begin{cases} (MR^{--\cdots-}(\{(p_1(a),0,\cdots,0),(p_2(a),c_1(a),\cdots,c_n(a))\}), \\ \quad MR^{+-\cdots-}(\{(p_1(a),0,\cdots,0),(p_2(a),c_1(a),\cdots,c_n(a))\})) \\ \qquad\qquad\qquad\qquad \text{if } a \in Act_p \\ (MR^{--\cdots-}(\{(p_2(a),0,\cdots,0)\}), \\ \quad MR^{+-\cdots-}(\{(p_1(a),0,\cdots,0)\})) \\ \qquad\qquad\qquad\qquad \text{if } a \in Act_o \end{cases} \\
INT^*(\&_\wedge(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} \\
 & & in\ (MR^{--\cdots-}(\{(p\cdot p', c_1+c_1', \cdots, c_n+c_n') \mid \\
 & & \qquad (p,c_1,\cdots,c_n) \in V_1, (p',c_1',\cdots,c_n') \in V_2\}), \\
 & & \quad MR^{+-\cdots-}(\{(p\cdot p', c_1+c_1', \cdots, c_n+c_n') \mid \\
 & & \qquad (p,c_1,\cdots,c_n) \in W_1, (p',c_1',\cdots,c_n') \in W_2\})) \\
INT^*(\&_\vee(t_1,t_2)) & = & let\ (V_i,W_i) = INT(t_i), i \in \{1,2\} \\
 & & in\ (MR^{--\cdots-}(\{(1-(1-p)(1-p'), c_1+c_1', \cdots, c_n+c_n') \mid \\
 & & \qquad (p,c_1,\cdots,c_n) \in V_1, (p',c_1',\cdots,c_n') \in V_2\}), \\
 & & \quad MR^{+-\cdots-}(\{(1-(1-p)(1-p'), c_1+c_1', \cdots, c_n+c_n') \mid \\
 & & \qquad (p,c_1,\cdots,c_n) \in W_1, (p',c_1',\cdots,c_n') \in W_2\})) \\
INT^*(\&_\neg(t)) & = & let\ (V,W) = INT(t) \\
 & & in\ (MR^{--\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in W\}), \\
 & & \quad MR^{+-\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in V\})) \\
INT^*(\&_\sim(t)) & = & let\ (V,W) = INT(t) \\
 & & in\ (MR^{--\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in W\}), \\
 & & \quad MR^{+-\cdots-}(\{(1-p,c_1,\cdots,c_n) \mid (p,c_1,\cdots,c_n) \in V\})) \\
INT^*(\&_{\texttt{true}}) & = & (\{(1,0,\cdots,0)\},\{(1,0,\cdots,0)\}) \\
INT^*(\&_{\texttt{false}}) & = & (\{(0,0,\cdots,0)\},\{(0,0,\cdots,0)\})
\end{array}
$$

Our current methods focus on the players independently: for evaluating the cost of the proponent p, we set the cost of the opponent to 0 and assign a budget to p. In future work, it would be interesting to investigate the extension of the model with a budget for the opponent, so as to compute the optimal solutions for both players at once.

We have now concluded Part I where we approached the challenge of attack and attack-defence trees analysis improving on the literature but remaining in the realm of direct evaluation of tree-like objects. In the remainder of the disser-

tation we shall instead explore the connection between attack and attack-defence trees and stochastic models, moving from a static to a dynamic perspective on the problem of security.

Part II

From Attack-Defence Trees to Security Games

Formal graphical models help identify and understand the security threats to system and to study the possible countermeasures. In the previous part we studied two such approaches, namely attack trees and attack-defence trees, and proposed evaluation techniques that address multi-parameter optimisation of objectives such as probability and cost. The techniques we developed for attack trees and attack-defence trees optimise all parameters at once and compute the set of optimal solutions, levering the concept of Pareto efficiency.

Attack-defence scenarios, illustrated through attack-defence trees, can be seen as the interaction between two players, an attacker and a defender, where the players compete with each other for achieving conflicting objectives. In this part of the work we look into attack-defence scenarios from this perspective and explore the relation between attack-defence trees and games in the stochastic domain. A game-theoretic approach to the analysis of attack-defence trees allows to reap the benefits of the great many developments in the area of game model checking.

The probabilistic features of many scenarios and the competitive behaviour of attackers and defenders make it natural to model attack-defence scenarios as *Stochastic Two-player Games*. We define a translation from attack-defence trees to STGs and use automated verification techniques such as *probabilistic model checking* for evaluating security properties and computing optimal attack and defence strategies to the players. In order to express the dependencies between attacker and defender actions we introduce sequential operators in attack-defence trees, extending our formalism. This allows to handle the situations when the choices of players may depend on the history of the game.

We investigate probability and expected cost-related properties of a scenario and use Probabilistic Alternating-Time Temporal Logic with Rewards (rPATL) to express security properties. rPATL allows to specify quantitative properties and reason about the strategies available to the competing players. To support the modelling and verification of stochastic multi-player games we rely on the model checker PRISM-games.

Finally, we discuss an extension of our approach for analysing properties with multiple objectives. We consider *multi-objective probabilistic model checking* for stochastic games, which allows to verify multiple, conflicting objectives such as probability and expected cost. This is a key contribution in our approach to analysing attack-defence trees, as we have stressed throughout Part I. Observe that in Part I we always dealt with the *exact* cost of an attack, defined as the sum of the costs of the actions leading to the goal. On the other hand, the literature on stochastic model checking focuses on *expected* reward, weighing costs by the probability of incurring them. However, it is not clear how to compare an expected cost to a fixed budget available to a player. We will investigate in Part III how to overcome this limitation.

The organisation of this part is as follows. Chapter 5 provides background material on stochastic systems and the logics used in subsequent chapters. We

present our formalism for attack-defence trees, the proposed translation from trees to STGs, as well as the evaluation of security properties and synthesis of strategies in Ch. 6.

Chapter 5

Preliminaries: Probabilistic Models

This chapter aims at introducing necessary preliminaries on the probabilistic models used in Ch. 6 and Ch. 7. In particular, it presents Discrete-Time Markov Chains and Stochastic Two-player Games, as well as their analysis by means of probabilistic model checking.

5.1 Probabilistic Models

Probability is an important component for modelling unreliable and unpredictable system behaviour, for example fault-tolerant systems or communication networks. Many real-life systems have a stochastic nature, such as probabilistic protocols or randomised distributed systems. In order to express random and unpredictable behaviour, the models are annotated with information about the probability of the occurrence of events. In this part of the work we consider two probabilistic models: Discrete-Time Markov Chains (DTMCs), that represent purely probabilistic behaviour, and Stochastic Two-player Games (STGs), that model probabilistic systems embedding competitive behaviour.

The chapter is organised as follows. We begin in Sect. 5.1.1 with a brief background material on DTMCs. In Sect. 5.1.2 we give an overview of STGs and related concepts. Model checking DTMCs and games is discussed in Sects. 5.2.1 and 5.2.2, respectively. Finally, Sect. 5.3 presents the model checker PRISM. For a thorough treatment of the topic the reader is referred to [BK08, Ch.10] [KNP07, CFK+13a].

5.1.1 Discrete Time Markov Chains

Discrete-Time Markov Chains model the probabilistic behaviour of the system. In DTMCs the probability of making a transition from one state to another depends only on the current state and not on any past states, that is, the system is *memoryless*.

Definition 5.1 (DTMC) A Discrete-Time Markov Chain is a tuple $\mathcal{D} = (S, s_0, P, AP, L)$, where:

- S is a finite, non-empty set of states;
- $s_0 \in S$ is the initial state;
- $P : S \times S \to [0, 1]$ is a transition probability function such that for all states $s \in S$, $\sum_{s' \in S} P(s, s') = 1$;
- AP is a set of atomic propositions; and
- $L : S \to 2^{AP}$ is a labelling function.

The transition probability function indicates the probability $P(s, s')$ of moving from state s to state s' in a single transition. It is required that $\sum_{s' \in S} P(s, s') = 1$ for all state $s \in S$, thus terminating states are modelled by adding a self-loop. The function P can be viewed as the matrix $(P(s, s'))_{s,s' \in S}$, where each element $P(s, s')$ gives the probability of moving from s to s'. The labelling function L maps each state s to a set of atomic propositions.

A *path* through a DTMC is a sequence of states $\pi = s_0 s_1 \cdots$ where $s_i \in S$ and $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. A path can be infinite or finite. The sets of all infinite and finite paths starting in a state s are denoted by $Path_s$ and $Path_s^{fin}$, respectively. The i-th state of the path is denoted by $\pi[i]$.

Reasoning about quantitative properties of a DTMC requires determining the probabilities that certain paths are taken. This is done by defining a probability measure Pr_s over the set of infinite paths $Path_s$ based on σ-algebra. For more details we refer the reader to [BK08, Ch.10].

In addition to the probability, DTMCs can be augmented with *reward structures* of the form $r : S \times S \to \mathbb{Q}_{\geq 0}$, which map each transition to a non-negative rational number. In our work we use reward as a mechanism to model costs associated with performing an action, i.e., with taking a transition.

5.1.2 Stochastic Two-Player Games

Systems with both probabilistic and competitive behaviour can be modelled as stochastic games. In this dissertation, we consider Stochastic Two-player Games for modelling the interaction between the attacker and the defender of an attack-defence scenario.

Definition 5.2 (STG) A Stochastic Two-player Game is a tuple $\mathcal{G} = (\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_\circledcirc), P, T, AP, L)$, where:

- $\Pi = \{A, D\}$ is a set of players;
- $S = S_A \uplus S_D \uplus S_P \uplus S_\circledcirc$ is a finite set of states, partitioned into attacker states (S_A), defender states (S_D), probabilistic states (S_P) and final states ($S_\circledcirc$);
- $s_0 \in S$ is the initial state;
- α is a finite, non-empty set of actions;
- $P : S_P \times S \to [0, 1]$ is a probabilistic transition function such that for all probabilistic states $s \in S_P$ $\sum_{s' \in S} P(s, s') = 1$;
- $T : (S_A \cup S_D) \times \alpha \to S$ is a transition function;
- AP is a set of atomic propositions; and
- $L : S \to 2^{AP}$ is a labelling function.

An STG has two players, which we fix as A (attacker) and D (defender). Each player controls a subset of the states: A chooses between available actions $a \in \alpha$ in states $s \in S_A$, while D does so for states $s \in S_D$. The choice between outgoing transitions in probabilistic states $s \in S_P$ is made probabilistically, applying the probabilistic transition function P. States in $S_\circledcirc$ are terminating and have no transitions. As for DTMCs, paths are (finite or infinite) sequences of connected states. The sets of all infinite and finite paths starting in state s are denoted by $Path_s$ and $Path_s^{fin}$, respectively.

Strategy. A *strategy* for player $i \in \Pi$ is a function $\sigma_i : S^* S_i \to Dist(\alpha)$, where $Dist(\alpha)$ is the set of discrete probability distributions over α. Informally, the strategy resolves the choice of actions for the player during their turns, based on the previous history of the STG. A strategy σ_i is *memoryless* if for any $\pi, \pi' \in S^*$ and $s \in S_i$, $\sigma_i(\pi s) = \sigma_i(\pi' s) = \sigma_i(s)$. A strategy is *deterministic* if it always selects actions with probability 1, and *randomised* otherwise. The set of all strategies for player i is denoted Σ_i. The strategies for both players form a *strategy profile* $\sigma = \sigma_A, \sigma_D$. A probability measure Pr_s^σ under a strategy profile σ is defined in the standard way, similarly to DTMC.

Given a strategy profile $\sigma = \sigma_A, \sigma_D$, the resulting behaviour of the STG is represented by the induced DTMC, from which we can obtain a probability measure Pr_s^σ over $Path_s$. If the strategies are both memoryless, the induced DTMC has the same state space S as the STG.

Definition 5.3 (Induced DTMC) Let $\mathcal{G} = (\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_\circledcirc), P, T, AP, L)$ be an STG and σ_A, σ_D memoryless strategies on $\mathcal{G}$. The DTMC

$\mathcal{D}_\sigma = (S, P_\sigma, s_0, AP, L)$ induced by strategy profile $\sigma = \sigma_A, \sigma_D$ on $\mathcal{G}$ has P_σ as probabilistic transition function, defined by

$$P_\sigma(s,s') = \begin{cases} 1 & \text{if} \quad s \in S_A \wedge T(s,\sigma_A(s)) = s' \\ 1 & \text{if} \quad s \in S_D \wedge T(s,\sigma_D(s)) = s' \\ P(s,s') & \text{if} \quad s \in S_P \\ 1 & \text{if} \quad s \in S_\circledcirc \wedge s = s' \\ 0 & \textit{otherwise} \end{cases}$$

Like for DTMCs, we also consider reward structures. For an STG, these take the form $r_A : S_A \times \alpha \to \mathbb{Q}_{\geq 0}$ and $r_D : S_D \times \alpha \to \mathbb{Q}_{\geq 0}$, annotating the transitions controlled by the attacker and defender, respectively.

5.2 Probabilistic Model Checking

Probabilistic model checking provides automated analysis of quantitative properties, formally specified in temporal logic, against various probabilistic models, including DTMCs and STGs.

For expressing the properties of DTMCs Probabilistic Computation Tree Logic (PCTL) and its extensions with cost operator are used, while Probabilistic Alternating-Time Temporal Logic with Rewards (rPATL) is used as specification language for STGs.

In this section, we provide a brief overview of PCTL and its extension with rewards, and rPATL, while omitting the respective model checking algorithm. We refer the reader to [HJ94, KNP07, CFK+13a] for a more in-depth coverage of the topic.

5.2.1 PCTL and Model Checking DTMCs

Syntax. Probabilistic Computation Tree Logic is a branching-time temporal logic, based on the temporal logic CTL.

Definition 5.4 (PCTL Syntax) The syntax of PCTL is defined as follows:

$$\begin{array}{lcl} \phi & ::= & \textit{true} \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie p}(\psi) \\ \psi & ::= & X\phi \mid \phi_1 U \phi_2 \end{array}$$

where $a \in AP$ is an atomic proposition, $\bowtie \in \{\geq, >, \leq, <\}$, and $p \in \mathbb{Q} \cap [0,1]$.

Note that atomic proposition a must be taken from the set AP of a probabilistic model for which the formula is specified.

A formula defined in PCTL can be either a state formula ϕ evaluated over states, or a path formula ψ evaluated over paths. State formulae are used to express the properties of the model, while path formulae are used only as the

parameter of the *probabilistic operator* P. Intuitively, a state s satisfies $P_{\bowtie p}(\psi)$ if the probability that the formula ψ holds on the paths from s satisfies $\bowtie p$.

Path formulae are constructed with the operators *next* and *until*, denoted by X and U, respectively. The formula $X\phi$ is true on a path π if ϕ is true on the second state $\pi[1]$ of the path. The formula $\phi_1 U \phi_2$ is true on a path π if ϕ_2 is true on the j-th state $\pi[j-1]$ of the path and ϕ_1 is true up until the j-th state.

A number of additional operators can be derived based on the syntax of PCTL. In particular, well-known logical equivalences such as:

$$\mathit{false} \equiv \neg \mathit{true}$$

$$\phi_1 \vee \phi_2 \equiv \neg(\neg\phi_1 \wedge \neg\phi_2)$$

Moreover, the path operator U allows to derive the new path operator *eventually*, denoted by F, as follows:

$$F\phi \equiv \mathit{true}\ U\ \phi$$

Intuitively, the formula $F\phi$ means that ϕ will be eventually satisfied.

Semantics. For a DTMC $\mathcal{D}$, $s \models \phi$ denotes that a state $s \in S$ satisfies the state formula ϕ. Similarly, for a path π and a path formula ψ, $\pi \models \psi$ states that π satisfies ψ.

Definition 5.5 (PCTL Semantics) Let $\mathcal{D} = (S, s_0, P, AP, L)$ be a DTMC. For a state $s \in S$ the satisfaction relation $\models$ of PCTL is defined inductively by:

$$\begin{array}{lll}
s \models \mathit{true} & & \forall s \in S \\
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\phi & \text{iff} & s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models P_{\bowtie p}(\psi) & \text{iff} & Pr_s(\psi) \bowtie p
\end{array}$$

where $Pr_s(\psi) = Pr_s(\{\pi \in Path_s \mid \pi \models \psi\})$, and for a path π in $\mathcal{D}$:

$$\begin{array}{lll}
\pi \models X\phi & \text{iff} & \pi[1] \models \phi \\
\pi \models \phi_1 U \phi_2 & \text{iff} & \exists j \geq 0 : \pi[j] \models \phi_2 \wedge (0 \leq k < j : \pi[k] \models \phi_1)
\end{array}$$

Extension of PCTL with rewards. The logic PCTL is extended with the reward operator E, which allows to specify reward-related properties such as "is the expected reward of a successful attack at most 500?".

Definition 5.6 (rPCTL Syntax) The syntax of rPCTL is defined as follows:

$$\begin{array}{lcl} \phi & ::= & \mathit{true} \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie p}(\psi) \mid E^r_{\bowtie x}(F\phi) \\ \psi & ::= & X\phi \mid \phi_1 U \phi_2 \end{array}$$

where $x \in \mathbb{Q}_{\geq 0}$ and $r : S \times S \to \mathbb{Q}_{\geq 0}$ is a reward structure.

The *expected reward operator* E is used to evaluate the expected cost of reaching a state that satisfies ϕ. Intuitively, a state s satisfies $E^r_{\bowtie x}(F\phi)$ if the overall expected cumulative reward before reaching a state satisfying ϕ fulfils $\bowtie x$.

For $\mathcal{D} = (S, s_0, P, AP, L)$ and a state $s \in S$ the semantics of the expected reward operator is defined as follows:

$$s \models E^r_{\bowtie x}(F\phi) \text{ iff } Exp_s(Z^r_{F\phi}) \bowtie x$$

where $Exp_s(Z^r_{F\phi})$ denotes the expectation of the random variable $Z^r_{F\phi} : Path_s \to \mathbb{Q}_{\geq 0}$ with respect to the probability measure Pr^σ_s, and for a path $\pi \in Path_s$

$$Z^r_{F\phi}(\pi) = \begin{cases} \infty & \text{if } \pi[i] \not\models \phi \text{ for all } i \in \mathbb{N} \\ \sum_{i=0}^{min\{j \mid \pi[j] \models \phi\}-1} r(\pi[i]) & \text{otherwise} \end{cases}$$

Quantitative extension of PCTL. PCTL allows to verify whether or not the probability with which a path formula is satisfied meets the bound p. However, it is possible also to compute the actual probability of a path formula being satisfied, since the model checking algorithm computes the actual probability first and then compares it with the bound. Formally, we will write $P_{=?}(\psi)$. The same holds for the expected reward operator, where we can compute the actual expected value with the formula $E^r_{=?}(F\phi)$.

5.2.2 rPATL and Model Checking Games

Syntax. Alternating-Time Temporal Logic with Rewards is an extension of Alternating Temporal Logic (ATL). It combines the logic rPCTL with the temporal logic ATL that allows to express probabilistic properties for given coalitions of players.

Definition 5.7 (rPATL Syntax) The syntax of rPATL is:

$$\begin{array}{l} \phi ::= \mathit{true} \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \langle\langle C \rangle\rangle P_{\bowtie p}(\psi) \mid \langle\langle C \rangle\rangle E^r_{\bowtie x}(F^*\phi) \\ \psi ::= X\phi \mid \phi_1 U \phi_2 \end{array}$$

where $a \in AP$ is an atomic proposition, $C \subseteq \Pi$ is a set of players, $\bowtie \in \{\geq, >, \leq, <\}$, $p \in \mathbb{Q} \cap [0,1]$, $x \in \mathbb{Q}_{\geq 0}$, r is a reward structure and $* \in \{0, \infty, c\}$.

The coalition operator $\langle\langle C\rangle\rangle\phi$ means that the coalition of the players $C \subseteq \Pi$ has a strategy to satisfy ϕ, regardless of the strategies of the players in $\Pi \backslash C$. In our setting, C is either $\{A\}$ or $\{D\}$ (in our case this is an exclusive or).

A formula can be either a state formula ϕ or a path formula ψ. Atomic propositions and logical connectives have the same meaning as in rPCTL. Similarly to rPCTL, the operators for building path formulae are next and until, denoted by X and U. The operator eventually is defined as in PCTL by $F\phi \equiv \mathit{true}\, U\, \phi$. Intuitively, the formula $\langle\!\langle C\rangle\!\rangle P_{\bowtie p}(\psi)$ means that the coalition of players C has a strategy which can ensure that the probability of ψ being satisfied is in the interval specified by $\bowtie p$, regardless of the strategies of the other players. An example formula is $\langle\langle A\rangle\rangle P_{\geq 0.1}(F\, \mathit{success})$, which is true if the attacker has a strategy that guarantees to reach a state labelled with *success* (e.g., denoting a successful attack), regardless of the strategy of the defender.

The operator E allows to reason about expected cumulative reward until a state satisfying ϕ is reached. The parameter $*$ specifies the result if ϕ is not reached. In the developments of Ch. 6 we fix $* = \infty$. Intuitively, the formula $\langle\!\langle C\rangle\!\rangle E^r_{\bowtie x}(F^*\phi)$ means that the coalition of players C has a strategy which can ensure that the expected reward before reaching a state satisfying ϕ is in the interval specified by $\bowtie x$, regardless of the strategies of the other players.

Semantics. As with PCTL, $s \models \phi$ indicates that the state s satisfies the state formula ϕ, and $\pi \models \psi$ indicates that the path π satisfies the path formula ψ.

Definition 5.8 (rPATL Semantics) Given a game $\mathcal{G}$ and a state s the satisfaction relation $\models$ of rPATL is defined inductively by:

$$
\begin{array}{lll}
s \models \mathit{true} & & \forall s \in S \\
s \models a & \text{iff} & a \in L(s) \\
s \models \neg\phi & \text{iff} & s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\
s \models \langle\langle C\rangle\rangle P_{\bowtie p}(\psi) & \text{iff} & \exists \sigma_C \in \Sigma_C \text{ such that } \forall \sigma_{\Pi\backslash C} \in \Sigma_{\Pi\backslash C} : \\
 & & Pr_s^{\sigma_C,\sigma_{\Pi\backslash C}}(\psi) \bowtie p \\
s \models \langle\langle C\rangle\rangle E^r_{\bowtie x}(F^*\phi) & \text{iff} & \exists \sigma_C \in \Sigma_C \text{ such that } \forall \sigma_{\Pi\backslash C} \in \Sigma_{\Pi\backslash C} : \\
 & & Exp_s^{\sigma_C,\sigma_{\Pi\backslash C}}(rew(r,*,Sat(\phi))) \bowtie x
\end{array}
$$

where $Pr_s^{\sigma_C,\sigma_{\Pi\backslash C}}(\psi) = Pr_s^{\sigma_C,\sigma_{\Pi\backslash C}}\{\pi \in Path_s \mid \pi \models \psi\}$ and $Exp_s^{\sigma_C,\sigma_{\Pi\backslash C}}(rew(r,*,Sat(\phi)))$ denotes the expectation of the reward function $rew(r,*,T) : Path_s \to \mathbb{Q}_{\geq 0}$ under schedulers $\sigma_C, \sigma_{\Pi\backslash C}$ with respect to the probability measure $Pr_s^{\sigma_C,\sigma_{\Pi\backslash \widehat{C}}}$, and for a path π:

$$
rew(r,*,T)(\pi) = \begin{cases} g(*) & \text{if } \pi[i] \not\models \phi \text{ for all } i \in \mathbb{N} \\ \sum_{i=0}^{min\{j \mid \pi[j] \models \phi\}-1} r(\pi[i]) & \text{otherwise} \end{cases}
$$

where $g(*) = *$ if $* \in \{0, \infty\}$ and $g(*) = \sum_{i \in \mathbb{N}} r(\pi[i])$ if $* = c$.

For a path π, the satisfaction relation is defined by:

$$\begin{aligned}
\pi &\models X\phi && \text{iff} && \pi[1] \models \phi \\
\pi &\models \phi_1 U^k \phi_2 && \text{iff} && \exists j \leq k : \pi[j] \models \phi_2 \wedge (0 \leq i < j : \pi[i] \models \phi_1) \\
\pi &\models \phi_1 U \phi_2 && \text{iff} && \exists j \geq 0 : \pi[j] \models \phi_2 \wedge (0 \leq i < j : \pi[i] \models \phi_1)
\end{aligned}$$

Quantitative extension of rPATL. The rPATL operators defined above are qualitative ones, meaning that the operators verify whether or not the given bound is satisfied. The logic can be extended with quantitative operators that return numerical values. For the probabilistic computation the formulae $\langle\langle C \rangle\rangle P_{min=?}(\psi)$ and $\langle\langle C \rangle\rangle P_{max=?}(\psi)$ compute the minimum and the maximum probabilities of ψ being satisfied, respectively. Similarly, the formulae $\langle\langle C \rangle\rangle E^r_{min=?}(F^* \phi)$ and $\langle\langle C \rangle\rangle E^r_{max=?}(F^* \phi)$ compute the minimum and the maximum expected reward, respectively.

5.3 PRISM and PRISM-Games

The probabilistic symbolic model checker PRISM [KNP11] is a tool for formal modelling and analysis of systems with probabilistic and nondeterministic behaviour.

PRISM provides support for various Markov models such as Discrete- and Continues-Time Markov Chains and Markov Decision Processes. The model checking techniques for such models are implemented in PRISM, allowing to analyse a wide range of qualitative and quantitative properties. The language used for specifying properties includes various temporal logics such as PCTL, CSL, LTL, as well as extensions for quantitative specifications and rewards. The tool has been widely used to analyse systems from many application domains, e.g., security protocols, wireless communication protocols, randomised distributed algorithms and biological systems.

The model checker PRISM has been extended to support the modelling and analysis of systems with competitive or cooperative behaviour. PRISM-games [KPW16] is a model checking tool for verification and strategy synthesis for stochastic multi-player games. The tool supports the probabilistic model checking algorithm for the logic rPATL.

PRISM-games supports verification of multi-objective properties expressed as a Boolean combination of reward-based objectives. Moreover, the tool can compute and display a Pareto curve in case of conflicting objectives.

The interested reader is referred to [KNP11, KPW16, PRI] for a more in-depth coverage of PRISM and PRISM-games.

Chapter 6

Quantitative Verification and Synthesis of Attack-Defence Scenarios

So far we have studied the evaluation of attack-defence scenarios through attack-defence trees. We shall now focus on the competitive behaviour of the attacker and the defender, and explore the relation between attack-defence trees and Stochastic Two-player Games.

In this chapter, we propose a novel framework for the formal analysis of quantitative properties of attack-defence scenarios using an extension of attack-defence trees. We allow the trees to incorporate information about the temporal ordering of some actions or sub-goals. We then propose a novel class of attack/defence strategies that incorporate *dependencies*, allowing decisions about which actions are taken by the attacker or defender to be based on which actions have been taken earlier and the outcomes of those actions.

Formally, these strategies are defined as *decision trees*. For an attack-defence scenario, modelled as an attack-defence tree, and specific strategies for both the attacker and defender, we give a semantics defining the resulting behaviour. We allow attack-defence trees to be annotated with the probability of success or failure of individual basic actions, so the semantics takes the form of a Markov model, more precisely a Discrete-Time Markov Chain.

In order to formally analyse attack-defence scenarios modelled in this way, we employ quantitative verification techniques, in particular probabilistic model checking.

For attack-defence trees, it is natural to model the interactions between attacker and defender as a two-player game. So, in our setting, where quantitative and probabilistic aspects are essential, we use STGs, building upon the probabilistic model checking techniques for stochastic games, and implemented in the PRISM-games model checking tool.

This approach uses the temporal logic rPATL, introduced in Ch. 5. rPATL allows us to explicitly reason about the strategies available to the competing players, and provides a variety of operators to specify quantitative properties of these strategies relating to the probability of certain events occurring, or a cost or reward measure associated with them.

Probabilistic model checking of rPATL on stochastic games allows us to take two distinct approaches to the analysis of an attack-defence scenario: we can *verify* security properties of them (e.g., "whatever the attacker does, the defender can always guarantee that the probability of a successful attack is at most 0.001"); or we can *synthesise* strategies for a player with respect to some goal (e.g., "what strategy for the attacker maximises the probability of a successful attack, regardless of the actions employed by the defender?"). rPATL provides expressiveness which goes beyond the standard queries typically used on attack-defence scenarios [KMS12].

In order to use these game-theoretic verification techniques within our framework, we define a translation from attack-defence trees to STGs. This model captures the set of all strategies available to the attacker and defender players, and the resulting (Markov Chain) semantics induced by each pair of such strategies. Using probabilistic model checking, we can either verify a security property on this game model, or synthesise a strategy achieving or optimising a desired property. In the latter case, a player strategy generated from the stochastic game as a result is then converted into a decision tree for either the attacker or defender.

Finally, we discuss the evaluation of multi-objective properties in this setting. We use an extension of the basic rPATL-based approach for *multi-objective probabilistic model checking* of stochastic games [CFK+13b]. This allows us to analyse multiple, conflicting objectives such as probability and expected cost (e.g., "what strategy minimises the expected cost incurred by the attacker, whilst guaranteeing a successful attack with probability at least 0.5?"), and also to compute the Pareto curve associated with the objectives.

We implement our approach and illustrate it on the example of a Radio-Frequency Identification (RFID) warehouse goods management system, focusing on a fragment of the overall system that includes also the example of Ch. 4 [BKMS12].

In Sect. 6.1 we present our formalism for attack-defence trees and strategies, as well as their semantics. In Sect. 6.2 we describe our proposed translation from attack-defence trees to STGs, and the evaluation of single-objective properties. The evaluation of STGs with multiple objectives is discussed in Sec. 6.3. The

Table 6.1: The syntax of attack-defence trees and the type system for defining well-formed trees.

$$t \quad ::= \quad nt \mid \&_{\overrightarrow{\wedge}}(t_1, t_2) \mid \&_{\overrightarrow{\vee}}(t_1, t_2)$$

$$nt \quad ::= \quad a \mid \&_{\wedge}(nt_1, nt_2) \mid \&_{\vee}(nt_1, nt_2) \mid \&_{\sim} nt \mid \&_{\texttt{true}} \mid \&_{\texttt{false}}$$

$$\vdash a : A \text{ if } a \in Act_A \qquad \vdash a : D \text{ if } a \in Act_D$$

$$\frac{\vdash t_1 : \tau \quad \vdash t_2 : \tau}{\vdash \&_{\overrightarrow{\wedge}}(t_1, t_2) : \tau} \qquad \frac{\vdash t_1 : \tau \quad \vdash t_2 : \tau}{\vdash \&_{\overrightarrow{\vee}}(t_1, t_2) : \tau}$$

$$\frac{\vdash nt_1 : \tau \quad \vdash nt_2 : \tau}{\vdash \&_{\wedge}(nt_1, nt_2) : \tau} \qquad \frac{\vdash nt_1 : \tau \quad \vdash nt_2 : \tau}{\vdash \&_{\vee}(nt_1, nt_2) : \tau} \qquad \frac{\vdash nt : \tau}{\vdash \&_{\sim} nt : \tau'} \; \tau' = \tau^{-1}$$

$$\vdash \&_{\texttt{true}} : \tau \qquad \vdash \&_{\texttt{false}} : \tau$$

results of the evaluation are discussed on the case study in Sect. 6.4. This chapter is based on [ANP16].

6.1 Attack-Defence Trees with Sequential Operators

In this section, we present the key ingredients of our formalism: attack-defence trees, to represent attack-defence scenarios, and strategies (in the form of decision graphs), to represent the behaviour of the attacker and defender. The formalism for attack-defence tree is the extension of our formalism presented in Ch. 4. We start by defining the syntax and terminology for each of these, in Sect. 6.1.1 and 6.1.2, respectively. Then, in Sect. 6.1.3, we describe their formal semantics.

6.1.1 Attack-Defence Trees

The abstract syntax of an attack-defence tree is presented in the top part of Table 6.1, organised into rules for a tree t and a non-sequential tree nt. A tree is either a leaf or the application of a tree operator to one or two sub-trees. A leaf a is a basic action of either the attacker or the defender. We denote the attacker's and defender's sets of basic actions by Act_A and Act_D, respectively. We assume these are disjoint and write $Act = Act_A \uplus Act_D$ for the set of all basic actions.

Tree operators include conjunction, disjunction and changing player, already

defined in Ch. 4, as well as two additional operators: *sequential* conjunction $\&_{\overrightarrow{\wedge}}$ and *sequential* disjunction $\&_{\overrightarrow{\vee}}$. In order to simplify the technical developments, we require that the sequential operators only occur *above* the conjunction and disjunction operators in a tree. Thus, we disallow trees such as $\&_{\wedge}(\&_{\overrightarrow{\wedge}}(a,b),c)$ for basic actions a,b,c. This is imposed by the division of the syntax into trees t and *non-sequential* trees nt.

The sequential variants additionally impose an ordering on the sub-trees. Sequential conjunction $t = \&_{\overrightarrow{\wedge}}(t_1,t_2)$ requires that the goals of both t_1 and t_2 are achieved and that the former is performed before the latter. Sequential disjunction $t = \&_{\overrightarrow{\vee}}(t_1,t_2)$ similarly requires t_1 to be performed before t_2 and that the goal of at least one sub-tree is achieved. Intuitively, in a sequential disjunction it only makes sense to *attempt* t_2 after failing in *achieving* t_1.

As in Ch. 4, a simple type system enforces the association between players and nodes, as shown in the second section of Table. 6.1.

Phases. In an attack-defence tree t of the form described above, we associate each maximal non-sequential sub-tree with a *phase*. We say that the maximal non-sequential sub-trees $nt_1,\cdots,nt_n$ divide the tree t into phases $\mathfrak{p}_1,\cdots,\mathfrak{p}_n$, where any two non-sequential sub-trees are connected with a sequential operator. Thus, the number of phases in a tree is one more than the number of sequential operators. We denote by *Phases* the set of all phases in a tree. We indicate by $Act_{\mathfrak{p}_i,A}$ and $Act_{\mathfrak{p}_i,D}$, respectively, the set of attacker and defender basic actions in phase $\mathfrak{p}_i$ (non-sequential tree nt_i), and by $Act_{\mathfrak{p}_i}$ the set of all basic actions in phase $\mathfrak{p}_i$, i.e., $Act_{\mathfrak{p}_i} = Act_{\mathfrak{p}_i,A} \uplus Act_{\mathfrak{p}_i,D}$.

Example. Let us introduce an example that we will develop throughout this chapter. We consider a modified version of a simple scenario borrowed from [KPS14b], where an attacker wants to infect a computer with a virus. In order to do so, the attacker needs to put the virus file on the system and only after that execute it. The attacker can transmit the virus either by sending an e-mail with an attachment or by distributing a USB stick to a user of the system. The defender, on the other hand, can try to prevent the attack by running an anti-virus program. Once the virus file is on the computer, the attacker can execute it either directly or through a third person. The defender can counteract this by restoring the registry The corresponding attack-defence tree is shown in Figure 6.1, where we label leaves for ease of reference. The syntactic term corresponding to the full tree is:

$$t = \&_{\overrightarrow{\wedge}}(\&_{\overrightarrow{\wedge}}(\&_{\vee}(se,usb),\&_{\sim}rav),\&_{\wedge}(ef,\&_{\sim}rr))$$

The tree t has three non-sequential sub-trees: $t_1 = \&_{\vee}(se,usb)$, $t_2 = \&_{\sim}rav$, and $t_3 = \&_{\wedge}(ef,\&_{\sim}rr)$ and thus three phases.

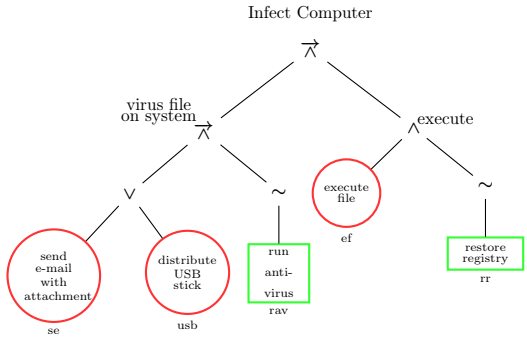


Figure 6.1: An example of an attack-defence tree for infecting a computer.

6.1.2 Strategies as Decision Trees

As we have seen in Ch. 4 for the Boolean setting, in the standard model of attack-defence trees a possible attack (or defence) is a *set* of basic actions for one player which will be performed. Given an attack and a defence, the tree defines the success or failure of these according to the way that the basic actions are combined within the attack-defence tree (e.g., if the root node belongs to the attacker, the attack is successful if the tree evaluates to true).

Here, we take a different approach, proposing a more powerful class of *strategies* for attackers and defenders which can incorporate *dependencies* on events that happened previously when deciding which basic action to perform. In particular, a strategy's choice can depend on the attempt, success or failure of another basic action from an earlier phase. We represent these strategies, for either attacker or defender, by means of *decision trees*, which illustrate the relation between phases in an intuitive way.

The abstract syntax of a decision tree d is presented in Table 6.2. We assume that this defines a strategy for an attack-defence tree with n phases $\mathfrak{p}_1, \cdots, \mathfrak{p}_n$ and, for convenience, we add a "dummy" phase $\mathfrak{p}_{n+1}$ denoting the end of the strategy. In the syntax, d_i^τ represents a strategy (for a sub-tree t_i) for player τ (where $\tau = A, D$) which starts in phase $\mathfrak{p}_i$. So the root node of a decision tree is of the form d_1^τ.

In phase $\mathfrak{p}_i$ (for $1 \leq i \leq n$), a decision tree can be either: (i) an *action node* $B.d_{i+1}^\tau$, indicating that player τ performs the (possibly empty) set of basic actions $B \subseteq Act_{\mathfrak{p}_i,\tau}$ in that phase and then proceeds to sub-tree d_{i+1}^τ; or (ii) a *decision node* of the form $\mathit{if}(c_i^\tau, d_i'^\tau, d_i''^\tau)$, which branches based on the condition c_i^τ. Once n phases have passed, the decision tree moves to a *stop* node, representing the end.

Decision nodes represent conditional dependencies: deciding whether to perform some basic actions or to move to another phase can be conditional on events that have already occurred. For instance, in phase $\mathfrak{p}_i$, a player may wish to only

Table 6.2: The syntax of a decision tree.

d_i^τ	::=	$B.d_{i+1}^\tau$ where $B \subseteq Act_{\mathfrak{p}_i,\tau}$	c_i^τ	::=	$a?$ if $\tau = A$ and $a \in Act_{\mathfrak{p}_i,D}$
d_i^τ	::=	$if(c_i^\tau, d_i'^\tau, d_i''^\tau)$	c_i^τ	::=	$\mathfrak{p}_j?$ if $1 \leq j < i$
d_{n+1}^τ	::=	$stop$	c_i^τ	::=	$c_i'^\tau \wedge c_i''^\tau$
			c_i^τ	::=	$c_i'^\tau \vee c_i''^\tau$
			c_i^τ	::=	$\neg c_i'^\tau$

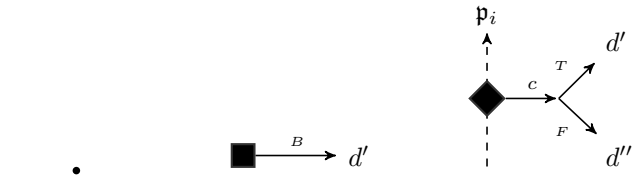
perform an action in the case of a failed attempt at performing some action in phase $\mathfrak{p}_{i-1}$. A decision node consists of a condition c_i^τ and two possible sub-trees, where execution of the strategy moves to the first sub-tree $d_i'^\tau$ if the condition is true, or the second sub-tree $d_i''^\tau$ otherwise.

Conditions are expressed as Boolean expressions over atomic conditions of two types, $\mathfrak{p}_j?$ or $a?$. A condition of the form $\mathfrak{p}_j?$, in a phase $\mathfrak{p}_i$ decision tree node (where $j < i$), asks about the success of an earlier phase. The success of a phase is determined by an evaluation of the corresponding node of the sub-tree (more precisely, the root node of the corresponding maximal non-sequential tree), which we will define more precisely in the next section.

In the case of a decision tree for an attacker strategy (i.e., where $\tau = A$), we also allow conditions $a?$ that ask whether some defender action $a \in Act_{\mathfrak{p}_i,D}$ was performed within the current phase $\mathfrak{p}_i$ (note that we ask whether it was attempted, not whether it was successful, since the latter is not yet known). We do *not* allow the reverse: where defenders ask about attacker action's performed in the same phase. In our work, we assume that the attacker has slightly more power/information than the defender in this respect, favouring a conservative approach to verifying the absence of attacks.

The intuition behind the strategy for the players is as follows. In each phase $\mathfrak{p}_i$ (for $1 \leq i \leq n$) the attacker can ask about the success of the earlier phases $\mathfrak{p}_j$ (where $j < i$), and whether some defender action $a \in Act_{\mathfrak{p}_i,D}$ was attempted in the same phase. Based on the outcome of such queries the attacker chooses the set of actions $B \subseteq Act_{\mathfrak{p}_i,A}$ to be performed in $\mathfrak{p}_i$. On the contrary, in each phase $\mathfrak{p}_i$ (for $1 \leq i \leq n$) the defender can ask only about the success of the earlier phases $\mathfrak{p}_j$ (where $j < i$), and based on the result chooses the set of basic actions $B \in Act_{\mathfrak{p}_i,D}$ to be performed in phase $\mathfrak{p}_i$. We assume that in each phase the players always perform a set of basic actions. In case the player has no action in the current phase we denote it with $B = \emptyset$. The phase $\mathfrak{p}_i$ ends when the players performed the set of actions and the strategy moves to the next phase $\mathfrak{p}_{i+1}$.

Graphical representation. The graphical representation of each node in a strategy is shown in Figure 6.2. A stop node is represented by a black dot,



(a) stop node (b) action node (c) decision node

Figure 6.2: Graphical representation of decision tree nodes.

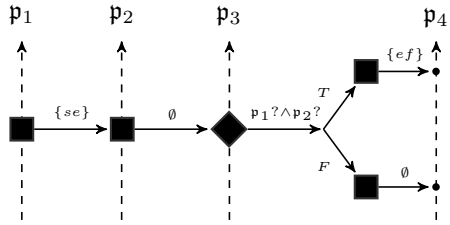


Figure 6.3: An attacker strategy for the attack-defence tree of Fig 6.1.

and occurs only as a leaf at the end of the tree (in the last phase $\mathfrak{p}_{n+1}$). An action node $B.d'$ is represented by a black square with an outgoing edge to the successor node d' labelled with the set of performed actions B. A conditional node $\mathit{if}(c, d', d'')$ is drawn as a black diamond. It has an outgoing transition labelled with the condition c and two transitions, labelled with "T" (true) and "F" (false), corresponding to the "then" and "else" branches of the "if" statement. In the rightmost node in Figure 6.2 (the decision node), we also illustrate the use of vertical dashed lines to mark the boundaries of phases.

Example. Figure 6.3 illustrates a possible attacker strategy for the tree t, displayed in Figure 6.1. The attack strategy is represented by the decision tree with the following syntactic term:

$$d_1^A = \{se\}.\emptyset.\mathit{if}(\mathfrak{p}_1? \wedge \mathfrak{p}_2?, \{ef\}.\mathit{stop}, \emptyset.\mathit{stop})$$

According to the strategy d_1^A, in the first phase, the attacker sends an e-mail with an attachment. As there are no attacker actions in the second phase, the attacker does nothing and moves to the third phase. Before performing an action in the third phase, the attacker checks the success of the previous phases $\mathfrak{p}_1, \mathfrak{p}_2$. In case of success, i.e., if the attacker successfully sent an e-mail in phase $\mathfrak{p}_1$ and the defender did nothing or failed to run the anti-virus program in phase $\mathfrak{p}_2$, the attacker executes the file; otherwise, he/she does nothing.

Figure 6.4 illustrates a possible defender strategy for the same tree. The difference for a defender strategy with respect to an attacker strategy is that in each phase the defender knows about the success of the previous phases but

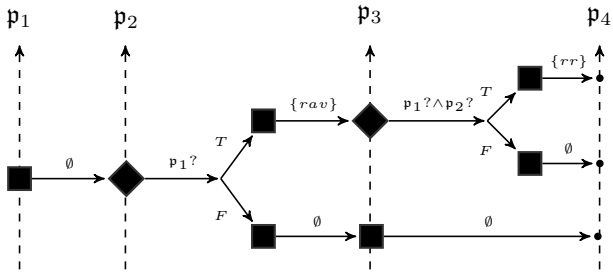


Figure 6.4: A defender strategy for the attack-defence tree of Fig 6.1.

not about actions attempted by attacker in that phase. The defender strategy is represented by the following term:

$$d_1^D = \emptyset.\mathit{if}(\mathfrak{p}_1?, \{rav\}.\mathit{if}(\mathfrak{p}_1? \wedge \mathfrak{p}_2?, \{rr\}.\mathit{stop}, \emptyset.\mathit{stop}), \emptyset.\emptyset.\mathit{stop})$$

In the first phase, the defender does not have any actions to perform. In the second, it runs the anti-virus program if $\mathfrak{p}_1$ was successful, that is, if the attacker succeeded in sending a virus over e-mail; otherwise, he/she does nothing. In the third phase, depending on the success of the previous phases, the defender either restores the registry or does nothing.

6.1.3 Semantics of Attack-Defence Trees

So far we have presented the syntax of attack-defence trees and explained how strategies can be represented by means of decision trees. Given strategies for both players, the attack-defence scenario described by an attack-defence tree is determined, meaning that the value at the root can be computed. The computation depends on the type of semantics chosen for the basic actions, e.g., Boolean, probabilistic, etc. Moreover, fixed a semantics, different evaluation approaches can be exploited in the computation. For example, a bottom-up evaluation (without sequential operator) for the Boolean and probabilistic case can be found in [KMRS10] and for multi-objective Pareto analyses in Ch. 4 and in [AN15].

In the following, we define, given an attack-defence tree and strategies for both players, a probabilistic semantics of the resulting attack-defence scenario, and show how to evaluate it. Since our attack-defence trees incorporate temporal ordering (through the sequential operators) and our strategies involve dependencies on earlier events, we define our semantics as a state-based model capturing the possible sequences of events that can unfold. As discussed below, basic actions can fail stochastically, so the semantics is in fact represented as a (Discrete-Time) Markov Chain. We will also annotate this model with costs or rewards (e.g., to represent the cost of performing basic actions) and, later, will show how the semantic model can be analysed against a range of quantitative measures using probabilistic model checking. This will allow us to check for

the existence (or absence) of particular strategies, for example: "what is the maximum probability of a successful attack?".

Probabilities and costs. We associate each basic action a with a success probability $p(a)$, $p : Act \to [0, 1]$, that it is achieved successfully if a is performed. Moreover, we assume that each basic action a has a cost $c(a)$ of performing it, $c : Act \to \mathbb{Q}_{\geq 0}$. Note that, the probabilities of individual basic actions are all assumed to be independent of each other. Similarly, the costs incurred by each action and by each player are all independent. Moreover, cost-related questions are player-dependent, meaning that the scenario is evaluated from a given player's perspective. For instance, for computing the minimum cost of an attack we need only the cost of the attacker's actions and do not require the cost of the defender's actions. Thus, in our model we will not consider the cost of the defender actions while evaluating the attacker cost, and vice versa. The relation between actions (and the resulting impact this has in terms of, e.g., the probability of a successful attack) is captured by the structure of the attack tree. In other words, the assumptions retrace those from Ch. 4.

Semantics. We define the DTMC semantics for attack-defence tree t, with n phases, represented by non-sequential trees $t = t_1, \ldots, t_n$, and decision trees d_1^D, d_1^A. The semantics is given by the function build in Table 6.3. It constructs the DTMC, which takes the form of a tree, recursively, each call returning a new root state.

It is worthwhile noticing that the order in which players perform actions is not determined in an attack-defence tree. On the contrary, in the DTMC, this needs to be made explicit. Since we assume, as discussed earlier, that the attacker knows the actions attempted by the defender in the current phase, in the DTMC semantics, the defender is the first to move in each phase.

Throughout the evaluation of the function we assume to have an attack-defence tree t with n phases, $t = t_1, \cdots, t_n$, and the corresponding probability function p and cost function c as a global parameters. The build function operates recursively over the two decision trees: each call is of the form $\mathsf{build}[t, p, c](d^\tau, d^{\tau^{-1}}, \tau, Succ_{\mathfrak{p}}, Done_i, Succ_i)$, where the parameters have the following meaning. The first two parameters correspond to a player's decision tree d_i^τ and the opposite player's decision tree $d_i^{\tau^{-1}}$ that still have to be evaluated. The third parameter represents the next player to move, and is used to identify the end of each phase, i.e, it is the end of the phase if $\tau = A$. The remaining parameters record the phases that were successful ($Succ_{\mathfrak{p}}$), the set of actions attempted in the current phase ($Done_i$) and the ones that succeeded ($Succ_i$). At the top-level, the function is called as $\mathsf{build}[t, p, c](d_1^D, d_1^A, D, \emptyset, \emptyset, \emptyset)$. The function is structurally defined over decision trees as explained below.

If the decision tree is an *if*-clause, $d_i^\tau = if(c_i^\tau, d_i'^\tau, d_i''^\tau)$, we evaluate the condition c_i^τ over the success of the previous phases and, when $\tau = A$, also over the set

Table 6.3: The function build describing the semantics of an attack-defence tree as a DTMC.

$$\mathsf{build}[t,p,c](\mathit{if}(c_i^\tau, d_i'^\tau, d_i''^\tau), d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$$
$$\begin{cases} \mathsf{build}[t,p,c](d_i'^\tau, d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) & \text{if } [\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i) \\ \mathsf{build}[t,p,c](d_i''^\tau, d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) & \text{if } \neg[\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i) \end{cases}$$

$\mathsf{build}[t,p,c]((B \cup \{a\}).d_{i+1}^\tau, d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$ new state s with:
$L(s) = \{a\}, P(s,s') = p(a), P(s,s'') = 1-p(a)$ where:
$s' = \mathsf{build}[t,p,c](B.d_{i+1}^\tau, d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i \cup \{a\})$
$s'' = \mathsf{build}[t,p,c](B.d_{i+1}^\tau, d_i^{\tau^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i)$
and: $r(s,s') = r(s,s'') = c(a)$

$$\mathsf{build}[t,p,c](\emptyset.d_{i+1}^\tau, d_i^{\tau^{-1}}, D, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$$
$$\mathsf{build}[t,p,c](d_i^{\tau^{-1}}, d_{i+1}^\tau, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i)$$

$$\mathsf{build}[t,p,c](\emptyset.d_{i+1}^\tau, d_{i+1}^{\tau^{-1}}, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$$
$$\begin{cases} \mathsf{build}[t,p,c](d_{i+1}^{\tau^{-1}}, d_{i+1}^\tau, D, \mathit{Succ}_\mathfrak{p} \cup \{\mathfrak{p}_i\}, \emptyset, \emptyset) & \text{if } [\![t_i]\!](\mathit{Succ}_i) \\ \mathsf{build}[t,p,c](d_{i+1}^{\tau^{-1}}, d_{i+1}^\tau, D, \mathit{Succ}_\mathfrak{p}, \emptyset, \emptyset) & \text{if } \neg[\![t_i]\!](\mathit{Succ}_i) \end{cases}$$

$\mathsf{build}[t,p,c](\mathit{stop}, \mathit{stop}, A, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i, \mathit{Succ}_i) =$ new state s with:
$L(s) = \{\mathit{success}\}$ if $[\![t]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Succ}_i)$ and $\{\mathit{failure}\}$ otherwise

of actions attempted in the current phase. The evaluation $[\![c_i^\tau]\!](\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i)$ is a standard Boolean evaluation, where c_i^τ is a Boolean expression and $(\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i)$ give the truth values of the variables, where the variable is tt if it is in the set $(\mathit{Succ}_\mathfrak{p}, \mathit{Done}_i)$ and ff otherwise. If $\tau = D$ we can omit the component Done_i from the evaluation. The DTMC is constructed recursively, from either $d_i'^\tau$ or $d_i''^\tau$, depending on whether c_i^τ evaluates to true.

If the root of the decision tree is an action node containing action a, i.e., $d_i^\tau = (B \cup \{a\}).d_{i+1}^\tau$, we create a DTMC state labelled with a, with outgoing transitions corresponding to the success or failure of executing a (with probability $p(a)$ and $1-p(a)$). We also label the transitions with the cost $c(a)$. The successor states are constructed recursively, adding a to Done_i and, if appropriate, Succ_i. In case a executed successfully we call $\mathsf{build}[t,p,c](B.d_{i+1}^\tau, d^{\tau_i^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i \cup \{a\})$, otherwise we call $\mathsf{build}[t,p,c](B.d_{i+1}^\tau, d^{\tau_i^{-1}}, \tau, \mathit{Succ}_\mathfrak{p}, \mathit{Done}_i \cup \{a\}, \mathit{Succ}_i)$.

If the set of actions in the action node is empty, $d_i^\tau = \emptyset.d_{i+1}^\tau$, and the current player is D, it means that the defender does not have any more moves in phase i and we need to start exploring the attacker decision tree in the same

phase. Thus, we change the current player from D to A, and construct DTMC recursively from $d_i^{\tau^{-1}}$.

On the contrary, if the action set is empty and the current player is A, then we are at the end of the phase. Hence, we evaluate the success of phase i based on the set $Succ_i$, $[\![t_i]\!](Succ_i)$, where $[\![t_i]\!]$ is Boolean formula of which the non-sequential sub-tree t_i is a parse tree and the actions in the formula are tt if the actions are in the set $Succ_i$ and ff otherwise. If phase i was successful, we add $\mathfrak{p}_i$ to the set $Succ_{\mathfrak{p}}$. We start the new phase $\mathfrak{p}_{i+1}$ with player D to move next, and resetting the sets $Done_i$ and $Succ_i$ to empty.

Finally, if the decision trees for both players consist of the *stop* node, $d_i^{\tau} = stop, d_i^{\tau^{-1}} = stop$, and A is to move next, then we are at the end of both strategies. We create a final node in the DTMC and label it with the result of the evaluation of the tree t over the success of all phases, $[\![t]\!](Succ_{\mathfrak{p}})$.

Once we have obtained a DTMC, we can verify the properties of interest by means of probabilistic model checking. Below, we will see some examples of security properties verified on the DTMC corresponding to the tree given in Figure 6.1.

So far we assumed that the strategies for both players are given and focused on the values of the properties for fixed strategies. This leads to the following questions:

- how to obtain strategies (possibly in an automated way)?
- how to evaluate security properties over all possible strategies for both players?

A game semantics for attack-defence scenarios is the answer to both questions.

Example. Consider the example attack-defence tree given in Figure 6.1, the strategies for attacker and defender given in Figures 6.3 and 6.4, and the probability and cost values for basic actions listed in Table 6.4. Figure 6.5 shows the resulting DTMC semantics. We verify the following security properties: "Is the success probability of an attack greater than or equal to 0.005?" and "What is the success probability of an attack?". The first one is expressed in PCTL as the formula $P_{\geq 0.005}[F\, success]$ which evaluates to *true*, while the second one is expressed in PCTL as the formula $P_{=?}[F\, success]$ and the obtained result is 0.00675.

6.2 Game-based Modelling and Verification

In this part of the dissertation, we use probabilistic model checking techniques to evaluate attack-defence scenarios using the formalism proposed in the previous section. In particular, we aim at verifying whether certain types of attack are

Table 6.4: Probabilities and costs for the basic actions in the example.

Label	Name of the Node	Success probability	Cost
se	send e-mail with attachment	0.2	20
usb	distribute USB stick	0.6	80
rav	run anti-virus	0.7	70
ef	execute file	0.75	50
rr	restore registry	0.85	65

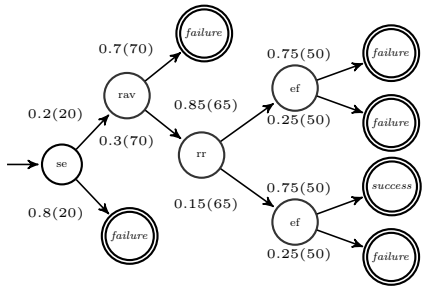


Figure 6.5: The DTMC for attack-defence tree from Figure 6.1 and decision trees d_1^A, d_1^D from Figures 6.3 and 6.4.

impossible, or to synthesise attack or defence strategies satisfying some formally specified property. The basic idea is to transform an attack-defence tree into a Stochastic Two-player Game, in which the players are the attacker and defender, and strategies (in the sense of the stochastic game) correspond to strategies represented by decision trees over an attack tree.

In this section, we explain the transformation of an attack tree to a game and describe how probabilistic model checking can be applied to answer the kinds of questions posed above. We also then explain how to extract decision tree strategies from the results of model checking.

6.2.1 From Attack-Defence Trees to Stochastic Games

Given an attack-defence tree t with n non-sequential sub-trees (phases) we transform the tree t to an STG in two steps. First we transform each sub-tree to a game and then combine the games by means of the sequential composition, mimicking the behaviour of a sequential operator connecting corresponding non-sequential sub-trees.

Before explaining the algorithm, it is worthwhile discussing the behaviour of the players in the trees. Each sub-tree represents the static behaviour of the players, i.e., each player makes a choice of their actions independently and the outcome of each action affects only the overall result of the sub-tree and not the other basic actions. Thus, in a game corresponding to a sub-tree, we consider the set of attempted actions for each player instead of one action at a time.

Algorithm 1 Transformation of non-sequential tree to STG.

Input: a non-sequential tree t_i with probabilistic function p and cost function c for basic actions, and $Act_{\mathfrak{p}_i,A}, Act_{\mathfrak{p}_i,D}, Act = Act_{\mathfrak{p}_i,A} \uplus Act_{\mathfrak{p}_i,D}$ sets
Output: STG $(\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_\circledcirc), P, T, AP, L)$

$\Pi \leftarrow \{A, D\}$; $\alpha \leftarrow 2^{Act}$; $AP \leftarrow Act \uplus \{success, failure\}$;
$P \leftarrow \emptyset$; $T \leftarrow \emptyset$; $F_1 \leftarrow \emptyset$; $F_2 \leftarrow \emptyset$;
$S_A \leftarrow \emptyset$; $S_D \leftarrow \emptyset$; $S_P \leftarrow \emptyset$; $S_\circledcirc \leftarrow \emptyset$;
Create state s_D; $S_D \leftarrow S_D \cup \{s_D\}$; $s_0 \leftarrow s_D$;
for all $B \subseteq Act_{\mathfrak{p}_i,D}$ **do**
  Create state s_A; $S_A \leftarrow S_A \cup \{s_A\}$; $F_1 \leftarrow F_1 \cup \{s_A\}$;
  $T(s_D, B) \leftarrow s_A$; $L(s_A) \leftarrow B$;
  $r_D(s_D, B) \leftarrow \sum_{a \in B} c(a)$;
end for
for all $s_A \in F_1$ **do**
  for all $C \subseteq Act_{\mathfrak{p}_i,A}$ **do**
    Create state s_P; $S_P \leftarrow S_P \cup \{s_P\}$; $F_2 \leftarrow F_2 \cup \{s_P\}$;
    $T(s_A, C) \leftarrow s_P$;
    $L(s_P) \leftarrow C \cup B$ where $B \subseteq L(s_A)$;
    $r_A(s_A, C) \leftarrow \sum_{a \in C} c(a)$;
  end for
end for
Create states s_s, s_f; $S_\circledcirc \leftarrow S_\circledcirc \cup \{s_s, s_f\}$;
$L(s_s) \leftarrow \{success, \mathfrak{p}_i = T\}$; $L(s_f) \leftarrow \{failure, \mathfrak{p}_i = F\}$;
for all $s_P \in F_2$ **do**
  let $p = \sum_{E \subseteq BC s.t. eval(t,E)} \prod_{a \in E} p(a) \prod_{a \in BC \setminus E} 1 - p(a)$ where $BC \subseteq L(s_P)$;
  $P(s_P, s_s) \leftarrow p$; $P(s_P, s_f) \leftarrow 1 - p$;
end for
$S \leftarrow S_A \uplus S_D \uplus S_P \uplus S_\circledcirc$;

Moreover, similarly to a DTMC, we cannot generate games without fixing an order of the players. We assume that the attacker has more information than the defender, thus in a game the defender will be the first to move. On the contrary, a tree consisting of two non-sequential sub-trees t_1, t_2 combined with a sequential operator illustrates the dynamic behaviour of the players. Here, the choices of the basic actions in t_2 might depend on the outcome of t_1. We take care of this in the sequential composition of two sub-trees, formalised below.

Algorithm 1 displays how to transform a non-sequential tree t_i to a game. The transformation first considers all nondeterministic transitions of the defender and of the attacker, and then the probabilistic transitions. We start with the initial state s_0 belonging to the defender. For all subsets $B \subseteq Act_{\mathfrak{p}_i,D}$ of the defender actions we have an outgoing edge from s_0 entering an attacker state

Algorithm 2 Sequential composition of two sub-trees.

Input: an attack-defence tree $t = op(t_1, t_2)$, $op \in \{\&_{\overrightarrow{\wedge}}, \&_{\overrightarrow{\vee}}\}$ and corresponding STGs $\mathcal{M}_1, \mathcal{M}_2$
Output: STG $(\Pi, S, s_0, \alpha, (S_A, S_D, S_P, S_{\circledcirc}), P, T, AP, L)$

Let m be the number of final states in $\mathcal{M}_1$;
Create m disjoint copies $\mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$ of $\mathcal{M}_2$;
Merge $\mathcal{M}_1, \mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$;
Replace each final state j labelled with *"success"* of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$;
Add the label $\mathfrak{p}_1 = T$ to the starting state of $\mathcal{M}_2^j$;
Replace each final state j labelled with *"failure"* of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$;
Add the label $\mathfrak{p}_1 = F$ to the starting state of $\mathcal{M}_2^j$;
Change the label of each final state of $\mathcal{M}_2^j$ base on the evaluation $[\![t]\!](Done)$;

labelled with the subset B. The outgoing edges are labelled with the sum of the costs of the actions in the subset B, $\sum_{a \in B} c(a)$. For each attacker state we do a similar construction, i.e., each attacker state has as many outgoing edges as the subsets $C \subseteq Act_{\mathfrak{p}_i, A}$ of the attacker actions. Similarly, the edges are labelled with the sum of the costs $\sum_{a \in C} c(a)$ and they enter the probabilistic states labelled with the corresponding subset C. Each probabilistic state has two outgoing edges. One of the edges enters the final state labelled with *success* and is labelled with the sum of the success probabilities of the actions that evaluates the tree t_i to true. The other edge enters the final state labelled with *failure* and is labelled with the sum of the failure probabilities. The final states labelled with *success* and *failure* are also instrumented with $\mathfrak{p}_i = T$ meaning that the phase $\mathfrak{p}_i$ was successful, and $\mathfrak{p}_i = F$ meaning that the phase $\mathfrak{p}_i$ failed, respectively.

So far we have described how to transform each non-sequential sub-trees to a Stochastic Two-player Game. We combine the games corresponding to sub-trees by means of the sequential composition. Consider two sub-trees t_1, t_2 connected with a sequential operator $op \in \{\&_{\overrightarrow{\wedge}}, \&_{\overrightarrow{\vee}}\}$, $t = op(t_1, t_2)$, and the corresponding games $\mathcal{M}_1, \mathcal{M}_2$. The sequential composition of two games is presented in Algorithm 2, and is as follows. Assume $\mathcal{M}_1$ has m final states. We create m disjoint copies of $\mathcal{M}_2$, denoted $\mathcal{M}_2^1, \cdots, \mathcal{M}_2^m$. For each final state j of $\mathcal{M}_1$ labelled with *success* we connect $\mathcal{M}_2^j$ with $\mathcal{M}_1$ by replacing the final state j of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$ and adding the label $\mathfrak{p}_i = T$ to the starting state of $\mathcal{M}_2^j$. Similarly, for each final state j of $\mathcal{M}_1$ labelled with *failure* we connect $\mathcal{M}_2^j$ with $\mathcal{M}_1$ by replacing the final state j of $\mathcal{M}_1$ with the starting state of $\mathcal{M}_2^j$ and add the label $\mathfrak{p}_1 = F$ to the starting state of $\mathcal{M}_2^j$. We evaluate and re-label (if needed) each final state of $\mathcal{M}_2^j$ based on the set *Done*

of performed actions on the path from the starting state of $\mathcal{M}_1$ till the final state, $[\![t]\!](\mathit{Done})$, where $[\![t]\!]$ is the Boolean formula of which the tree t is a parse tree.

In the special case where there are only sequential conjunctions, we can optimise the construction of the game by merging together all final states labelled with *success* and all final states labelled with *failure*. Observe that merging the final states together does not cause a lose of information in the history of a game.

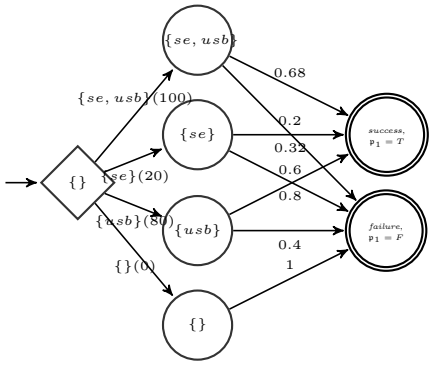
Example. Let us construct an STG from the tree t, displayed in Figure 6.1, by following the steps described above. First we transform each basic sub-tree to a game through Algorithm 1. Figure 6.6 presents the constructed games for each basic sub-tree. As we can see, each game has first the nondeterministic transitions of the defender, then the nondeterministic transitions of the attacker and finally the probabilistic transitions. We combine the constructed games by means of the sequential composition, as explained in Algorithm 2. As the tree has only sequential conjunction, we merge the final states with same label during the sequential composition. The full game for the attack-defence tree t is illustrated in Figure 6.7.

6.2.2 Probabilistic Model Checking Stochastic Games

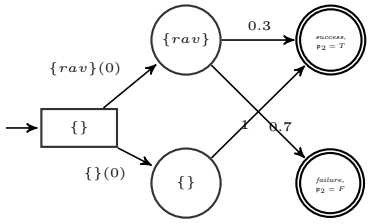
In the previous section we proposed a transformation from attack-defence trees to STGs. The main focus of this section is to show how to evaluate security properties over all possible strategies and how to synthesise optimal attack (or defence) strategies. We start with a discussion of the security properties of interest and then discuss their representation in the temporal logic rPATL. This allows us to perform our analysis of attack-defence trees using the existing model checking techniques implemented in PRISM-games.

Security properties. We can phrase a great many useful quantitative questions on attack-defence scenarios, concerning either one player or both players. It is worth observing that a question might refer to one or both players depending on the parameters they are formulated over. For example, cost-related questions refer to one player: for computing the cost of an attack we do not require the cost of the defender actions. On the other hand, probability-related questions refer to both players, i.e., if the attacker succeeds with probability p then the defender succeeds with probability $1 - p$.

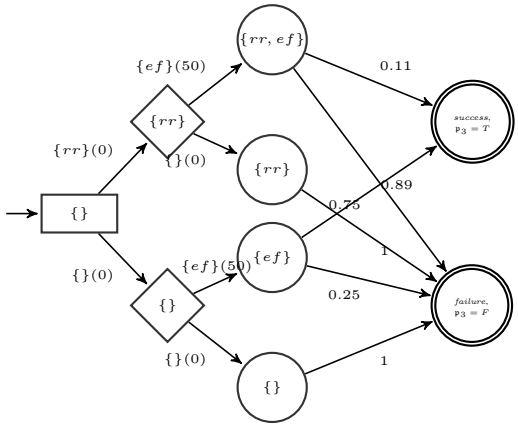
In this work we characterise the basic actions of an attack-defence scenario with the success probability and the cost of an attack and a defence. We then study properties with one objective, e.g., "is there an attack which is successful with probability greater than or equal to 0.03?" or "what is the maximum success probability of an attack?". Later in Sect. 6.3 we will investigate properties with multiple objectives.



(a) An STG for $t_1 = \vee(se, usb)$



(b) An STG for $t_2 = \sim rav$



(c) An STG for $t_3 = \wedge(ef, \sim rr)$

Figure 6.6: Transformation of basic sub-trees to games (attacker/defender/probabilistic states shown as diamonds/rectangles/circles).

Verification of security properties. Formal verification is used to determine whether or not the system under study exhibits certain precisely specified properties. For verifying security properties of STGs, we exploit probabilistic model checking of rPATL. This logic allows us to express a wide range of properties. For instance, the first single-objective property above is expressed in rPATL as the formula $\langle\langle A\rangle\rangle P_{\geq 0.03}[F\, success]$, while the second property is expressed as the formula $\langle\langle A\rangle\rangle P_{max=?}[F\, success]$.

Model checking systematically explores all states and transitions in the model to check whether it satisfies the given property. Moreover, probabilistic model checking of rPATL also allows us to synthesise strategies for a player with respect

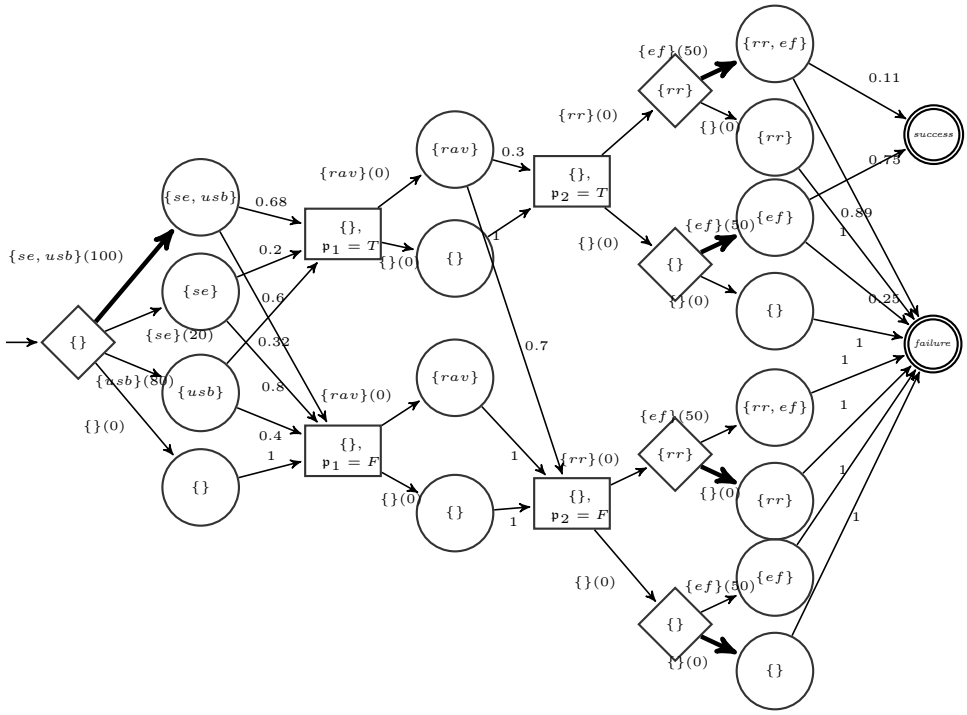


Figure 6.7: The STG for attack-defence tree t, from Figure 6.1. An optimal strategy for the attacker player is marked in bold.

to a given property. For instance, we can determine which is the optimal strategy for the attacker in terms of maximising the success probability of the attack, for all possible strategies that the defender may choose. In fact, we can also determine, at the same time, what the best strategy for the defender to ensure that the probability of success does not exceed this.

The model checking techniques described here are all implemented in PRISM-games, which we therefore employ for verification and strategy synthesis problems on attack-defence trees. PRISM-games also generates optimal strategies.

Correctness. We conclude this section by sketching the correctness of our approach, i.e., that the construction and analysis of the stochastic game described above yields the right answers to questions phrased in terms of attack-defence trees. This relies on the correspondence between an attack-defence tree t, as formalised in Sect. 6.1, and the STG $\mathcal{G}$ whose construction is outlined in Sect. 6.2.1. More precisely, this depends on a correspondence between decision trees for t and their corresponding attacker or defender player strategies in the stochastic game $\mathcal{G}$.

In Sect. 6.1.3, we gave a precise definition of the semantics of a pair of attacker/defender decision trees d^A, d^D in terms of a Discrete-Time Markov Chain. Each decision tree d^τ has an equivalent strategy, say σ_τ, for player τ in $\mathcal{G}$. As mentioned in Sect. 5.1.2, the behaviour of $\mathcal{G}$ under a pair of strategies σ_A, σ_D is also represented by a DTMC. It is the equivalence of these two Markov Chains which underlies the correctness of the overall approach. An important issue here is the class of stochastic game strategies that we need to consider. For the properties used in this paper (those in the logic rPATL), it suffices to consider *memoryless* strategies, which makes the equivalence of the two Markov Chains relatively straightforward. The relation between stochastic game strategies and decision trees is expanded upon in the following section.

Example. Consider the game given in Figure 6.7. We use the tool PRISM-games to verify the security properties mentioned above. For example, the verification of the query $\langle\langle A\rangle\rangle P_{\geq 0.03}[F\, success]$ returns "false", meaning that there is no attack with success probability greater than or equal to 0.03. The verification of the quantitative query $\langle\langle A\rangle\rangle P_{max=?}[F\, success]$) computes the maximum success probability of an attack, which is 0.0229. Figure 6.7 also shows an optimal attacker strategy, marked in bold.

6.2.3 Synthesising Strategies as Decision Trees

After synthesising an optimal strategy from the stochastic game, as described above, we can transform it to a corresponding decision tree. This provides a high-level, syntactic description of the strategy, in particular, capturing the dependencies within the strategy on the outcomes of earlier actions and choices by players. We now describe this process, first for an attacker, and then for a defender.

Attacker strategies. Synthesis of an attacker decision tree, from an STG $\mathcal{G}$ and attack strategy σ_A, is done using the recursive function $\mathsf{generateAD}[\mathcal{G}, \sigma_A](s, i)$, shown in Table 6.5, which operates over the structure of $\mathcal{G}$. The first parameter s is a state of $\mathcal{G}$ and the second parameter i keeps track of the current phase (to be precise, the phase i associated with the decision tree node currently being created). Throughout the evaluation of the function we assume to have the STG $\mathcal{G}$, and the attack strategy σ_A as a global parameters. At the top-level, we call the function as $\mathsf{generateAD}[\mathcal{G}, \sigma_A](s_0, 1)$, where s_0 is the initial state of $\mathcal{G}$.

By construction of the game $\mathcal{G}$ (see Sect.6.2.1), its states are grouped by phase, and within each phase there are (possibly) defender and then (possibly) attacker states, followed by probabilistic states at the end of the phase. We treat the three classes of state separately.

If s is a defender state, $s = s_D \in S_D$, then the strategy σ_A will not have resolved the choice of actions in s_D and we need to consider each of the possible

Table 6.5: generateAD: construction of attacker decision tree from STG and attacker player strategy.

$$\mathsf{generateAD}[\mathcal{G},\sigma_A](s_D,i) = \mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,Act_{\mathfrak{p}_i,D},\emptyset)$$

$$\mathsf{generateAD}[\mathcal{G},\sigma_A](s_A,i) = \sigma_A(s_A).\mathsf{generateAD}[\mathcal{G},\sigma_A](s_P,i+1)$$
where $s_P = T(s_A,\sigma_A(s_A))$

$$\mathsf{generateAD}[\mathcal{G},\sigma_A](s_P,i) =$$
$$\mathit{if}(\mathfrak{p}_{i-1}?,\mathsf{generateAD}[\mathcal{G},\sigma_A](s',i),\mathsf{generateAD}[\mathcal{G},\sigma_A](s'',i))$$
where $P(s_P,s') > 0 \wedge$ "$\mathfrak{p}_{i-1} = T$" $\in L(s')$
and $P(s_P,s'') > 0 \wedge$ "$\mathfrak{p}_{i-1} = F$" $\in L(s'')$

$$\mathsf{generateAD}[\mathcal{G},\sigma_A](s_P,n+1) = \mathit{stop}$$

$$\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,LA \cup \{a\},\mathit{Done}) =$$
$$\mathit{if}(a?,\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,LA,\mathit{Done}\cup\{a\}),\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,LA,\mathit{Done}))$$

$$\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,\emptyset,\mathit{Done}) =$$
$$\begin{cases} \mathsf{generateAD}[\mathcal{G},\sigma_A](s',i) & \text{if } s' = T(s_D,\mathit{Done}) \in S_A \\ \emptyset.\mathsf{generateAD}[\mathcal{G},\sigma_A](s',i+1) & \text{if } s' = T(s_D,\mathit{Done}) \in S_P \end{cases}$$

outgoing branches. These will be translated into *if* statements in the decision tree, which can ask whether a defender action was performed in the current phase (see Sect. 6.1.2). This is done by calling $\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,Act_{\mathfrak{p}_i,D},\emptyset)$, explained below.

If the state s is an attacker state, $s = s_A \in S_A$, we create an action node with the set of attacker actions performed in state s_A, as specified by the strategy choice $\sigma_A(s_A)$, and the next node in the decision tree is generated recursively for the successor state $T(s_A,\sigma_A(s_A))$, which, by construction of the game, will be a probabilistic state.

For a probabilistic state, $s = s_P \in S_P$, we have reached the end of current phase in the STG. We create a decision node whose condition depends on the success of the phase, and then recursively construct the decision tree for the successor states of s_P corresponding to the scenarios where the phase succeeded or failed. Notice that we create a decision node for phase i (i.e., a node d_i^A from Table 6.2) which queries the state of the preceding phase $\mathfrak{p}_{i-1}$. Once we reach the end of the phases (indicated by $i = n+1$), we have reached the end of the STG and there are no further actions to be taken so we create a stop node in the decision tree.

As mentioned above, defender states s_D are treated using an auxiliary recursive function $\mathsf{construct}[\mathcal{G},\sigma_A](s_D,i,LA,\mathit{Done})$, which is also given in Table 6.5. The first two parameters are as for generateAD, the third, LA, is the set of the defender actions to be performed and the last, *Done*, is the set of actions already performed. The function iterates over the actions in LA (initially, the set

Table 6.6: generateDD: construction of defender decision tree from STG and defender player strategy.

$\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_D, i, D) = \sigma_D(s_D).\mathsf{generateDD}[\mathcal{G}, \sigma_D](s', i+1, A)$
where $s' = T(s_D, \sigma_D(s_D))$

$\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_A, i, A) = \mathsf{generateDD}[\mathcal{G}, \sigma_D](s_P, i+1, A)$
where $s_P = T(s_A, B)$ for some B

$\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_A, i, D) = \emptyset.\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_P, i+1, A)$
where $s_P = T(s_A, B)$ for some B

$\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_P, i, A) =$
$\mathit{if}(\mathfrak{p}_{i-1}?, \mathsf{generateDD}[\mathcal{G}, \sigma_D](s', i, D), \mathsf{generateDD}[\mathcal{G}, \sigma_D](s'', i, D))$
where $P(s_P, s') > 0 \wedge$ "$\mathfrak{p}_{i-1} = T$" $\in L(s')$
and $P(s_P, s'') > 0 \wedge$ "$\mathfrak{p}_{i-1} = F$" $\in L(s'')$

$\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_P, n+1, A) = \mathit{stop}$

$Act_{\mathfrak{p}_i, D}$ of all defender actions for phase i), each time removing an action a and creating a decision node with condition a? and recursively building the decision tree for the cases where the condition is true or false. This creates decision nodes that branch over the possible combinations. Once, the parameter LA is empty, we recursively construct the next part of the decision tree, using the outgoing transitions of the s_D state. These will either go to an attacker state s_A or directly to a probabilistic state s_P. In the latter case, we add an action node with an empty action set, indicating that the attacker performs no actions in this phase.

Defender strategies. The generation of a defender decision tree from a game and a defender tree is slightly different, since, here, the defender can ask only about the success of the previous phases, not any attacker actions from the current phase.

Again, we use a recursive function operating over the states of the game $\mathcal{G}$. This function, $\mathsf{generateDD}[\mathcal{G}, \sigma_D](s, i, \tau)$ is shown in Table 6.6 and constructs a decision tree for a strategy σ_D of the defender player in $\mathcal{G}$. Parameters s and i are the current state and phase, as for generateAD, above; parameter τ represents the next player to move. At the top-level, we call the function as $\mathsf{generateDD}[\mathcal{G}, \sigma_D](s_0, 1, D)$.

If the state is a defender state, $s = s_D \in S_D$, we create an action node with the set of actions performed by the defender in s_D, obtained from the defender strategy σ_D, and proceed recursively using the successor of s_D chosen by σ_D.

If the state is an attacker state, $s = s_A \in S_A$, and the current player is A, we just move to the next probabilistic state s_P. The state s_P is chosen

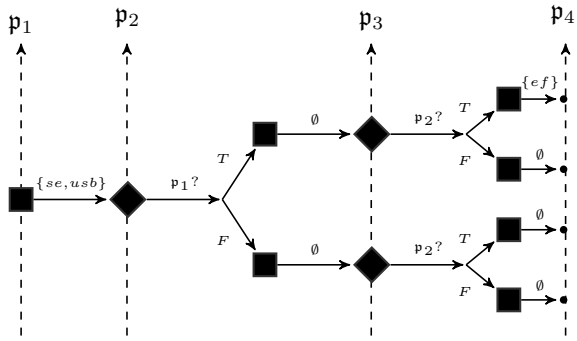


Figure 6.8: An attack decision tree for the optimal attacker strategy highlighted in the stochastic game shown in Figure 6.7.

nondeterministically. Note, that for any choice of s_P further construction on the decision tree is the same. If the current player is D, this means that there is no defender action in the current phase. Thus, we create an empty set in the decision tree and move to the next probabilistic state s_P.

On probabilistic states, the function generateDD behaves the same as the function generateAD, as described above.

Finally, we note that the decision tree constructed as above can subsequently be optimised by merging identical sub-trees and removing decision nodes with identical then/else branches.

Example. The stochastic game in Figure 6.7 also shows an optimal attacker strategy marked in bold. We show in Figure 6.8 the (optimised) attacker decision tree corresponding to the optimal attacker strategy.

6.3 Extension with Multi-Objective Properties

So far we proposed a game-theoretic approach, translating attack-defence trees to STGs, and then employed probabilistic model checking techniques for formally analyse these models. We studied the analysis of single-objective properties for an attack-defence scenario. We shall now move towards multi-objective properties.

An rPATL-based approach for multi-objective probabilistic model checking of stochastic games allows to analyse multiple objectives, and compute the associated Pareto curve. The approach focuses on expected reward objectives. Although expected reward objective can express various useful properties for games, however the main focus in an attack-defence tree evaluation is on exact cost, i.e., cost of performing a successful attack without taking into consideration the probability measure. That is why, here we will briefly discuss the

analysis of multi-objective properties expressed as expected reward by omitting the details. We will discuss exact cost in Part III.

In this section we present briefly how we can extend the proposed framework for verifying multi-objective properties of the attack-defence scenarios, and synthesising strategies for the attacker and the defender which guarantee or optimise some multi-objective quantitative property.

Optimal strategies for multi-objective properties may be randomised. Thus, we extend the framework to incorporate *randomisation*, i.e., we extend the syntax and the semantics of decision trees, as well as the construction of a decision tree from a strategy to incorporate randomisation. Observe that multi-objective queries in stochastic games need infinite-memory strategies in general, but our games are trees (or DAGs) so memoryless strategies suffice.

Randomisation in decision trees and semantics. Decision trees represent the attacker or the defender strategies that incorporate dependencies. For representing strategies that optimise multiple objectives, we consider randomisation in decision trees. More precisely, instead of just allowing the attacker or defender to choose to execute a set of actions B in a given phase, we allow them to probabilistically select between several different action sets. In terms of the syntax of a decision tree, presented in Table 6.2, in addition to action nodes $d = B.d'$ for $B \subseteq Act_{\mathfrak{p}_i,\tau}$, we allow $d = \mu.d'$, where μ is a discrete probability distribution over $Act_{\mathfrak{p}_i,\tau}$, indicating that each action set B_i may be picked, with probability $\mu(B_i)$, before proceeding to the sub-tree d'.

The random selection of actions in decision trees can be added to the semantics in Table 6.3 in straightforward fashion: a node $d = \mu.d'$ results in a single DTMC state with one outgoing transition for each element of the support of μ, each of which is a normal action node of the form $d'' = B.d'$.

Randomisation in construction of decision trees. We consider decision trees that incorporate randomisation. In the transformation of a strategy to a corresponding decision tree, that means that strategy σ_A (or σ_D) may select a distribution over actions in a state s_A (or s_D), rather than a single action. The decision tree synthesis algorithms in Tables 6.6 and 6.5 thus remain unchanged but the rules for states s_A and s_D, respectively generate random action nodes.

Multi-objective security properties. Here, we study properties with multiple objectives, such as "can we achieve an attack with an expected cost of at most 500 and a success probability of at least 0.005?". As in the case of a single objective, we use rPATL to express the properties. For example, the above mentioned multi-objective property is expressed as the formula $\langle\langle A\rangle\rangle(E^{r_A}_{\leq 500}[F\,success] \wedge P_{\geq 0.005}[F\,success])$.

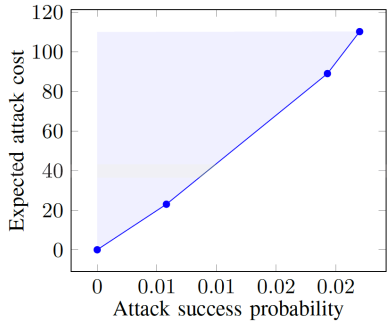


Figure 6.9: Pareto curve illustrating the trade-off between attack success probability and expected attack cost over strategies in the running example.

For verification of multi-objective properties we use an extension of rPATL model checking [CFK+13b]. The extension allows us both to verify security properties and to synthesise strategies for a player, e.g., "what strategy of the attacker ensures that the expected cost of an attack is at most 500, while the success probability is at least 0.005?". In addition, we can compute the Pareto curve of achievable objectives.

We employ PRISM-games for verification and strategy synthesis of multi-objective queries. PRISM-games also can compute and display graphically the Pareto curve associated with two objectives.

Example. Consider the game given in Figure 6.7. We verify multi-objective queries such as $\langle\langle A\rangle\rangle(E^{r_A}_{\leq 500}[F\,success] \wedge P_{\geq 0.005}[F\,success])$. The property evaluates to "true" meaning that there is an attack with cost at most 500 and success probability at least 0.005. Moreover, Figure 6.9 illustrates the Pareto curve computed by PRISM-games when maximising probabilities and minimising cost of an attack.

6.4 Implementation

We applied our approach to a real-life scenario studied in [BKMS12]: we consider part of a Radio-Frequency Identification (RFID) goods management system for a warehouse, modified by introducing temporal dependencies between actions.

The warehouse uses RFID tags to electronically identify all goods. In the attack-defence scenario that we consider, the attacker aims to physically remove

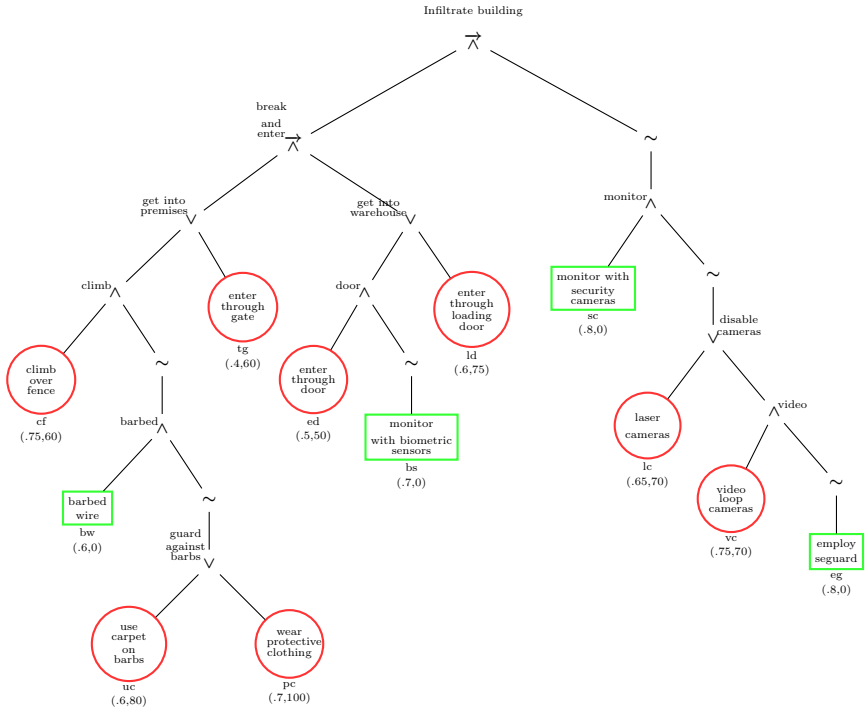


Figure 6.10: Attack-defence tree for breaking and entering a building.

some RFID tags after infiltrating the building.

In order to achieve this goal, the attacker has to first get into the premises and then into the warehouse. For getting into the premises the attacker can climb over the fence or enter through the main gate. The defender can protect against climbing by setting some barbed wire on the fence. To protect against the barbed wire the attacker can guard against barbs either by using a carpet over the barbs or by wearing protective cloths. Once the attacker succeeds in accessing the premises, they have to get into the warehouse. The attacker can achieve this sub-goal either by entering through the door or by entering through the loading dock. The former action can be defended against by monitoring the door with biometric sensors.

The defender can prevent the attacker from attaining the main goal by monitoring the premises with security cameras. In order to overcome the camera issue the attacker can disable them either by shooting a strong laser at the cameras or by video looping the camera feed. The defender, in turn, can employ guards in order to patrol the premises and counter this kind of attack.

The corresponding attack-defence tree is given in Figure 6.10. The leaves (basic attack and defence actions) of the tree are decorated with success proba-

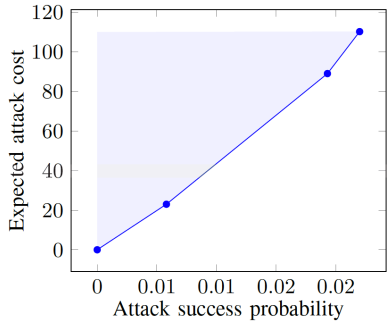


Figure 6.11: Pareto curve illustrating the trade-off between attack success probability and expected attack cost over strategies for the RFID example.

bility and cost values. The attack-defence tree has three phases: the first phase corresponds to the sub-tree with the root "get into premises", the second phase is the "get into warehouse" sub-tree, and the last phase is the sub-tree on the right of the main goal with the defender action on the root. The syntactic term corresponding to each phase and to the full tree is:

$$\begin{aligned} t_1 &= \&_\vee(\&_\wedge(ef, \&_\sim\&_\wedge(bw, \&_\sim\&_\vee(uc, pc))), tg) \\ t_2 &= \&_\vee(\&_\wedge(ed, \&_\sim bs), ld) \\ t_3 &= \&_\sim(\&_\wedge(sc, \&_\sim\&_\vee(lc, \&_\wedge(vc, \&_\sim eg)))) \\ t &= \&_{\overrightarrow{\wedge}}(\&_{\overrightarrow{\wedge}}(t_1, t_2), t_3) \end{aligned}$$

The resulting stochastic game generated from the attack-defence by our approach has 1072 states and 2052 transitions. We verified a variety of properties, including the numerical property $\langle\langle A\rangle\rangle P_{max=?}[F\, success]$) that computes the maximum success probability of an attack (equal to 0.41), and the multi-objective qualitative property $\langle\langle A\rangle\rangle(R^{r_A}_{\leq 150}[F\, success] \wedge P_{\geq 0.1}[F\, success])$ (which evaluates to true, meaning that there is an attack with cost at most 150 and success probability at least 0.1). We also examine the trade-off between the probability of a successful attack at the expected cost of doing so. The Pareto curve generated for this pair of properties is shown in Figure 6.11.

Implementation. We have developed a prototype implementation of our techniques, comprising a converter from attack-defence trees, specified in XML, into stochastic games modelled in the input language of PRISM-games [KPW16], available in Java at

`www2.compute.dtu.dk/~zaas/ADT2PRISM.zip`

The output of the tool can then be used to perform verification and strategy synthesis as described earlier. The presented two example in this chapter are available with the tool.

6.5 Concluding Remarks

As we have argued in Ch. 4, attack-defence trees are a useful tool to study attack-defence scenarios and present the interaction between an attacker and a defender in an intuitive way. However, they offer a rather static view on the scenario, while a game approach promotes the dynamic interaction of the players as a first-class citizen of the domain, thanks to the notion of strategy. Stochastic Two-player Games are a more general and widely-used formalism to study competing behaviours, and benefit from a rich corpus of literature on automated verification and tool support.

In this chapter, we explored the relation between attack-defence trees and STGs. We proposed a framework for evaluating security properties of attack-defence scenarios, by developing an extension of attack-defence trees in which temporal dependencies among sub-goals can be expressed. In order to formally represent strategies for the players in presence of such dependencies, we have defined the novel concept of decision trees, whose semantics we have given in terms of Discrete-Time Markov Chains. Moreover, we have shown how to encode an attack-defence tree into an STG, where it becomes natural to study the interaction between players and to account for quantitative and probabilistic aspects of a scenario. This allows us to exploit the power of probabilistic model checking techniques and tools, to verify security properties automatically and synthesise strategies for attacks and defences. These strategies can be converted to decision trees, linking the outcome of the verification on the game model to the original attack-defence tree, facilitating communication of the results to end-users.

We implemented our approach in a prototype tool and applied it to the example of an RFID goods management system, where the analysis gives insights on the points of the system open to attack and the corresponding effort to the attacker and likelihood of success.

Our current approach requires that sequential operators only occur above non-sequential operators in an attack-defence tree. Future work includes generalisation of the approach allowing sequential operators to occur anywhere in a tree. Moreover, it would be interesting to consider partially-observable games instead of fully-observable ones.

In order to address multi-parameter optimisation we considered multi-objective probabilistic model checking and extended the framework to incorporate randomisation. The approach for model checking of Stochastic Two-player Games

with multiple objectives investigates the *expected* cost or reward objectives, i.e., the summation of the rewards multiplied with the probabilities along the path. The expected reward captures many useful properties of stochastic games. However, it is sometimes important to know the *exact* costs of attacks or defences, especially when a given budget is constraining the resources available to either player. Moreover, exact cost is studied in many evaluation techniques of attack-defence scenarios, hence we cannot claim the verification techniques proposed in this chapter fully cover those developed in Chs. 3, 4 directly on attack and attack-defence trees. To the best of our knowledge, no extension of rPATL exists that encompass exact costs. Therefore, in Part III we will embrace the challenge of extending the model checking approach to attack tree evaluation to consider exact cost analysis.

Graphical and game-theoretical approaches. While most extensions study static attack trees, a few consider dynamic aspects. Arnold et al. [AHPS14] analysed the timing of attack scenarios using Continuous-Time Markov Chains, but do not reason about strategies; [KRS15] used priced time automata and the Uppaal model checker to analyse attack trees, but without probabilities. More recently, [HKKS16] explored how stochastic timed automata can be used to study attack-defence scenarios where timing plays a central role. None of these approaches use game-based models.

Elsewhere, various studies have explored a game-theoretic approach to modelling security aspects of a system. In particular, stochastic games [Sha53] have proven useful to model uncertainty and randomisation of security scenarios, and have been explored in several application domains. Lye and Wing [LW05] modelled the security of computer network as a stochastic game and computed Nash equilibrium strategies for the players. Ma et al. [MRY11] presented a game-theoretic approach for studying rational attackers and defenders in the security of cyber-physical systems. Along similar lines, Vigo et al. [VBY13] proposed a framework for modelling and analysing the security of cyber-physical systems by means of stochastic games.

Part III

The Logic erPCTL for Attack Trees

In Part II we showed the connection between attack-defence trees and STGs, and proposed a game-theoretic approach for analysing attack-defence scenarios. We discussed a limitation of rPATL, namely, the logic does not support the evaluation of the *exact cost* of the execution.

The reward operator in rPATL computes the expectation of the cumulative cost with respect to the probability measure. It does not evaluate the cost value itself without probabilities. The cumulative exact cost is however a useful concept to reason about cost-related properties of the execution. It allows to compute the cheapest execution or to verify that the cost of an execution is within a given budget. In order to compute the exact cost of an attack and to analyse queries with probability *and* cost bounds, we extend rPCTL with new operators. We develop the theory in the simpler setting of attack trees. This simplification removes the defender from the scene focusing on the evaluation of attack scenarios and moving the game from two players to one player, namely MDPs.

We use MDPs to model the probabilistic and nondeterministic behaviour of an attacker in an attack scenario, and propose a translation from attack trees to MDPs. To verify security properties expressed in rPCTL, we employ probabilistic model checking. The logic rPCTL allows to express properties ranging over probability and expected cost. Even though there exist a few studies that have extended rPCTL with reward-bounded properties, as we will discuss in Sect. 7.3, they evaluate properties on Discrete- and Continuous-Time Markov Chains. To evaluate exact cost-related properties on MDPs, we extend rPCTL with new operators and present the corresponding model checking algorithm.

Moreover, we study the connection between attack trees and DTMCs for representing the purely probabilistic behaviour of the system, for which we develop a bisimulation-based analysis that leads to a straightforward visualisation of the feasibility of an attack.

This part is organised as follows. We start with the background on MDPs and probabilistic model checking in Ch. 7, where we also present the translation of attack trees to DTMCs and the evaluation of probabilistic properties expressed in rPCTL. The extended logic with cost-related operators, as well as the proposed translation from trees to MDPs is formulated in Ch. 8.

Chapter 7

Preliminaries : Markov Decision Processes

So far we presented DTMCs, which are purely probabilistic models, and STGs, which incorporate the competitive behaviour of two players. In this chapter we review MPDs, which extend DTMCs with nondeterministic choices. Moreover, MDPs are a special case of STGs where the nondeterministic actions of one player are fixed.

MDPs are transition systems that model nondeterministic and probabilistic behaviour. MDPs are used to model various kinds of systems, including for example randomised distributed algorithms, planning, queuing and optimisation problems. Phenomena like message loss, processor failure and the like may be modelled by nondeterminism. As for DTMCs, in order to analyse such systems quantitative verification techniques, namely *probabilistic model checking*, are applied, extended to deal with nondeterministm.

Probabilistic model checking is used to verify automatically whether or not a model satisfies its specification. It requires as input a state transition system and a property of interest. In order to express the qualitative and quantitative properties of the system, the temporal logic rPCTL is used. This logic allows to express probability and reward-based properties such as "the attack will succeed at least with probability 0.7" or "what is the maximum probability of eventually reaching a successor state?".

Section 7.1 gives the essential background on MDPs and related concepts, while Sect. 7.2 reviews the syntax and the semantics of the logic, as well as the model checking algorithm, presented in the wake of [BK08, Ch.10] [FKNP11,

KP12]. We discuss various reward operators proposed in the literature for rPCTL in Sect. 7.3. Finally, Sect. 7.4 presents a translation from attack trees to DTMCs and shows what analyses are supported by existing techniques.

7.1 Markov Decision Processes

Definition 7.1 (MDP) A Markov Decision Process is a tuple $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ where we can find sets S_A (of attacker nondeterministic states), S_P (of probabilistic states), and $S_\circledcirc$ (of final states), such that

- $S = S_A \uplus S_P \uplus S_\circledcirc$, where $\uplus$ denotes the finite disjoint union of sets;
- α is a finite, non-empty set of actions;
- $P : S_P \times S \to [0, 1]$ is a probabilistic transition function such that for all probabilistic states $s \in S_P$ $\sum_{s' \in S} P(s, s') = 1$;
- $T : S_A \times \alpha \to S$ is a transition function;
- $s_0 \in S$ is the initial state;
- AP is a set of atomic propositions; and
- $L : S \to 2^{AP}$ is a labelling function.

The probabilistic transition function P is defined in a similar way as for DTMCs. It describes the probability $P(s, s')$ of a transition from the state s to the state s' in one step. Final states are modelled by adding a self-loop, i.e., for all $s \in S_\circledcirc$ the probability of a transition going back to s is 1, $P(s, s) = 1$.

The transition function T is used to solve nondeterminism. For a state s and an action $l \in \alpha$ selected nondeterministically function T specifies the successor state s', $T(s, l) = s'$. We denote by $\alpha(s)$ the set of enabled actions in the state $s \in S_A$, $\alpha(s) = \{l \in \alpha \mid s \in S_A \text{ and } T(s, l) \text{ is defined}\}$. The labelling function L maps each state s to a set of atomic propositions.

The set of direct successors of s is defined as $Post(s) = \{s' \in S_A \mid \exists l \in \alpha : T(s, l) = s'\} \cup \{s' \in S_P \cup S_\circledcirc \mid P(s, s') > 0\}$. Similarly, the set of direct predecessors is defined as $Pre(s) = \{s' \in S_A \mid \exists l \in \alpha : T(s', l)\ s\} \cup \{s' \in S_P \cup S_\circledcirc \mid P(s', s) > 0\}$.

An *infinite path* in an MDP is a non-empty sequence of states $\pi = s_0 s_1 \cdots$ where $s_i \in S$. A *finite path* is a finite sequence of states $\pi = s_0 \cdots s_n$, where $s_i \in S$. We denote by $Path_s^{fin}$ and $Path_s$ the set of all finite and infinite paths that start in state s, respectively, and by $\pi[i]$ we denote the i-th state of the path, $\pi[i] = s_i$.

Schedulers. Reasoning about probabilities in an MDP relies on the determination of nondeterminism. A scheduler solves nondeterminism by choosing an action for each nondeterministic state of an MDP. A *scheduler* for $\mathcal{M}$ is a function $\sigma : S^*S_A \to \alpha$ that maps a finite path to an action. A scheduler corresponds to one possible resolution of nondeterminism. A scheduler σ is *memoryless* if for any $\pi, \pi' \in S^*$ and $s \in S_A$, $\sigma(\pi s) = \sigma(\pi' s) = \sigma(s)$. We denote by Σ the set of all possible schedulers of an MDP. A probability measure Pr_s^σ under a scheduler σ is defined in the standard fashion.

As we mentioned above, a scheduler σ solves nondeterministic choices in an MDP reducing the model to a purely probabilistic one. Thus, a scheduler σ induces a DTMC that corresponds to the behaviour of the MDP under the decision of σ. The following definition formalises this intuition.

Definition 7.2 (Induced DTMC) Let $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ be an MDP and σ a memoryless scheduler on $\mathcal{M}$. The DTMC $\mathcal{D}_\sigma = (S, P_\sigma, s_0, AP, L)$ induced by σ on $\mathcal{M}$ has P_σ as probabilistic transition function, defined by

$$P_\sigma(s, s') = \begin{cases} 1 & \text{if} \quad s \in S_A \wedge T(s, \sigma(s)) = s' \\ P(s, s') & \text{if} \quad s \in S_P \\ 1 & \text{if} \quad s \in S_\circledcirc \wedge s = s' \\ 0 & \textit{otherwise} \end{cases}$$

A *reward structure* is helpful to reason about the quantitative information of an MDP. In this work we consider only state rewards and use the terminology *reward* to model *cost*. Thus, a *reward (cost) structure* on an MDP $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ is a function $r : S \to \mathbb{Q}_{\geq 0}$ that assigns the states with a cost. For a finite path $\pi = s_0 s_1 \cdots s_n$ we define the *total cost* as $cost(\pi) = \sum_{s_i \in \pi} r(s_i)$.

Remark 1 *As shown by Baier [Bai98], there are two kinds of MDP-based models. Concurrent probabilistic systems, where each state is assigned nondeterministic alternatives followed by probabilistic distribution; and stratified systems, where states are distinguished between nondeterministic and probabilistic ones. This two models have equivalent expressive power. Baier showed how to transform concurrent probabilistic systems to stratified systems.*

Building on top of Baier's approach, we could show that the stratified systems discussed in this chapter can be transformed into concurrent probabilistic systems. Informally, the procedure would work as follows. We modify transitions between states and make them two-step transitions, first nondeterministic and then probabilistic ones. For a nondeterministic states s, after choosing an action from $\alpha(s)$, the successor state is chosen probabilistically with probability 1. For a probabilistic state s we first add a nondeterministic choice of an action and then choose the successor state according to probability distribution $P(s, s')$.

7.2 Model Checking MDPs

In this section we describe rPCTL that is used to specify the properties of the system modelled by an MDP and we give the model checking algorithm for verifying the specified properties. The algorithms presented in this section are based on [FKNP11].

7.2.1 rPCTL

Syntax. We consider the extension of PCTL with rewards to evaluate the expected cost properties on an MDP. Let us remind the reader of the syntax of rPCTL, introduced in Ch. 5

$$\begin{array}{lll} \phi & ::= & \mathit{true} \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_J(\psi) \mid E^r_{\bowtie x}(F\phi) \\ \psi & ::= & X\phi \mid \phi_1 U \phi_2 \end{array}$$

where $a \in AP$ is an atomic proposition, $\bowtie \in \{\geq, >, \leq, <\}$, $J \subseteq [0,1]$ is an interval with rational bounds, $x \in \mathbb{Q}_{\geq 0}$, $r : S \to \mathbb{Q}_{\geq 0}$ is a reward structure.

Observe that there is a minor difference in the syntax of P with respect to Ch. 5. Here we write $P_J(\psi)$ as oppose to $P_{\bowtie p}(\psi)$, where the interval J allows to express bounds from below and above in a concise way. As we will see shortly, the expressiveness of the two version of the logic is the same. Moreover, reasoning about intervals will make our exposition simpler to follow.

Semantics. The semantics of the propositional logic fragment and of the path formulae is defined as for DTMCs. The difference is in the probabilistic and expected reward operators, where the notion of scheduler is introduced and the computation ranges over all schedulers $\sigma \in \Sigma$. The formal semantics of rPCTL is as follows.

Definition 7.3 (rPCTL Semantics) Let $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ be an MDP, σ a scheduler of M and $s \in S$. The satisfaction relation $\models$ of rPCTL for state formulae is defined inductively by:

$$\begin{array}{lll} s \models \mathit{true} & & \forall s \in S \\ s \models a & \text{iff} & a \in L(s) \\ s \models \neg\phi & \text{iff} & s \not\models \phi \\ s \models \phi_1 \wedge \phi_2 & \text{iff} & s \models \phi_1 \text{ and } s \models \phi_2 \\ s \models P_J(\psi) & \text{iff} & Pr^\sigma_s(\psi) \in J \text{ for all schedulers } \sigma \in \Sigma \\ s \models E^r_{\bowtie x}(F\phi) & \text{iff} & Exp^\sigma_s(Z^r_{F\phi}) \bowtie x \text{ for all schedulers } \sigma \in \Sigma \end{array}$$

where $Pr^\sigma_s(\psi) = Pr^\sigma_s(\{\pi \in Path_s \mid \pi \models_\sigma \psi\})$, and $Exp^\sigma_s(Z^r_{F\phi})$ denotes the expectation of the random variable $Z^r_{F\phi} : Path_s \to \mathbb{Q}_{\geq 0}$ under scheduler σ with

respect to the probability measure Pr_s^σ,

$$Z^r_{F\phi}(\pi) = \begin{cases} \infty & \text{if } \pi[i] \not\models \phi \text{ for all } i \in \mathbb{N} \\ \sum_{i=0}^{min\{j|\pi[j]\models\phi\}-1} r(\pi[i]) & \text{otherwise} \end{cases}$$

Moreover, for a path π in M the satisfaction relation $\models_\sigma$ is defined by:

$$\begin{aligned} \pi &\models_\sigma X\phi \quad \text{iff} \quad \pi[1] \models \phi \\ \pi &\models_\sigma \phi_1 U \phi_2 \quad \text{iff} \quad \exists j \geq 0 : \pi[j] \models \phi_2 \wedge (0 \leq k < j : \pi[k] \models \phi_1) \end{aligned}$$

From now onwards we will omit σ from the satisfaction relation and write $\models$ when the meaning is clear from the context.

Operator P and E. Before discussing the model checking algorithm, we expand on the semantics of P and E defined above, showing how queries over all schedulers reduce to reasoning over infimum and supremum over all schedulers. This corresponds to computing $\inf_{\sigma\in\Sigma} Pr_s^\sigma(\psi)$ and $\sup_{\sigma\in\Sigma} Pr_s^\sigma(\psi)$.

In order to simplify the technical details, we use the symbol $\lceil$ to denote open and closed lower bounds, $\lceil \in \{[, (\}$ and we use $\rceil$ to denote open and closed upper bounds, $\rceil \in \{],)\}$. Accordingly, we use the symbol $\not\bowtie$ to denote "greater than" inequality symbols, $\not\bowtie \in \{\geq, >\}$, and we use the symbol $\not\bowtie$ to denote "less than" inequality symbols, $\not\bowtie \in \{\leq, <\}$, where $\not\bowtie = \geq$, $\not\bowtie = >$, $\not\bowtie = \leq$ and $\not\bowtie = <$.

From the equivalence result on PCTL formulae discussed by Baier and Katoen [BK08, Ch.10], we know that the following holds:

$$P_{\lceil p_1,p_2\rceil}(\psi) \equiv P_{\not\bowtie p_1}(\psi) \wedge P_{\not\bowtie p_2}(\psi)$$

We are interested in computing the *minimum* and the *maximum* probabilities and expected values. By the result in [BK08, Ch. 10], we know that there exist memoryless schedulers $\sigma_{\min}$ and $\sigma_{\max}$ that minimise and maximise, respectively, the probabilities of eventually reaching a state that satisfies ϕ:

$$Pr_s^{\sigma_{\min}}(F\phi) = \inf_{\sigma\in\Sigma} Pr_s^\sigma(F\phi)$$

$$Pr_s^{\sigma_{\max}}(F\phi) = \sup_{\sigma\in\Sigma} Pr_s^\sigma(F\phi)$$

This holds for every state s. In particular we will have:

$$s \models P_{\bowtie p}(\psi) \Leftrightarrow Pr_s^{\sigma_{\min}}(\psi) \bowtie p \text{ for } \bowtie \in \{\geq, >\}$$

$$s \models P_{\bowtie p}(\psi) \Leftrightarrow Pr_s^{\sigma_{\max}}(\psi) \bowtie p \text{ for } \bowtie \in \{\leq, <\}$$

Similar reasoning holds for the operator E, where we are interested in computing the minimum and the maximum expected cost values over all schedulers.

From [FKNP11] we know that there exist memoryless schedulers $\sigma_{\min}$ and $\sigma_{\max}$ that *minimise* and *maximise*, respectively, the expected cumulative reward of reaching a state that satisfies ϕ:

$$Exp_s^{\sigma_{\min}}(Z^r_{F\phi}) = \inf_{\sigma} Exp_s^{\sigma}(Z^r_{F\phi})$$

$$Exp_s^{\sigma_{\max}}(Z^r_{F\phi}) = \sup_{\sigma} Exp_s^{\sigma}(Z^r_{F\phi})$$

In particular, we can write:

$$s \models E^r_{\bowtie x}(F\phi) \Leftrightarrow Exp_s^{\sigma_{\min}}(Z^r_{F\phi}) \bowtie x \text{ for } \bowtie \in \{\geq, >\}$$

$$s \models E^r_{\bowtie x}(F\phi) \Leftrightarrow Exp_s^{\sigma_{\max}}(Z^r_{F\phi}) \bowtie x \text{ for } \bowtie \in \{\leq, <\}$$

Quantitative extension of rPCTL. The operators P and E express qualitative properties, i.e., they validate whether or not the probability and expected cost of the paths satisfying the formula meet the given bounds, respectively. For instance, the property "is the probability of reaching a state satisfying ϕ_2 while passing through states satisfying ϕ_1 greater than or equal to 0.7?" can be express with the formula $P_{[0.7,1]}(\phi_1 U \phi_2)$. Even though such queries are qualitative, however the model checking algorithm is solving the corresponding quantitative formulation. Thus, the logic can be extended with quantitative properties such as $P_{min=?}(\psi), P_{max=?}(\psi), R_{min=?}(\psi)$ and $R_{max=?}(\psi)$, and compute the minimum and the maximum values of probability and expected cost, respectively. For instance, the formula $P_{max=?}(Fsuccess)$ will compute the maximum probability of eventually reaching successful states.

Remark 2 (Extending E.) *According to the semantics the expected reward operator returns infinity (∞) when the set of target states $T \subseteq S$ is not reached for some path π, and this is what we will consider in the following part of the dissertation. However, inspired by the different forms of the reward formula defined by Baier and Katoen [BK08, Ch.10], the expected reward operator can be extended with alternative values for the reward when T is not reached. In particular, the expected reward $*$ can be 0 ($* = 0$) when T is not reached, or it can be equal to the cumulative reward c ($* = c$) along the whole path.*

The general definition of the random variable $Z^r_{F^*\phi}(\pi)$ for different values for the reward, when T is not reached, is:

$$Z^r_{F\phi}(\pi) = \begin{cases} g(*) & \text{if } \pi[i] \not\models \phi \text{ for all } i \in \mathbb{N} \\ \sum_{i=0}^{min\{j|\pi[j]\models\phi\}-1} r(\pi[i]) & \text{otherwise} \end{cases}$$

where $g(*) = *$ if $* \in \{0, \infty\}$, and $g(*) = \sum_{j \in N} r(\pi[j])$ if $* = c$.

7.2.2 Model Checking rPCTL

Given a model of the system defined by an MDP $\mathcal{M}$ and a property specified by an rPCTL state formula ϕ, model checking verifies whether the model $\mathcal{M}$ satisfies the formula ϕ. For verification of the formula ϕ the model checking algorithm automatically determines the states of $\mathcal{M}$ that satisfies ϕ. The algorithm recursively traverses the parse tree of ϕ in a bottom-up fashion, where the internal nodes of the parse tree represents the sub-formulae of ϕ and the leaves correspond to the constant *true* or an atomic proposition $a \in AP$. For each sub-formula ϕ' of ϕ, the algorithm recursively computes the set of satisfying states $Sat(\phi') = \{s \in S \mid s \models \phi'\}$.

For atomic propositions and logical connectives the model checking algorithm is as follows:

$$\begin{array}{lcl} Sat(true) & = & S \\ Sat(a) & = & \{s \in S \mid a \in L(s)\} \\ Sat(\neg\phi) & = & S \backslash Sat(\phi) \\ Sat(\phi_1 \wedge \phi_2) & = & Sat(\phi_1) \cap Sat(\phi_2) \end{array}$$

Model checking the operators P and E is not trivial. We need to compute the probability and expected cost values over all schedulers and then check them against the given bounds. Below we discuss the model checking algorithm for each operator separately.

The complexity of PCTL model checking over MDPs is linear in the size of the formula ϕ and polynomial in the size of the state space S.

7.2.2.1 Model Checking the Probabilistic Operator P

For model checking the formulas $P_J(\psi)$ we need to compute the probability of satisfying the formula ψ for each state s, and check whether or not the interval J is satisfied for all schedulers $\sigma \in \Sigma$. This is reduced to the computation of the minimum and maximum probabilities depending on the bound. Thus, we have:

$$Sat(P_{\bowtie p}(\psi)) = \{s \in S \mid Pr_s^{min}(\psi) \bowtie p\} \text{ for } \bowtie \in \{\geq, >\}$$

$$Sat(P_{\bowtie p}(\psi)) = \{s \in S \mid Pr_s^{max}(\psi) \bowtie p\} \text{ for } \bowtie \in \{\leq, <\}$$

where $Pr_s^{min}(\psi) = \inf_{\sigma \in \Sigma} Pr_s^{\sigma}(\psi)$ and $Pr_s^{max}(\psi) = \sup_{\sigma \in \Sigma} Pr_s^{\sigma}(\psi)$.

The computation is done for each path formula separately. For the sake of simplicity, we denote $Pr_s^{min}(\psi)$ by x_s^{min} and $Pr_s^{max}(\psi)$ by x_s^{max}.

The operator Next. We first consider the operator next. For

$$x_s^{min} = Pr_s^{min}(X\phi)$$

the following equation system is satisfied:

$$x_s^{min} = \begin{cases} 0 & \text{if} \quad s \in S_{\circledcirc} \\ \sum_{s' \in Sat(\phi)} P(s,s') & \text{if} \quad s \in S_P \\ \min_{l \in \alpha} \begin{cases} 1 & \text{if } T(s,l) \in Sat(\phi) \\ 0 & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

In case the system had multiple solutions, we would consider the least one as we are interested in the minimum probability. However, from the results discussed in [FKNP11], it follows that the above equation system has a unique solution.

Similarly, for

$$x_s^{max} = Pr_s^{max}(X\phi)$$

the following equation system is satisfied and has a unique solution:

$$x_s^{max} = \begin{cases} 0 & \text{if} \quad s \in S_{\circledcirc} \\ \sum_{s' \in Sat(\phi)} P(s,s') & \text{if} \quad s \in S_P \\ \max_{l \in \alpha} \begin{cases} 1 & \text{if } T(s,l) \in Sat(\phi) \\ 0 & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

The operator Until. For computing the operator until, we first determine the sets of states satisfying the formula with probability 1 and 0. The set $Sat(\phi_2)$ contains the states satisfying the formula ϕ_2 with probability 1, while the set $S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr_s^\sigma(\phi_1 U \phi_2) = 0\}$ denotes all states from which it is not possible to reach a state satisfying ϕ_2 by following the states satisfying ϕ_1. The computation of the set S^0_{min} is done through the breadth-first graph traversal algorithm presented in Algorithm 3. We refer the reader to [FKNP11] for more details about the algorithm and the computation.

Algorithm 3 : Computation of the set S^0_{min}

Input: MDP $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$, $\psi = \phi_1 U \phi_2$
Output: the set $S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr_s^\sigma(\phi_1 U \phi_2) = 0\}$

$R \leftarrow Sat(\phi_2)$; $R' \leftarrow \emptyset$;
while $R \neq R'$ **do**
  $R' \leftarrow R$;
  $R \leftarrow R' \cup \{s \in Sat(\phi_1) \mid (s \in S_P \wedge (\exists s' \in R' : P(s,s') > 0)) \vee$
    $(s \in S_A \wedge (\forall l \in \alpha(s) : (\exists s' \in R' : T(s,l) = s')))\}$;
end while
return $S \backslash R$;

For each set of states and for

$$x_s^{min} = Pr_s^{min}(\phi_1 U \phi_2)$$

we have:

$$x_s^{min} = \begin{cases} 1 & \text{if} \quad s \in Sat(\phi_2) \\ 0 & \text{if} \quad s \in S^0_{min} \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{min} & \text{if} \quad s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \\ \min_{l \in \alpha} x^{min}_{T(s,l)} & \text{if} \quad s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \end{cases}$$

As before, we are interested in the minimum solution of the system. However, here as well the above equation system has a unique solution.

For computing

$$x_s^{max} = Pr_s^{max}(\phi_1 U \phi_2)$$

we have:

$$x_s^{max} = \begin{cases} 1 & \text{if} \quad s \in Sat(\phi_2) \\ 0 & \text{if} \quad s \in S^0_{max} \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{max} & \text{if} \quad s \in S_P \backslash (S^0_{max} \cup Sat(\phi_2)) \\ \max_{l \in \alpha} x^{max}_{T(s,l)} & \text{if} \quad s \in S_A \backslash (S^0_{max} \cup Sat(\phi_2)) \end{cases}$$

where $S^0_{max} = \{s \in S \mid \forall \sigma \in \Sigma : Pr_s^{\sigma}(\phi_1 U \phi_2) = 0\}$ and the computation is presented in Algorithm 4. Analogously, the following system has a unique solution.

Algorithm 4 : Computation of the set S^0_{max}

Input: MDP $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$, $\psi = \phi_1 U \phi_2$
Output: the set $S^0_{min} = \{s \in S \mid \forall \sigma \in \Sigma : Pr_s^{\sigma}(\phi_1 U \phi_2) = 0\}$

$R \leftarrow Sat(\phi_2)$; $R' \leftarrow \emptyset$;
while $R \neq R'$ **do**
  $R' \leftarrow R$;
  $R \leftarrow R' \cup \{s \in Sat(\phi_1) \mid (s \in S_P \wedge (\exists s' \in R' : P(s,s') > 0)) \vee$
    $(s \in S_A \wedge (\exists l \in \alpha(s) : (\exists s' \in R' : T(s,l) = s')))\}$;
end while
return $S \backslash R$;

7.2.2.2 Model Checking the Reward Operator E

Similarly to the case of probabilistic operator, for computing the expected cost value over all schedulers we compute the minimum and maximum values based on the bound.

$$Sat(E^r_{\bowtie x}(F\phi)) = \{s \in S \mid Exp_s^{min}(Z^r_{F\phi}) \bowtie x\} \text{ for } \bowtie \in \{\geq, >\}$$

$$Sat(E^r_{\bowtie x}(F\phi)) = \{s \in S \mid Exp^{max}_s(Z^r_{F\phi}) \bowtie x\} \text{ for } \bowtie \in \{\leq, <\}$$

where $Exp^{min}_s(Z^r_{F\phi}) = \inf_{\sigma\in\Sigma} Exp^{\sigma}_s(Z^r_{F\phi})$ and $Exp^{max}_s(Z^r_{F\phi}) = \sup_{\sigma\in\Sigma} Exp^{\sigma}_s(Z^r_{F\phi})$.

The computation of

$$x^{min}_s = Exp^{min}_s(Z^r_{F\phi})$$

can be done by solving the following equation system:

$$x^{min}_s = \begin{cases} 0 & \text{if} \quad s \in Sat(\phi) \\ \infty & \text{if} \quad s \in S^{<1}_{min} \\ r(s) + \sum_{s'\in S} P(s,s') \cdot x^{min}_{s'} & \text{if} \quad s \in S_P \backslash (S^{<1}_{min} \cup Sat(\phi)) \\ r(s) + \min_{l\in\alpha} x^{min}_{T(s,l)} & \text{if} \quad s \in S_A \backslash (S^{<1}_{min} \cup Sat(\phi)) \end{cases}$$

where $S^{<1}_{min} = \{s \in S \mid \forall\sigma \in \Sigma : Pr^{\sigma}_s(F\phi) < 1\}$. The set $S^{<1}_{min}$ is the set of states for which the probability of reaching the states satisfying ϕ is strictly less than 1 for all schedulers. We omit the detail computation of the set $S^{<1}_{min}$ and refer the reader to [FKNP11].

We are interested in the minimum value of the above system. Based on the discussion in [FKNP11], for the equation system above to have a unique solution we need to modify the MDP by removing states with self-loop and zero reward.

For the computation of the maximum value we replace *min* with *max* in the system above and use the set $S^{<1}_{max} = \{s \in S \mid \exists\sigma \in \Sigma : Pr^{\sigma}_s(F\phi) < 1\}$.

7.3 Reward Operators for rPCTL

We have seen so far a classic expected reward operator for rPCTL. It is worth noticing, however, that different operators have been investigated to compute different kinds of rewards. For instance, Forejt et al. [FKNP11] define PCTL with new operators that are used to evaluate *instantaneous* and *cumulative* expected reward, while operators for expressing *long-run* and *accumulated* expected reward are presented in [AHK03].

Nevertheless, these extensions do not reason about the cumulative reward along the path, i.e., they cannot express properties such as "the probability of reaching the success state is at least 0.7, while the cumulative reward is at most 50". To overcome this limitation, rPCTL has been extended with the *reward-bounded until* path operator [BK08, Ch.10], [BHHK00, CFK$^+$13a]. The operator verifies if the cumulative reward along a path satisfying the property meets the given reward bound.

All these operators evaluate the *expected* reward of the model, i.e., the sum of the rewards along the path multiplied with probabilities. To the best of our knowledge, they do not evaluate the exact reward, i.e., the sum of the rewards without considering the probability measure. As in the attack and

attack-defence scenarios the exact cost is one of the properties of interest, in Ch. 8 we extend rPCTL with an operator that evaluates the exact cost of the model.

7.4 Verification of Attack Trees though DTMCs

Attack trees in probabilistic setting present the purely probabilistic behaviour of the system. In this setting, where only probabilistic aspects are essential, the connection between attack trees and probabilistic models established in the area of probabilistic model checking becomes natural. In particular, we study the relation between attack trees and DTMCs, and propose an approach based on DTMCs for evaluating the probability of an attack scenario.

In order to verify the probabilistic properties of an attack scenario we use probabilistic model checking. First we present the transformation of an attack tree to a probabilistic model (DTMC), and then discuss how probabilistic model checking can be used to validate the properties of interest.

In the following we consider the attack tree formalism presented in Ch. 3, where the sub-trees are combined either conjunctively or disjunctively. The set of basic actions of an attack tree t is denoted by Act. For evaluating the probabilistic behaviour of the system, we associate with each basic action $a \in Act$ a success probability $p(a)$ in case the attacker performs the action, $p : Act \to [0, 1]$. The probability of the failure of the action a will be $1 - p(a)$.

7.4.1 Construction of DTMCs

The transformation of an attack tree t to a DTMC is done with the help of transition matrices. First, we transform each basic action of an attack tree to a DTMC and compute the corresponding transition matrix. Then, we construct the product automata of DTMCs by combining the transition matrices by means of Kronecker product in a bottom-up fashion according to the structure of the tree. Finally, having a transition matrix for the root of the tree, we construct a DTMC corresponding to t.

The construction of a DTMC $\mathcal{D}$ from a basic action $a \in Act$ is presented in Algorithm 5. The construction is done as follows. For each basic action a we create three new states s_0, s_1 and s_2. The initial state s_0 of the DTMC $\mathcal{D}$ corresponds to performing a. If a is performed successfully, the state s_0 moves with probability $p(a)$ to a success state s_1. Otherwise, it moves with probability $1-p(a)$ to a failure state s_2. We associate with each state a Boolean label, such that a state has label *tt* if an action a performed successfully in that state, and it has label *ff* otherwise. Thus, we label the state s_1 with *tt* and the state s_2 with *ff*. The initial state has label *ff*.

Figure 7.1 shows graphically the constructed DTMC $\mathcal{D}$ from a basic action a. The state s_0 is the initial state labelled with *ff*. It has two outgoing transitions.

Algorithm 5 : Algorithm for constructing a DTMC from a basic action.

Input: a basic action $a \in Act$
Output: DTMC $\mathcal{D} = (S, s_0, P, AP, L)$

new states s_0, s_1, s_2;
$P(s_0, s_1) = p(a)$; $P(s_0, s_2) = 1 - p(a)$;
$P(s_1, s_1) = 1$; $P(s_2, s_2) = 1$;
$L(s_0) = \mathit{ff}$; $L(s_1) = \mathit{tt}$; $L(s_2) = \mathit{ff}$;

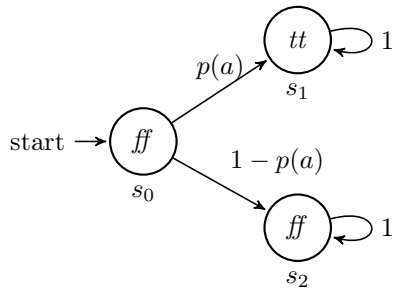


Figure 7.1: Markov chain constructed from a basic action a.

With the outgoing transition labelled with probability $p(a)$ the action succeeds and moves to the state s_1, while with the transition labelled with probability $1-p(a)$ it fails and moves to the state s_2. The success state s_1 is labelled with tt, while the failure state s_2 is labelled with ff. The states s_1 and s_2 are absorbing states, meaning that $Post(s_1) = \{s_1\}$ and $Post(s_2) = \{s_2\}$.

The transition matrix P and the labelling vector L of the constructed DTMC $\mathcal{D}$ have the following form:

$$P = \begin{bmatrix} 0 & p(a) & 1-p(a) \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad L = \begin{pmatrix} \mathit{ff} \\ \mathit{tt} \\ \mathit{ff} \end{pmatrix}$$

So far we have described how to transform each basic action to a DTMC. We combine DTMCs by computing a product automata. The product automata of two DTMCs $\mathcal{D}_1, \mathcal{D}_2$ corresponding to sub-trees t_1, t_2, and with transition matrices P_1, P_2 and labelling vectors L_1, L_2 is constructed as follows. First, we construct the transition matrix P and the labelling vector L of the product automata. The construction is based on the operator combining sub-trees t_1 and t_2:

- If $t = \&_\wedge(t_1, t_2)$, then $P = P_1 \otimes P_2$ and $L = L_1 \wedge L_2$
- If $t = \&_\vee(t_1, t_2)$, then $P = P_1 \otimes P_2$ and $L = L_1 \vee L_2$

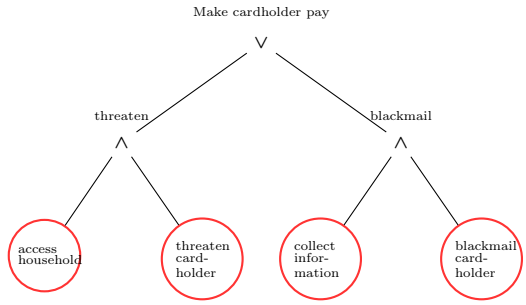


Figure 7.2: A fragment of the attack tree for forcing the cardholder to pay, from Ch. 3.

Table 7.1: The values of probability for the basic actions of the example.

Name of the Node	**Success probability**
Access household	0.6
Threaten cardholder	0.3
Collect information	0.55
Blackmail cardholder	0.2

where the labelling vector is combined by using point-wise conjunction (respectively disjunction) that applies the conjunction operator (respectively disjunction operator) element by element such that $L(ij) = L_1(i) \wedge L_2(j)$ (respectively $L(ij) = L_1(i) \vee L_2(j)$).

From the resulting transition matrix P and labelling vector L we construct the DTMC corresponding to the tree t by considering only reachable states. At each step of the construction we remove outgoing edges from the absorbing states by making them terminal states, meaning that $Post(s) = \emptyset$.

The constructed DTMC can be further simplified with the help of bisimulation reduction. A bisimilar DTMC can be computed by means of partition refinement, which has a complexity of $O(nt)$ for a Markov Chain with n states and t transitions [BK08, Buc08, Buc99].

Example. We will present our approach on a fragment of the attack tree discussed in Ch. 3. Consider the attack tree given in Figure 7.2, where the goal of the attacker is to steal money from the cardholder by forcing him/her to pay. The goal can be divided into two sub-goals: threatening or blackmailing. For a successful threatening the attacker should threaten the cardholder and access the household. In order to succeed in blackmailing the attacker should collect necessary information and blackmail the cardholder. The estimated success probability of basic actions is given in Table 7.1.

First, we translate each basic action to a DTMC having the form presented in Figure 7.1, where edges are labelled with corresponding probabilities. Using

our bottom-up approach we construct a DTMC for the root.

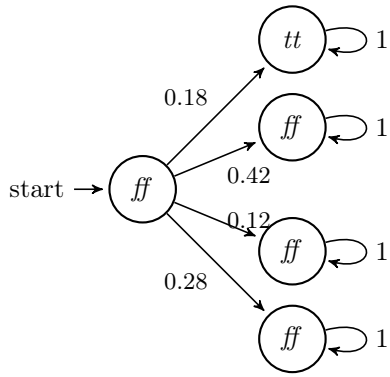
Let us first construct a DTMC for the node "threaten". The corresponding transition matrix P_{th}, which is the Kronecker product of transition matrices P_{ah} and P_{tc}, corresponding to the basic actions "access household" and "threaten cardholder", is:

$$P_{th} = P_{ah} \otimes P_{tc} = \left[\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0.18 & 0.42 & 0 & 0.12 & 0.28 \\ 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.6 & 0 & 0 & 0.4 \\ \hline 0 & 0 & 0 & 0 & 0.3 & 0.7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 0.7 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

The corresponding labelling vector L_{th} is the point-wise conjunction of the labelling vectors of the basic actions "access household" and "threaten cardholder".

$$L_{th} = L_{ah} \wedge L_{tc} = \left(\begin{array}{c} ff \\ ff \\ ff \\ \hline ff \\ tt \\ ff \\ \hline ff \\ ff \\ ff \end{array}\right)$$

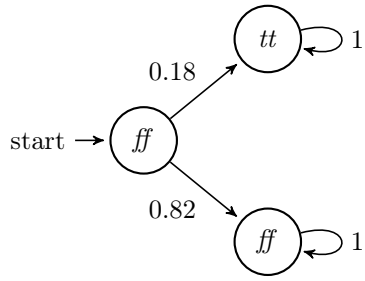
Considering only the reachable states of the transition matrix P_{th} and the labelling vector L_{th}, the DTMC for the node "threaten" is:



The new transition matrix P_{th}, derived from the figure above, is:

$$P_{th} = \begin{bmatrix} 0 & 0.18 & 0.42 & 0.12 & 0.28 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Observe that we can reduce the DTMC to the following one, which is bisimilar to the one above:



The corresponding transition matrix P_{th} and labelling vector L_{th} is:

$$P_{th} = \begin{bmatrix} 0 & 0.18 & 0.82 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{and} \qquad L_{th} = \begin{pmatrix} ff \\ tt \\ ff \end{pmatrix}$$

The DTMC for the node "blackmail" is constructed in the same way and the transition matrix P_{bl} have a similar structure with the appropriate probabilities.

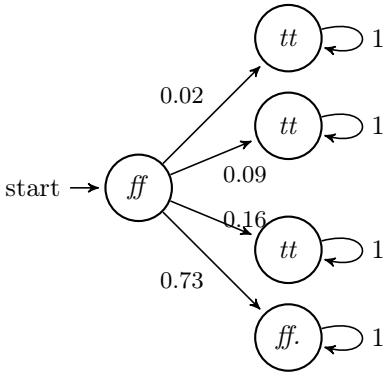
The transition matrix for the root of the tree is constructed through the Kronecker product of the matrices P_{th} and P_{bl}. Each of them are 5×5 matrices and as a result we get a 25×25 matrix and the corresponding labelling vector for the root. However, through bisimulation reduction we can reduce the size of the so obtained DTMC. In this case, the size of the transition matrix P_{th} is 3×3 as well as the size of the transition matrix P_{bl}, and the result of the Kronecker product of $P = P_{th} \otimes P_{bl}$ is a 9×9 matrix instead of a 25×25 matrix.

$$P = P_{th} \otimes P_{bl} = \left[\begin{array}{ccc|ccc|ccc} 0 & 0 & 0 & 0 & 0.02 & 0.16 & 0 & 0.09 & 0.73 \\ 0 & 0 & 0 & 0 & 0.18 & 0 & 0 & 0.82 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.18 & 0 & 0 & 0.82 \\ \hline 0 & 0 & 0 & 0 & 0.11 & 0.89 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.11 & 0.89 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right]$$

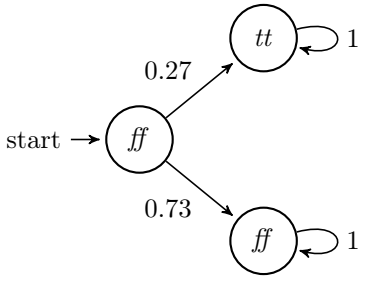
The labelling vector L for the root of the tree is:

$$L = L_{th} \vee L_{bl} = \begin{pmatrix} ff \\ tt \\ \underline{ff} \\ tt \\ tt \\ \underline{tt} \\ ff \\ tt \\ ff \end{pmatrix}$$

Similarly, considering only the reachable states, the DTMC corresponding to the tree in Figure 7.2 is:



Further simplifying the result, we obtain the following bisimilar DTMC:



7.4.2 Evaluation of DTMCs

In the previous section we have described a transformation of attack trees in probabilistic setting to DTMCs. It is worth observing that this procedure always leads to a DTMC with three states that contains the essence of any attack tree evaluation, showing that the goal at the root can be attained or not and

with what probabilities. The probabilistic evaluation is performed during the construction of a DTMC through the computation of the transition matrix P and the labelling vector L in each step, leading to a DTMC with the result at the root for the query "what is the maximum probability of a successful attack?".

Of course, properties that can be verified on such a DTMC are limited to the probability of success and failure at the root. In particular, the transformation leads to loose information about the internal nodes (sub-goals) and the leaves that enable the attack. Even though the method provides limited insights onto the model and its properties, it could be argued that it results in a very concise outcome, easy to communicate. In case when the main focus of the security analysis of an attack scenario is to reason about the probability of an attack and not the path leading to it, this method provides a simple visual representation of the query under investigation.

Chapter 8

Evaluation of Attack Trees through MDPs

In Sect. 7.4 we have studied attack trees where basic actions are associated only with the probability of success and presented the connection between attack trees and DTMCs. We shall now turn our attention to other attributes of attack trees, for probability is just one of various security attributes to be found in the literature and in the practice.

In this chapter we extend the model of attack tree with a cost attribute. The attribute describes the cost of performing a basic action and allows to evaluate properties such as "is there a cheapest way to attack the system?" and "what is the minimum cost of a successful attack?". Moreover, the extended model allows to represent an attack tree with multiple parameters, which leads to the analysis of security properties with various, possibly conflicting, objectives such as "what is the maximum success probability of an attack with cost at most 500?".

Attack scenarios with both probability and cost attributes express a combination of nondeterministic and probabilistic behaviour, i.e., an attacker has the nondeterministic choice of performing a basic action and paying the corresponding cost, while the performed basic action succeeds with a certain probability. In order to represent both choices, the nondeterministic choice for performing an action and the probabilistic choice for succeeding with a certain probability, we model the attack scenario as an MDP. We study the connection between these two models and define a translation from attack trees to MDPs.

In order to analyse attack scenarios formally and to compute the optimal

strategy of the attacker we employ probabilistic model checking. The probabilistic temporal logic rPCTL is then used to express a wide range of security properties. The logic allows to specify various properties related to the probability of certain events occurring, and an expected cost or reward measure associated with them.

The expected reward operator in rPCTL computes the expected value of the cumulative reward obtained before some set of target states is reached. Intuitively, the expected value is computed by multiplying the cost of the states by their probability of occurring along the paths reaching the target states, and the resulting products are summed. This operator is useful to analyse the expectation of certain events and allows to specify properties such as "what is the expected cost of an attack?". However, one might be interested in the exact cost computation of certain events occurring, i.e., computing the sum of the costs of states along the path satisfying the event. Exact cost can be used to evaluate the cost or time to execute a task and compare it to the available budget.

For specifying and verifying exact cost properties we extends rPCTL with an exact cost operator denoted by C. In our extended logic erPCTL we use a general notation of the cost-bounded until operator with any cost interval $I \in \mathbb{Q}_{\geq 0}$ and without step bound, i.e., the formula is satisfied when both the set of target states is reached and the cost interval is satisfied. We define the model checking algorithm erPCTL and demonstrate our developments on an example.

The chapter first presents the construction of MDPs from attack trees with multiple parameters in Sect. 8.1. The extended logic erPCTL and the model checking algorithm are defined in Sects. 8.2 and 8.3, respectively. The verification of security properties expressed in erPCTL is discussed in Sect. 8.4. We conclude and discuss future research directions in Sect. 8.5. This chapter is mainly based on [AN17].

8.1 From Attack Trees to Markov Decision Processes

In Sect. 7.4 we studied the probabilistic properties of an attack scenario. In the following, together with a probability attribute, we augment attack trees with the cost of performing basic actions. We associate with each basic action $a \in Act$ a cost c of performing a, $c : Act \to \mathbb{Q}_{\geq 0}$.

As noted above, an attack tree with probability and cost attributes encodes behaviour encompassing both probabilistic and nondeterministic features. Thus, we translate such an attack tree to an MDP. Before presenting the translation, it is worthwhile noticing that in an attack tree the order in which the basic actions are performed is not fixed. However, in the MDP this needs to be made

$\mathsf{construct}[t](A \cup \{a\}, \mathit{Done}, \mathit{Succ}) =$ new state $s \in S_A$ with:
$L(s) = \{a\}, T(s, Y) = s', T(s, N) = s''$ where:
$s' = \mathsf{construct}[t](A, \mathit{Done} \cup \{a\}, \mathit{Succ})$
$s'' = \mathsf{construct}[t](A, \mathit{Done}, \mathit{Succ})$
and: $r(s') = c(a), r(s'') = 0$

$\mathsf{construct}[t](\emptyset, \mathit{Done} \cup \{a\}, \mathit{Succ}) =$ new state $s \in S_P$ with:
$L(s) = \{a\}, P(s, s') = p(a), P(s, s'') = 1 - p(a)$ where:
$s' = \mathsf{construct}[t](\emptyset, \mathit{Done}, \mathit{Succ} \cup \{a\})$
$s'' = \mathsf{construct}[t](\emptyset, \mathit{Done}, \mathit{Succ})$
and: $r(s') = r(s'') = 0$

$\mathsf{construct}[t](\emptyset, \emptyset, \mathit{Succ}) =$ new state $s \in S_\circledcirc$ with:
$L(s) = \{\mathit{success}\}$ if $[\![t]\!](\mathit{Succ})$ and $\{\mathit{failure}\}$ otherwise, $P(s, s) = 1$

Table 8.1: The construction from an attack tree t to an MDP.

explicit. Since we assume that the basic actions of a tree are independent, we will also assume any linear order of the set of basic actions Act.

From an attack tree to an MDP. We construct an MDP $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ from an attack tree t according to Table 8.1, where $s_0 = \mathsf{construct}[t](\mathit{Act}, \emptyset, \emptyset)$. The set of states S is the disjoint union of sets $S_A, S_P, S_\circledcirc$, $S = S_A \uplus S_P \uplus S_\circledcirc$, while the set of actions is $\alpha = \{Y, N\}$. The transition functions P, T and the labelling function L are constructed according to Table 8.1, and a set of atomic propositions is $AP = \mathit{Act} \uplus \{\mathit{success}, \mathit{failure}\}$. Recall from Ch. 7 that $r : S \to \mathbb{Q}_{\geq 0}$ denotes the reward associated with the states of an MDP. The target state space S is exponential in the size of t.

The call $\mathsf{construct}[t](\mathit{Act}, \emptyset, \emptyset)$ of the recursive function $\mathsf{construct}$, defined in Table 8.1, constructs an MDP from t. The procedure first constructs all nondeterministic transitions of the target MDP, and then the probabilistic transitions.

Throughout the evaluation of the function we assume to have an attack tree t as a global parameter. At each step of the evaluation of $\mathsf{construct}[t](A, \mathit{Done}, \mathit{Succ})$ the first parameter A corresponds to the remaining set of attacker's basic actions that has still to be evaluated, the second parameter Done is the set of attempted actions, and the last parameter Succ is the set of attempted and succeeded actions. The construction function is structurally defined over the set of basic actions as explained below.

If the set of remaining actions contains the action a, $A = A' \cup \{a\}$, we create a nondeterministic state in the MDP labelled with a and with two outgoing

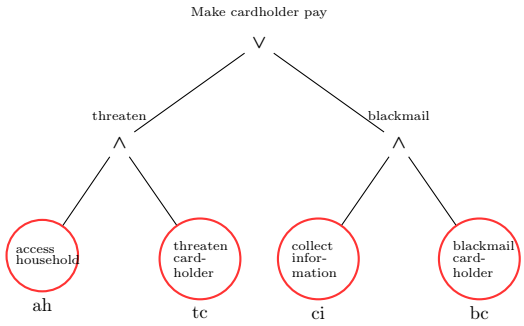


Figure 8.1: A fragment of the attack tree for forcing the cardholder to pay, from Ch. 3.

transitions. One transition corresponds to attempting a and is labelled with the action Y and cost $c(a)$, while the other transition corresponds to not attempting a and is labelled with the action N and cost 0. The successors of the state are constructed recursively, by calling $\mathsf{construct}[t](A', \mathit{Done} \cup \{a\}, \mathit{Succ})$ and $\mathsf{construct}[t](A', \mathit{Done}, \mathit{Succ})$, respectively.

If the set of remaining actions is empty, $A = \emptyset$, while the set of attempted actions contains the action a, $\mathit{Done} = \mathit{Done}' \cup \{a\}$, we create a probabilistic state labelled with a and with the outgoing transitions corresponding to the success and the failure of the attempted action a. We label these transitions with probabilities $p(a)$ and $1-p(a)$, and construct the successor of the state by calling $\mathsf{construct}[t](\emptyset, \mathit{Done}', \mathit{Succ} \cup \{a\})$ and $\mathsf{construct}[t](\emptyset, \mathit{Done}', \mathit{Succ})$, respectively.

If both the set of remaining actions and the set of attempted actions are empty, $A = \emptyset, \mathit{Done} = \emptyset$, then we are at the end of the procedure. We create a final state and label it with the result of the evaluation of t over the success of the basic actions, $[\![t]\!](\mathit{Succ})$, where $[\![t]\!]$ is the Boolean formula of which the tree t is a parse tree and the actions in the formula are tt if the actions are in the set Succ and ff otherwise.

Observe that the procedure builds finite MDPs as they contain no loop, hence all paths are finite.

Example. Let us introduce an example that we will develop throughout the chapter. We consider a fragment of the attack tree discussed in Chs. 3,7. The corresponding attack tree t is given in Figure 8.1. We label the leaves to refer to them easily.

The probability of success and cost values for the basic actions of the tree are given in Table 8.2, and we consider the following linearly ordered set for basic actions, $\mathit{Act} = \{\text{ah}, \text{tc}, \text{ci}, \text{bc}\}$.

Let us construct an MDP from the tree t displayed in Figure 8.1, by following

Table 8.2: Probabilities and costs for the basic actions in the example.

Label	Name of the Node	Probability	Cost
ah	access household	0.6	70
tc	threaten cardholder	0.3	30
ci	collect information	0.55	50
bc	blackmail cardholder	0.2	30

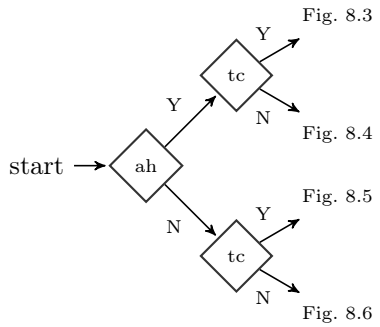


Figure 8.2: The MDP $\mathcal{M}_t$ constructed from the tree t. Due to the size of $\mathcal{M}_t$ we have split it into some figures.

the rules described in Table 8.1. First, all nondeterministic transitions are constructed, and then the probabilistic ones. The resulting MDP M_t is presented in Figure 8.2.

8.2 The Logic erPCTL for Attack Trees

In the following we introduce erPCTL (Probabilistic Computation Tree Logic with Exact Reward) for expressing probability as well as cost-related properties of MDPs. erPCTL is an extension of the temporal logic rPCTL. It allows to reason about the properties over cost measures such as probability within a cost bound or minimum cost of an execution.

Definition 8.1 (erPCTL Syntax) The syntax of the extended logic erPCTL is defined as follows:

$$\begin{aligned} \phi &::= true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_J(\psi) \mid E^r_{\bowtie x}(F\phi) \mid P_J(\psi \mid I) \mid C_I(\psi) \\ \psi &::= X\phi \mid \phi_1 U \phi_2 \end{aligned}$$

where $a \in AP$, $\bowtie \in \{\geq, >, \leq, <\}$, $J \subseteq [0,1]$ is a non-empty interval with rational bounds, $x \in \mathbb{Q}_{\geq 0}$, $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty interval with rational bounds, and $r : S \to \mathbb{Q}_{\geq 0}$ is a reward structure.

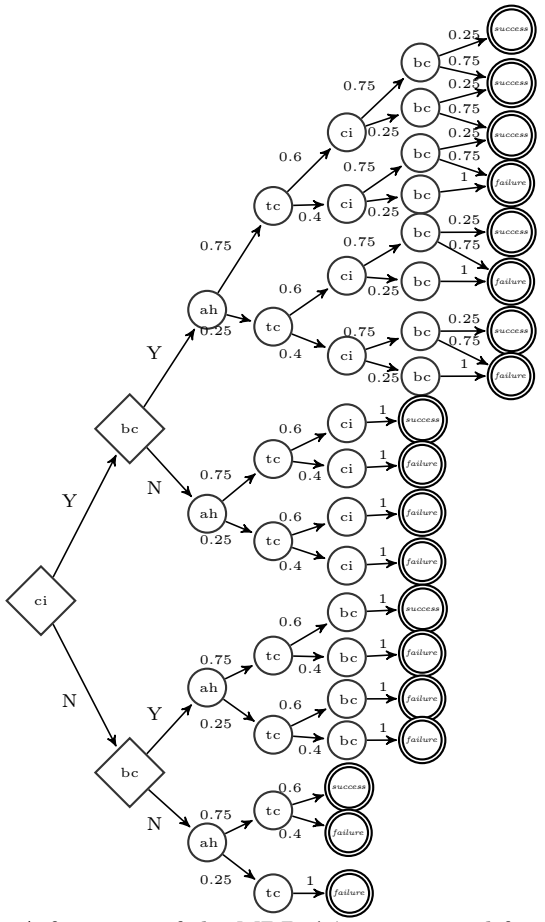


Figure 8.3: A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t.

Similarly to rPCTL, we differentiate between state formulae (ϕ) and path formulae (ψ). The operators inherited from rPCTL have the same semantics. The intuitive interpretation of the new operators is as follows. The *probabilistic operator with cost bound* $P_J(\psi \mid I)$ is used to evaluate the probability over the paths satisfying the cost bound I. Intuitively, a state s satisfies $P_J(\psi \mid I)$ if the probability that the formula ψ holds on the paths from s satisfying the cost bound I is in J. The *cost operator* $C_I(\psi)$ is used to evaluate the *exact cost* of the paths satisfying the formula ψ. Intuitively, a state s satisfies $C_I(\psi)$ if the cumulative cost of the paths satisfying the formula ψ is in I.

These operators allow us to check queries like "is the probability of an attack in a cost interval [300,540] smaller than or equal 0.85?", "what is the maximum probability of cheap attacks?" or "is the cost of all successful attacks greater

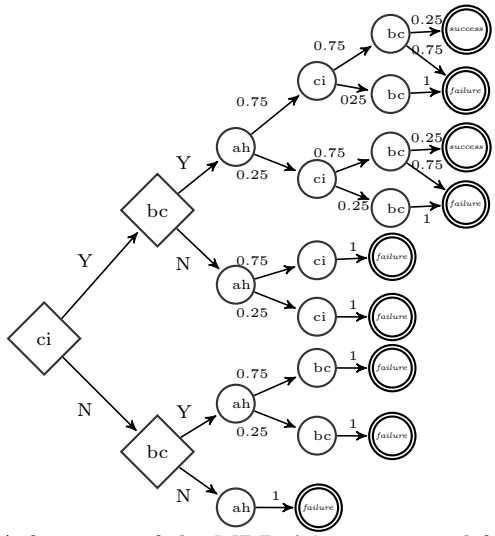


Figure 8.4: A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t.

than 300?".

For simplifying the technical developments, without loss of generality we move from rational numbers to integers for costs. We prove the following facts that we will use later in the chapter.

Fact 1 *For any finite set $Y \subseteq \mathbb{Q}$ there exists $N \in \mathbb{N}_{>0}$ such that $Y \subseteq \frac{\mathbb{Z}}{N} \subsetneq \mathbb{Q}$, where $\frac{\mathbb{Z}}{N}$ is a set of rational numbers expressed as fractions of the same non-zero denominator N.*

Proof. If the set is empty, $Y = \emptyset$, then the fact follows trivially, for the empty set is a subset of any set A, $\forall A : \emptyset \subseteq A$.

Assume the set contains $n > 0$ rational numbers, $Y = \{\frac{A_1}{B_1}, \cdots, \frac{A_n}{B_n}\}$ where $A_i \in \mathbb{Z}$, $B_i \in \mathbb{N}_{>0}$. We choose $N = B_1 \cdot \ldots \cdot B_n$ and re-write each number in the set as follows, $\frac{A_i}{B_i} = \frac{A_i \cdot \prod_{j \neq i} B_j}{N} \in \frac{\mathbb{Z}}{N}$. Thus, there exists N such that $Y \subseteq \frac{\mathbb{Z}}{N}$ and the fact follows.

□

Fact 2 *For all natural numbers $N \in \mathbb{N}_{>0}$ and sets $Y \subseteq \frac{\mathbb{Z}}{N}$ it holds that*

$$\inf(Y) \in \begin{cases} \{+\infty\} & \text{if } Y = \emptyset \\ \{-\infty\} & \text{if } Y \text{ is not bounded from below} \\ Y & \text{otherwise} \end{cases}$$

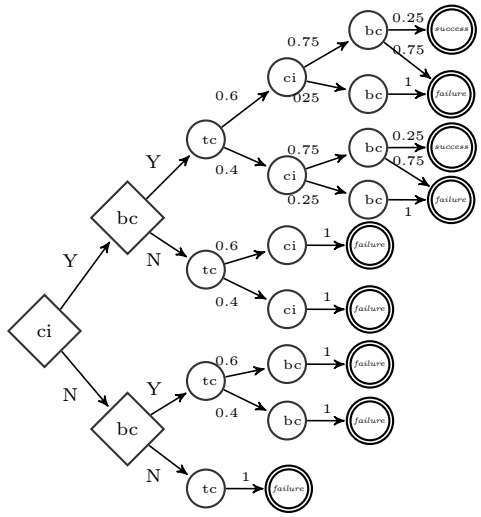


Figure 8.5: A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t.

$$\sup(Y) \in \begin{cases} \{-\infty\} & \text{if } Y = \emptyset \\ \{+\infty\} & \text{if } Y \text{ is not bounded from above} \\ Y & \text{otherwise} \end{cases}$$

Proof. We show the proof for the infimum. The case for the supremum is analogous.

Consider the set $Y \subseteq \frac{\mathbb{Z}}{N}$. The first two cases, when the set is empty or is not bounded from below, are trivial. Assume that Y is a non-empty set bounded from below. As N is a natural number, we can write $\inf(Y) = \frac{1}{N}\inf(Y \cdot N)$, where $Y \cdot N$ is a point-wise multiplication of the set Y by (a scalar) N resulting in a set of integers. We know that the infimum of a set of integers bounded from below is in the set. Thus, modifying Y as above we can conclude that $\inf(Y)$ is in Y.

□

Remark 3 *The new operators of erPCTL are treated similarly to the operators of rPCTL, reducing to the computation of infimum and supremum. For closed intervals, these values are in the interval. On the contrary, when the interval is open, it is not always the case that the infimum and the supremum are in the interval.*

The cost operator $C_I(\psi)$ computes the exact cost of reaching a state that satisfies ψ, where the cost values are added along the path without multiplying with probability, as opposed to the computation of the standard expected cost operator of rPCTL. Hence, with the help of Fact 2 and because $\frac{\mathbb{Z}}{N}$ is closed under addition, we can consider both open and closed cost intervals I.

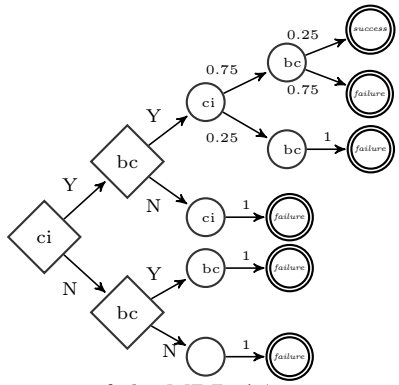


Figure 8.6: A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t.

A similar reasoning holds for the cost interval I in the probabilistic operator with cost bound $P_J(\psi \mid I)$, where we can consider both open and closed cost intervals. However, this is not the case for the probability interval J. In the operator $P_J(\psi \mid I)$ probabilities are multiplied along the path, hence we cannot use Fact 2 as $\frac{\mathbb{Z}}{N}$ is not closed under multiplication. Thus, we consider only closed intervals J.

As a future research direction we would like to investigate open intervals for probability in the operator $P_J(\psi \mid I)$ and study when the infimum and the supremum are in J.

Quantitative extension of erPCTL. Similarly to the probabilistic and expected reward operators of rPCTL, the operators $P_J(\psi \mid I)$ and $C_I(\psi)$ are validating whether or not the given bound is satisfied. They are not determining the actual probability and cost values. However, as the model checking algorithm is computing the values, we can extend the logic with quantitative operators such as $P_{min=?}(\psi \mid I)$, $P_{max=?}(\psi \mid I)$, $C_{min=?}(\psi)$ and $C_{max=?}(\psi)$.

The semantics of the propositional logic fragment and of probabilistic and reward formulae is defined as for rPCTL. Below we will discuss the semantics of the new operators $P_J(\psi \mid I)$ and $C_I(\psi)$.

8.2.1 Probabilistic Operator with Cost Bound $P_J(\psi \mid I)$

We propose the operator $P_J(\psi \mid I)$ for probability computation with cost bound, where ψ is a path formula, $J \subseteq [0,1]$ is a non-empty probability interval and $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty cost interval. The probability interval J can be only closed, while the cost interval I can be either open or closed. In order to simplify the technical developments, however, we do not consider intervals I of the form $(0, c]$ – if 0 is an extreme than it must fall in I.

Before defining the formal semantics of $P_J(\psi \mid I)$, let us introduce some useful notation. We define the semantics of each path formula with cost interval I as follows:

$$\begin{aligned} \pi \models_I X\phi \quad &\text{iff} \quad \pi[1] \models \phi \wedge cost(\pi[0\ 1]) \in I \\ \pi \models_I \phi_1 U \phi_2 \quad &\text{iff} \quad \exists j \geq 0 : \pi[j] \models \phi_2 \wedge (0 \leq k < j : \pi[k] \models \phi_1) \\ &\qquad \wedge cost(\pi[0 \cdots j]) \in I \end{aligned}$$

The semantics of the probabilistic operator with cost bound is as follows:

$$s \models P_J(\psi \mid I) \text{ iff } Pr_s^\sigma(\psi \mid I) \in J \text{ for all schedulers } \sigma \in \Sigma$$

where $Pr_s^\sigma(\psi \mid I) = Pr_s^\sigma\{\pi \in Path_s^\sigma \mid \pi \models_I \psi\}$.

Intuitively, $P_J(\psi \mid I)$ states that the probability of the paths starting from state s and satisfying the formula ψ and cost bound I meets the bounds given by J.

The operator $P_J(\psi)$ is treated as a special case of the operator $P_J(\psi \mid I)$. The two operators are equivalent if the cost interval is $[0, \infty)$:

$$P_J(\psi) \equiv P_J(\psi \mid [0, \infty))$$

As mentioned above, the semantics considers all possible schedulers, but we can rephrase it in terms of infimum and supremum. We know from [BK08] that the following equation holds:

$$P_{[p_1,p_2]}(\psi) \equiv P_{\geq p_1}(\psi) \wedge P_{\leq p_2}(\psi)$$

The result holds also in case of cost intervals on both sides of the equation:

$$P_{[p_1,p_2]}(\psi \mid I) \equiv P_{\geq p_1}(\psi \mid I) \wedge P_{\leq p_2}(\psi \mid I)$$

Similarly to the case of $P_J(\psi)$, we are interested in computing the *minimum* and the *maximum* probability values within given cost bounds.

$$s \models P_{\geq p}(\psi \mid I) \Leftrightarrow \inf_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \geq p$$

$$s \models P_{\leq p}(\psi \mid I) \Leftrightarrow \sup_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \leq p$$

where the clauses above hold thanks to the reduction of costs from $\mathbb{Q}$ to $\mathbb{Z}$ explained in Facts 1 and 2.

For the sake of conciseness, we define

$$Pr_s^{min}(\psi \mid I) = \inf_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I)$$

$$Pr_s^{max}(\psi \mid I) = \sup_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I)$$

8.2.2 Cost operator $C_I(\psi)$

We propose the operator $C_I(\psi)$ for exact cost computation, where ψ is a path formula and $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty cost interval. The cost interval I can be either open or closed.

Before defining the formal semantics of $C_I(\psi)$, let us introduce some useful notation. We define the cost of an infinite path $\pi = s_0 s_1 \cdots$ for each path formula, denoted by $cost(\pi, \psi)$, as follows:

$$\begin{aligned} cost(\pi, X\phi) &= \{cost(\pi[0\ 1]) \mid \pi[1] \models \phi\} \\ cost(\pi, \phi_1 U \phi_2) &= \{cost(\pi[0 \cdots k]) \mid \pi[k] \models \phi_2 \wedge (0 \leq i < k : \pi[i] \models \phi_1)\} \end{aligned}$$

When the path formula ϕ is not satisfied, then the set is empty. Otherwise, it contains the set of possible costs.

Fact 3 *For a path π, a cost interval I and a path formula ψ it holds that*

$$\pi \models_I \psi \Leftrightarrow \exists c \in cost(\pi, \psi) : c \in I$$

Fact 4 *For a path π and a path formula ψ it holds that*

$$\pi \models \psi \Leftrightarrow \pi \models_{[0,\infty)} \psi$$

The semantics of the cost operator is as follows:

$$s \models C_I(\psi) \text{ iff } \forall \sigma \in \Sigma : \forall \pi \in Path_s^\sigma : \forall c \in cost(\pi, \psi) : c \in I$$

Intuitively, $C_I(\psi)$ states that the exact (cumulative) cost of paths starting in state s and satisfying formula ψ under scheduler σ is in the interval I.

In order to verify the cost formula with a general cost interval, we reduce the problem to intervals with only lower and upper bounds according to the following equivalence result:

$$C_{\lceil c_1, c_2 \rceil}(\psi) \equiv C_{\not\bowtie c_1}(\psi) \wedge C_{\not\bowtie c_2}(\psi)$$

where $c_1 \leq c_2$, $c_1, c_2 \neq \infty$, $\not\bowtie\ \in \{\geq, >\}$, and $\not\bowtie \in \{\leq, <\}$.

Thus, to verify that the exact cost of each path satisfying the formula ψ is in the interval it is sufficient to verify that the exact cost of each path satisfying ψ meets the lower and upper bounds. Again, this problem can be reduced to verify that the infimum (respectively the supremum) cost meets the bound. Lemma 8.2 shows how such reduction is performed. Note that the cost computation is done over all schedulers and all paths.

Lemma 8.2

$$s \models C_{\bowtie c}(\psi) \Leftrightarrow (\inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, \psi)) \bowtie c, \quad for\ \bowtie \in \{\geq, >\}$$

$$s \models C_{\bowtie c}(\psi) \Leftrightarrow (\sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} \sup cost(\pi, \psi)) \bowtie c, \quad for \ \bowtie \in \{\leq, <\}$$

PROOF. Let us first proof the case of the infimum. The case when the set $cost(\pi, \psi)$ is empty or not bounded is straightforward. Assume that there exists at least one path satisfying ψ, hence $cost(\pi, \psi)$ is non-empty and bounded. Let us define $c_\sigma^{min} = \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, \psi)$.

It is sufficient to prove that for an MDP $\mathcal{M}$ with state space S there exists a memoryless scheduler σ_{min} such that for any $s \in S$

$$c_{\sigma_{min}} = \inf_{\sigma \in \Sigma} c_\sigma^{min}$$

We know that costs are non-negative rational numbers. We modify them into the form $\frac{\mathbb{Z}}{N}$ for $N \in \mathbb{N}_{>0}$. Since $\frac{\mathbb{Z}}{N}$ is closed under addition, we have that $cost(\pi, \psi) \subseteq \frac{\mathbb{Z}}{N}$. Based on Fact 2 we have that $c_\sigma^{min} \in \frac{\mathbb{Z}}{N}$. Then, it follows that there exists a scheduler σ_{min} such that $c_{\sigma_{min}} = \inf_{\sigma \in \Sigma} c_\sigma^{min}$ as the infimum of the set is in $\frac{\mathbb{Z}}{N}$ by Fact 2. The thesis follows.

The proof for the case of the supremum follows a different approach. Here we do not consider the cost of the shortest path in the underlying graph, but we take the maximum cost value. The poof is carried out by considering possible values for the supremum.

The case when the supremum is $-\infty$ is trivial as it can happen only when the set $cost(\pi, \psi)$ is empty and for the empty set the relation holds.

Assume the supremum is $+\infty$. We get this value only if a path meets a loop whose cost is greater than 0. In this case there must be a cost value greater than the upper bound c, and thus both sides of the relation cannot be satisfied.

Assume the supremum is finite. This case is similar to the corresponding one for the infimum. The proof follows by Fact 2: if the supremum is finite then the value must fall in the interval.

□

Duality. The cost formula $C_I(\psi)$ evaluates the cost over all schedulers and all paths. It is used to verify queries such as whether the cost of all successful attacks is within the given bounds. Such questions quantify universally over schedulers and paths. The dual questions, when we consider the existential quantifier, might be of interest as well. For instance, if the attacker has a limited budget, then he/she might want to know whether there is at least one cheap attack or a successful attack below his/her budget. Such "dual" queries can be phrased restoring to negation. For instance, verifying that the cost of satisfying ψ for at least one execution is at most c is equivalent to verifying the formula $\neg C_{>c}(\psi)$.

Semantically, for a given bound c, the following equivalence result holds:

$$\exists\sigma \in \Sigma : \exists\pi \in Path_s^\sigma : \exists c' \in cost(\pi,\psi) : c' \bowtie c \Leftrightarrow$$
$$\neg(\forall\sigma \in \Sigma : \forall\pi \in Path_s^\sigma : \forall c' \in cost(\pi,\psi) : \neg(c' \bowtie c))$$

8.3 Model Checking erPCTL

To verify properties defined in erPCTL we develop a model checking algorithm. The model checking algorithm for a erPCTL formula ϕ on an MDP $\mathcal{M}$ determines the states of $\mathcal{M}$ that satisfy ϕ. The algorithm is similar to the model checking algorithm for rPCTL. It recursively determines the set of satisfying states $Sat(\phi')$ for each sub-formula ϕ' of ϕ traversing the parse tree of ψ in a bottom-up manner.

For atomic propositions, logical connectives, the probabilistic operator and the reward operator the model checking algorithm is the same as for rPCTL. In the following we will discuss the algorithms of the new operators.

8.3.1 Model Checking the Operator $P_J(\psi \mid I)$

Similarly to the cases discussed above, the algorithm for the probabilistic operator with a cost bound is reduced to the computation of the minimum and the maximum values:

$$Sat(P_{\geq p}(\psi \mid I)) = \{s \in S \mid Pr_s^{min}(\psi \mid I) \geq p\}$$
$$Sat(P_{\leq p}(\psi \mid I)) = \{s \in S \mid Pr_s^{max}(\psi \mid I) \leq p\}$$

In the following we explain how to determine the minimum and maximum probability satisfying the formula in the cost interval, separately for each path formula ψ. The algorithm follows the development for the model checking probabilistic operator, presented in Sect. 7.2.2.1.

The operator Next ($\psi = X\phi$). First, we consider the operator next. For computing the minimum probability satisfying $X\phi$ in the cost interval I,

$$x_s^{min} = Pr_s^{min}(X\phi \mid I)$$

we are solving the following equations:

$$x_s^{min} = \begin{cases} 0 & \text{if} \quad s \in S_\circledcirc \\ \sum_{\substack{s' \in Sat(\phi) \\ r(s)+r(s') \in I}} P(s,s') & \text{if} \quad s \in S_P \\ \min_{l \in \alpha} \begin{cases} 1 & \text{if } T(s,l) \in Sat(\phi) \wedge (r(s) + r(T(s,l))) \in I \\ 0 & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

As the sets $S_{\circledcirc}, S_P, S_A$ are disjoint and we identified the set of states for which x_s^{min} equals 0, we can compute x_s^{min} as the unique solution of the system above.

Similarly, for computing the maximum probability satisfying $X\phi$ in the cost interval I,

$$x_s^{max} = Pr_s^{max}(X\phi \mid I)$$

the following equations have a unique solution:

$$x_s^{max} = \begin{cases} 0 & \text{if} \quad s \in S_{\circledcirc} \\ \sum_{\substack{s' \in Sat(\phi) \\ r(s)+r(s') \in I}} P(s,s') & \text{if} \quad s \in S_P \\ \max_{l \in \alpha} \begin{cases} 1 & \text{if } T(s,l) \in Sat(\phi) \wedge (r(s) + r(T(s,l))) \in I \\ 0 & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

The operator Until ($\psi = \phi_1 U \phi_2$). Let us now discuss the computation of $P_J(\psi \mid I)$ for the operator until. Again we are interested in computing $Pr_s^{min}(\psi \mid I)$ and $Pr_s^{max}(\psi \mid I)$. Before presenting the computation, it is worthwhile noticing that in many real-life scenarios the cost interval I has only an upper bound or a lower bound. Thus, we develop the computation in three different cases with respect to the cost bounds; only an upper bound $[0, c_2], [0, c_2)$ – recall that $I \subseteq \mathbb{Q}_{\geq 0}$, only a lower bound $[c_1, \infty), (c_1, \infty)$, *or* both bounds (cost interval) $[c_1, c_2], [c_1, c_2), (c_1, c_2]$ and (c_1, c_2).

Cases $I = [0, c_2]$ and $I = [0, c_2)$. Let us first look into the case of the interval $[0, c_2]$. Having only an upper bound c_2 for cost, the values of interest are $Pr_s^{min}(\phi_1 U \phi_2 \mid [0, c_2])$ and $Pr_s^{max}(\phi_1 U \phi_2 \mid [0, c_2])$. First, we define

$$x_s^{min}(\mathfrak{c}) = Pr_s^{min}(\phi_1 U \phi_2 \mid [0, \mathfrak{c}])$$

where $\mathfrak{c} \geq 0$ is the maximum amount that may be spent, where initially $\mathfrak{c} = c_2$. The algorithm follows the corresponding one for the probabilistic operator in rPCTL. The difference is that in each considered case (set of states) we examine the cost bound as well. For instance, for the set of state for which $Pr_s^{\sigma}(\phi_1 U \phi_2)$ is 1 we need to ensure that their costs are within the threshold $\mathfrak{c}$ ($r(s) \leq \mathfrak{c}$): instead, when their costs exceed the threshold ($r(s) > \mathfrak{c}$), these states are in the set for which $Pr_s^{\sigma}(\phi_1 U \phi_2)$ is 0. Thus, $x_s^{min}(\mathfrak{c})$ can be computed by solving the

Table 8.3: Computation of probabilities with upper cost bound, $Pr_s(\phi_1 U \phi_2 \mid [0, \mathfrak{c}])$.

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\leq \mathfrak{c}$	(1)	(2)	(3)	(4)
$> \mathfrak{c}$	(2)	(2)	(2)	(2)

following equation system:

$$x_s^{min}(\mathfrak{c}) = \begin{cases} 1 & \text{if} \quad s \in Sat(\phi_2) \wedge r(s) \leq \mathfrak{c} \quad (1) \\ 0 & \text{if} \quad s \in S^0_{min} \vee r(s) > \mathfrak{c} \quad (2) \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{min}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (3) \\ \min_{l \in \alpha} x_{T(s,l)}^{min}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (4) \end{cases}$$

where $S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr_s^\sigma(\phi_1 U \phi_2) = 0\}$.

To better understand the equation system, let us look into Table 8.3. The first row of the table illustrates the four disjoint set of states (1-4). The first column corresponds to the possible cost bounds. As we have an upper cost bound, all costs can be divided into two groups; those that are within the bound ($\leq \mathfrak{c}$) and those that are outside the bound ($> \mathfrak{c}$). Table 8.3 maps each possible combination of a set of states and cost bound for a state with a corresponding equation.

We consider the *minimum* value of the equation system in case of multiple solutions. However, the problem is similar to the stochastic shortest path problem, discussed in [DA98, BT91], and thus, the equation system has a unique solution.

For the computation of the maximum probability we change "min" to "max" in the system above. Analogously, the following system has a unique solution.

$$x_s^{max}(\mathfrak{c}) = \begin{cases} 1 & \text{if} \quad s \in Sat(\phi_2) \wedge r(s) \leq \mathfrak{c} \quad (1) \\ 0 & \text{if} \quad s \in S^0_{max} \vee r(s) > \mathfrak{c} \quad (2) \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{max}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_P \backslash (S^0_{max} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (3) \\ \max_{l \in \alpha} x_{T(s,l)}^{max}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_A \backslash (S^0_{max} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (4) \end{cases}$$

where $S^0_{max} = \{s \in S \mid \forall \sigma \in \Sigma : Pr_s^\sigma(\phi_1 U \phi_2) = 0\}$.

A correspondence holds between Table 8.3 and the equation system for computing the maximum probability as the one highlighted above for the minimisation case.

So far we have discussed the case for the closed interval $[0, c_2]$. For the computation of the open interval $[0, c_2)$ we change $\leq$ to $<$ and $>$ to $\geq$ in the equations above.

Cases $I = [c_1, \infty)$ and $I = (c_1, \infty)$. Let us present now the case when I has only a lower bound. We are interested in computing the probability of the paths that have cost greater than or equal to c_1. Having only a lower bound for the cost, the values of interest are $Pr_s^{min}(\phi_1 U \phi_2 \mid [c_1, \infty))$ and $Pr_s^{max}(\phi_1 U \phi_2 \mid [c_1, \infty))$.

We define

$$x_s^{min}(\mathfrak{c}) = Pr_s^{min}(\phi_1 U \phi_2 \mid [\mathfrak{c}, \infty))$$

where $\mathfrak{c} \in \mathbb{Q}$ is the required minimum amount to be spent.

First, we identify the set of states for which $Pr_s^{\sigma}(\phi_1 U \phi_2)$ is 0. The set with probability 0 is the same as for the probabilistic operator of rPCTL, that is

$$S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr_s^{\sigma}(\phi_1 U \phi_2) = 0\}$$

$$S^0_{man} = \{s \in S \mid \forall \sigma \in \Sigma : Pr_s^{\sigma}(\phi_1 U \phi_2) = 0\}$$

Observe that having a lower cost bound it might happen that a prefix of a path satisfies the formula but the required cost budget is not reached. We handle this situation by continuing the computation until we find a point where both the formula and the required cost budget are satisfied. Thus, for the set with probability 1 we check the satisfiability of the cost budget. If a state satisfies ϕ_2 ($s \in Sat(\phi_2)$) and the required cost amount $\mathfrak{c}$ ($r(s) \geq \mathfrak{c}$), then we stop the computation. Otherwise, we continue the iteration based on a type of the state.

$$x_s^{min}(\mathfrak{c}) = \begin{cases} 1 & \text{if} \quad s \in Sat(\phi_2) \wedge r(s) \geq \mathfrak{c} \quad (1) \\ 0 & \text{if} \quad s \in S^0_{min} \quad (2) \\ \sum_{s' \in S} P(s, s') \cdot x_{s'}^{min}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \vee \quad (3a) \\ & \quad (s \in S_P \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}) \quad (3b) \\ \min_{l \in \alpha} x_{T(s,l)}^{min}(\mathfrak{c} - r(s)) & \text{if} \quad s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \vee \quad (4a) \\ & \quad (s \in S_A \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}) \quad (4b) \end{cases}$$

We present Table 8.4 to associate each set of states and cost amount for a state with a corresponding equation. Observe that for the states in $Sat(\phi_2)$ and cost $< \mathfrak{c}$ there are two equations. The equations (3b) and (4b) correspond to the

Table 8.4: Computation of probabilities with lower cost bound, $Pr_s(\phi_1 U \phi_2 \mid [\mathfrak{c}, \infty])$.

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\geq \mathfrak{c}$	(1)	(2)	(3a)	(4a)
$< \mathfrak{c}$	(3b),(4b)	(2)	(3a)	(4a)

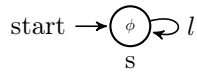


Figure 8.7: An MDP example.

continuation of the computation in case the formula is satisfied but the required cost amount is not reached.

In case of multiple solutions we consider the minimum value of the equation system above.

Remark 4 *Consider an MDP with one state $s \in S_A$ and $r(s) = 0$, like the one presented in Figure 8.7. We are interested in computing $Pr_s^{min}(true\, U \phi \mid [10, \infty))$. Checking the conditions of the equations above, we can see that the state s satisfies the condition (4b), as $s \in Sat(\phi)$ and $r(s) < \mathfrak{c}$. From the equation system we have that $x_s^{min}(\mathfrak{c}) = x_s^{min}(\mathfrak{c})$, and thus the system above has infinitely many solutions.*

For ensuring a unique solution of the equation system we can use the techniques described in [FKNP11, DA98]. The main idea is to modify the MDP by removing states with self-loop and zero cost. However, as the uniqueness of the solution is outside of the scope of this dissertation, we will not go into more details.

For the computation of the maximum probability we change "min" to "max" in the equations above and follow similar developments as for the computation of the minimum probability.

So far we have discussed the case for the closed interval $[c_1, \infty]$. The computation of the open interval $(c_1, \infty]$ is carried out by changing $\geq$ to $>$ and $<$ to $\leq$ in the system above.

General case $I = \lceil c_1, c_2 \rceil$. Let us now present the general case, where we have both lower and upper cost bounds, we are interested in computing $Pr_s^{min}(\phi_1 U \phi_2 \mid \lceil c_1, c_2 \rceil)$ and $Pr_s^{max}(\phi_1 U \phi_2 \mid \lceil c_1, c_2 \rceil)$, where $c_1 \leq c_2$.

Table 8.5: Computation of probabilities with upper and lower bounds, $Pr_s(\phi_1 U \phi_2 \mid [\mathfrak{c}', \mathfrak{c}''])$.

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\geq \mathfrak{c}'$ and $\leq c''$	(1)	(2)	(3a)	(4a)
$> \mathfrak{c}''$	(2)	(2)	(2)	(2)
$< \mathfrak{c}'$	(3b),(4b)	(2)	(3a)	(4a)

We define

$$x_s^{min}(\mathfrak{c}', \mathfrak{c}'') = Pr_s^{min}(\phi_1 U \phi_2 \mid [\mathfrak{c}', \mathfrak{c}''])$$

where $\mathfrak{c}' \in \mathbb{Q}$ is the required minimum amount to be spent and $\mathfrak{c}'' \geq 0$ is the maximum amount that may be spent. Similarly to previous cases, we examine the cost amount for each set of states. For instance, the states in the set with probability 1 should satisfy not only the formula ϕ_2 but also be in a cost interval $[\mathfrak{c}', \mathfrak{c}'']$, while the states above the cost interval ($r(s) > \mathfrak{c}''$) should be in the set with probability 0.

$$x_s^{min}(\mathfrak{c}', \mathfrak{c}'') = \begin{cases} 1 \quad \text{if} \;\; s \in Sat(\phi_2) \wedge r(s) \in [\mathfrak{c}', \mathfrak{c}''] & (1) \\ 0 \quad \text{if} \;\; s \in S^0_{min} \vee r(s) > \mathfrak{c}'' & (2) \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{min}(\mathfrak{c}' - r(s), \mathfrak{c}'' - r(s)) & \\ \qquad \text{if } (s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \wedge r(s) \leq \mathfrak{c}'') & (3a) \\ \qquad\quad \vee \; (s \in S_P \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}') & (3b) \\ \min_{l \in A} x_{T(s,l)}^{min}(\mathfrak{c}' - r(s), \mathfrak{c}'' - r(s)) & \\ \qquad \text{if } (s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \wedge r(s) \leq \mathfrak{c}'') & (4a) \\ \qquad\quad \vee \; (s \in S_A \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}') & (4b) \end{cases}$$

We present Table 8.5 to associate each set of states and cost amount for a state with a corresponding equation. Differently from the previous cases, here the cost values are divided into three groups; those that are inside the cost interval, those that are below the cost interval and those that are above the cost interval.

We consider the minimum solution of the equation system above. Like in the case with only lower bound, here as well the system above does not have a unique solution. Again, we can use the techniques presented in [FKNP11, DA98] and modify the MDP in order to compute the unique solution.

For the computation of the maximum probability we change "min" to "max" in the equations above and consider the maximum solution.

So far we have discussed the case for the close interval $[c_1, c_2]$. The computation of partially-opened, $(c_1, c_2], [c_1, c_2)$, or fully-opened, (c_1, c_2) intervals, is carried out as follows. When the lower bound is not included, we change $\geq$ to $>$ and $<$ to $\leq$, while when the upper bound is not included we change $\leq$ to $<$ and $>$ to $\geq$.

8.3.2 Model Checking the Operator $C_I(\psi)$

Let us now present the model checking algorithm for the operator $C_I(\psi)$. We need to compute the exact cost of the paths satisfying the formula ψ and check whether it is in I. The procedure reduces to the computation of the minimum and the maximum values depending on the bound:

$$Sat(C_{\bowtie c}(\psi) = \{s \in S \mid (\inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, \psi)) \bowtie c \;\text{ for }\; \bowtie \in \{\geq, >\}\}$$

$$Sat(C_{\bowtie c}(\psi) = \{s \in S \mid (\sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} \sup cost(\pi, \psi)) \bowtie c \;\text{ for }\; \bowtie \in \{\leq, <\}\}$$

In the following we explain how to determine the minimum and maximum cost, separately for each path formula ψ.

The operator Next ($\psi = X\phi$). We start with the computation of the minimum cost for the operator next. The minimum cost

$$y_s^{min} = \inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, X\phi)$$

for each state s can be computed by means of the following equations:

$$y_s^{min} = \begin{cases} +\infty & \text{if} \quad s \in S_{\circledcirc} \\ \min_{P(s,s')>0} \begin{cases} r(s) + r(s') & \text{if } s' \in Sat(\phi) \\ +\infty & \text{otherwise} \end{cases} & \text{if} \quad s \in S_P \\ \min_{l \in \alpha} \begin{cases} r(s) + r(T(s,l)) & \text{if } T(s,l) \in Sat(\phi) \\ +\infty & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

The equation system above has a unique solution.

Similarly, we compute the maximum cost

$$y_s^{max} = \sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} \sup cost(\pi, X\phi)$$

by means of the following equations:

$$y_s^{max} = \begin{cases} -\infty & \text{if} \quad s \in S_{\circledcirc} \\ \max_{P(s,s')>0} \begin{cases} r(s)+r(s') & \text{if } s' \in Sat(\phi) \\ -\infty & \text{otherwise} \end{cases} & \text{if} \quad s \in S_P \\ \max_{l \in \alpha} \begin{cases} r(s)+r(T(s,l)) & \text{if } T(s,l) \in Sat(\phi) \\ -\infty & \text{otherwise} \end{cases} & \text{if} \quad s \in S_A \end{cases}$$

The operator Until ($\psi = \phi_1 U \phi_2$). Similarly to the computations of the until operator for $P_J(\psi \mid I)$ the minimum cost of a path satisfying the formula $\phi_1 U \phi_2$ can be computed recursively. For computing the minimum cost of a path, we stop the first time ϕ_2 is satisfied, i.e., we compute the cost of the path $\pi = s_0 \cdots s_j$ where $j = \min\{j \mid \pi[j] \models \phi_2 \wedge (\forall k < j : \pi[k] \models \phi_1)\}$.

Thus, the computation of

$$y_s^{min} = \inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} cost^{inf}(\pi, \phi_1 U \phi_2)$$

corresponds to solving the following equations:

$$y_s^{min} = \begin{cases} r(s) + \min(\{y_{s'}^{min} \mid s \models \phi_1 \wedge P(s,s') > 0\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad \text{if} \quad s \in S_P \vee s \in S_{\circledcirc} \\ r(s) + \min(\{y_{s'}^{min} \mid s \models \phi_1 \wedge T(s,l) = s'\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad \text{if} \quad s \in S_A \end{cases}$$

Note that the system is solved in the set $\mathbb{Q} \cup \{-\infty, +\infty\}$, where $\inf \emptyset = \min \emptyset = +\infty$. Thus, when there is no state satisfying the formula (the set of solutions is empty), the system returns $+\infty$.

The equation system above might give more than one solution. In this case we consider the maximum one.

The computation of the maximum cost is slightly different. We are interested in computing the supremum over the set of costs $cost(\pi, \phi_1 U \phi_2)$. Thus, we are not stopping the first time ϕ_2 is satisfied but we continue the computation as follows. We check if the state satisfying ϕ_2 is the last one. If yes, then the cost is equal to $r(s)$, otherwise we continue the computation of the cost based on the type of the state. Observe that the computation is performed in the set $\mathbb{Q} \cup \{-\infty, +\infty\}$ where $\sup \emptyset = \max \emptyset = -\infty$, and it returns $-\infty$ if the set of costs is empty.

For computing

$$y_s^{max} = \sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} cost^{sup}(\pi, \phi_1 U \phi_2)$$

we solve the following equation system:

$$y_s^{max} = \begin{cases} r(s) + \max(\{y_{s'}^{max} \mid s \models \phi_1 \wedge P(s,s') > 0\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad \text{if } \; s \in S_P \vee s \in S_{\circledcirc} \\ r(s) + \max(\{y_{s'}^{max} \mid s \models \phi_1 \wedge T(s,l) = s'\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad \text{if } \; s \in S_A \end{cases}$$

Here as well the equation system might give multiple solutions and we consider the minimum one.

8.4 Evaluation of Attack Trees with erPCTL

In the previous two sections we have proposed a translation from attack trees to MDPs and presented the new logic erPCTL, an extension of rPCTL with cost-related operators. In the following, we discuss how security properties of interest can be evaluated by means of model checking of erPCTL.

Security properties. Attributes to basic actions play an important role in the analysis of an attack scenario. They are used to express various properties of interest. In this part of the work we characterise the basic actions of an attack scenario with the success probability and the cost of performing the action. The properties we study range from quantitative to qualitative as well as from one-objective to multiple-objective properties. We formalise them in erPCTL.

We study probability-related properties such as "is the success probability of an attack greater than or equal to 0.2?" or "what is the maximum probability of success?". The first qualitative property is expressed in erPCTL as the formula $P_{\geq 0.2}(F\,success)$, while the second quantitative property is express as the formula $P_{max=?}(F\,success)$.

The characterisation of basic actions with cost allows to deduce the cheapest attack. We study questions such as "what is the minimum cost of an attack?". The property is expressed as the formula $C_{min=?}(F\,success)$. Moreover, having a cost budget c for the attacker, we can study more specific properties. For example, the attacker might want to know if whatever he/she does the cost of all successful attacks is in I, i.e., whether the attacker can always succeed by spending no more than the budget. We can express it with the question "is the cost of all successful attacks within the budget c?" and it is expressed in erPCTL as the formula $C_{[0,c]}(F\,success)$. On the other hand, a defender who is looking at the attack scenario might want to verify whether all successful attacks are outside the attacker's budget, i.e., "is the cost of all successful attacks greater than or equal to c?". The corresponding formula is $C_{[c,\infty)}(F\,success)$.

So far the cost-related properties we considered are evaluated over all attacks. However, the (clever) attacker might want to know if there exists at least one

successful attack within the budget c. We can express this property as the formula $\neg C_{[c,\infty)}(F\,success)$.

Our framework allows also to study multiple-objective properties such as "is there an attack with success probability at least 0.4 and cost at most 1500?" or "what is the maximum probability of an attack with cost at most 1500?". They expressed as the formulae $P_{\geq 0.4}(F\,success \mid [0, 1500])$ and $P_{max=?}(F\,success \mid [0, 1500])$, respectively.

Verification of security properties. We use model checking erPCTL, presented in Sect. 8.3, for verifying the security properties of MDPs constructed from attack trees. The model checking algorithm allows to verify both qualitative as well as quantitative properties. It first computes the value of interest for each state and then checks whether it satisfies the given property. For example, to verify that the success probability of an attack is at least 0.2, the model checking algorithm first computes for all states the probability of eventually reaching the success state, and than checks whether it is at least 0.2.

Observe that MDPs constructed from attack trees are finite and acyclic. Moreover, most properties of interest concern reachability of a success state, i.e., the path formula has the form $(F\,success)$.

Example. Consider the MDP given in Figure 8.2. We exploit the model checking algorithm of erPCTL to verify the security properties mentioned above. For example, the verification of the probabilistic query $P_{\geq 0.2}(F\,success)$ returns "false", meaning that there exists at least one attack with success probability less than 0.2. We compute the maximum success probability of an attack with the query $P_{max=?}(F\,success)$, which is 0.549.

Assume the attacker has a cost budget equal to 1500 and let us check whether all successful attacks are within the budget. The query $C_{\leq 1500}(F\,success)$ returns "false" meaning that there exists a successful attack with cost greater than the budget. Now, let us check whether there exists at least one successful attack within the budget. We verify the query $\neg C_{>1500}(F\,success)$, which returns "true" meaning that there is an attack within the budget. We can also compute the minimum cost of a successful attack with the query $C_{min=?}(F\,success)$, which is 900.

Finally, we verify multi-objective queries, such as $P_{\geq 0.4}(F\,success \mid [0, 1500])$ and $P_{max=?}(F\,success \mid [0, 1500])$. The first property evaluates to "false" meaning that there is no attack with probability at least 0.4 and cost at most 1500, while the second property computes the maximum probability of an attack with cost at most 1500, which is 0.18.

8.5 Concluding Remarks

Markov Decision Processes are powerful objects for modelling uncertain and randomised behaviour. They are commonly used formalism for studying systems that show a combined probabilistic and nondeterministic behaviour. Thus, we investigated the relation between attack trees and MDPs for modelling the probabilistic and nondeterministic behaviour of an attack scenario with probability and cost attributes. We proposed a translation of attack trees into MDPs, and used probabilistic model checking for verifying the security properties of an attack scenario expressed in rPCTL.

The logic rPCTL allows to reason about probability and expected reward properties of the system. Even though the evaluation of the expected reward is of interest in many domains, the analysis of exact cumulative cost of an attack is a typical property of interest in the study of attack scenarios. There are few extensions of rPCTL with cost-bounded until operator. Nevertheless, to the best of our knowledge none of the rPCTL operators evaluate the exact cost.

To overcome this limitation, we extended the logic with an exact cost operator. The new logic erPCTL allows to reason about the probability and exact cost measures and to evaluate both qualitative and quantitative queries. We presented the model checking algorithm for erPCTL. Finally, we showed how to verify security properties of an attack scenario with the presented model checking approach.

As future work, it would be interesting to investigate in details open probability intervals in the operator $P_J(\psi \mid I)$ discussed in Sect. 8.2. Moreover, it would be worth moving from attack trees and MDPs to attack-defence trees and games, and propose a logic for evaluating exact cost properties in an attack-defence scenario.

Chapter 9

Conclusion

> Still round the corner there may wait
> A new road or a secret gate.
>
> J.R.R.Tolkien

In this dissertation we have investigated the evaluation of qualitative and quantitative security properties of attack and defence scenarios expressed as graphical models for security, and the synthesis of optimal strategies for the attacker and the defender. Our initial thesis was that

graphical models for security of socio-technical organisations can be assigned formal syntax and semantics supporting the development of automated verification techniques for the analysis of security properties, both from a qualitative and a quantitative perspective. Such formal models, analyses, and properties encompass both attacker and defender behaviour, and can be relied upon to synthesise optimal strategies, thereby drawing attention to vulnerable fragments of the overall system.

In this dissertation we have studied attack trees and attack-defence trees, graphical security models for analysing attack and defence scenarios, and their connection to one- and two-player games. We have proposed techniques for the formal evaluation of security properties of complex attack and defence scenarios, improving on the literature of both tree and game verification.

In Sect. 9.1 we summarise the contribution of this dissertation by discussing how our developments support the claim above. At a high level, the first part of

this dissertation examined *graphical models for security of socio-technical organisations* and *assigned formal syntax and semantics supporting the development of automated verification techniques for the analysis of security properties.* The models and properties studied in the second part *encompassed both attacker and defender behaviour*, and proposed a game-theoretic approach *to synthesise optimal strategies.* Finally, the third and last part investigated in detail cost-related security properties. We conclude with final remarks followed by directions for future work in Sect. 9.2.

9.1 Contribution

Let us summarise briefly the main contributions of our work presented in each part of this dissertation.

Evaluation of attack-defence trees. Many organisations are complex socio-technical systems consisting of physical infrastructure, software components, human actors and the environment which all these are deployed. Securing such systems from potential attacks requires a thorough investigation of vulnerabilities in both the social and technical levels. Moreover, properties with multiple conflicting objective are of interest in security analysis of many complex socio-technical systems.

Graphical models such as attack trees and attack-defence trees prove useful techniques to visualise social and technical attacks. We have proposed a new formal syntax for attack tree and attack-defence trees. For making the models more expressive and for explicit representation of the defender actions, we have extended the model with new negation and changing player operators.

We have devised automated techniques for evaluating security properties with multiple parameters. For addressing multi-parameter optimisation of tree models and optimising all parameters at once, our have proposed techniques computed the set of optimal solutions, in term of Pareto efficiency.

From attack-defence trees to security games. We have then explored the connection between attack-defence trees and games from a verification-oriented prospective, as opposed to the theoretical studies found in the literature. Dealing with two players, the attacker and the defender, we have translated attack-defence trees into stochastic two-player games. The proposed game-based analysis of attack-defence scenarios allows to automatically verify security properties and to synthesise strategies for a player with respect to some goal.

Investigation of cost-related properties. Finally, we have investigated some limitations of existing game-based verification frameworks. For evaluating the exact cost of an attack though model checking techniques, we have extended rPCTL with the new operators. The proposed new logic erPCTL allows to

express probability and exact cost-related properties of attack scenarios. We have presented a translation from attack trees to MDPs and investigated security properties of an attack scenario with single and multiple parameters. We have developed model checking algorithm for the new operators.

9.2 Future Directions

During the work on this dissertation we have identified a number of possible extensions of the developed techniques and a number of research directions that can continue this line of research. We discussed some of them in the concluding remarks that follows each chapter, focusing on technical topics. In the following, we highlight some broader topics whose investigation in connection with what we have presented seems promising.

Our attack and attack-defence tree evaluation techniques assume that basic actions are independent from each other with respect to all parameters that characterise them, including probability and cost. In other words, the model assumes that the decision of a player to perform an action as well as its outcome do not influence the outcome of other actions. While this assumption is common to most of the work on attack and attack-defence trees, and may be valid for some specific domains, it sounds quite unrealistic in general. In the second part of this dissertation we partly tackled this issue by extending attack-defence trees with sequential operator that puts an order of execution among basic actions. Nonetheless, we have not considered the possibility that values describing the basic actions could be expressed as functions of other parameters or entire subtrees, rather than being static and fixed once and for all. On a similar note, dependency of basic actions might be studied for one player, or among players, where for instance a defender's action would have an impact on other defender's actions and another impact on the attacker's actions. Hence, removing the independence assumptions on basic actions and explore how our developments can cope with that is a first open line of research we should like to mention.

A second line of research that stems directly from our developments concerns moving from fully-observable to partially-observable games. In particular, this extension seems to indicate that games offer a natural framework to study an obvious feature of attack-defence scenario, namely the security-by-obscurity approach of complex organisations, which is instead not encompassed in tree models (and, to say the truth, out of the range of any explicit graphical representation).

Finally, real world-security is seldom a two-opponent game, being rather a story of multiple stakeholders with goals that are sometimes not fully conflicting and not fully aligned for the entire duration of a play [Umm11]. In the setting of multi-player games the quest for Pareto-efficient solutions seems even more central to unravel the intricacies of competition and cooperation of players.

Appendix A

Soundness of the Algorithmic Evaluation for Attack Trees

This appendix contains the proof of equivalence of the evaluation techniques for attack trees presented in Ch. 3.

A.1 Boolean Case

Here, we show the equivalence of the semantic evaluation given in Table 3.2 and the algorithmic evaluation presented in Table 3.3, Sect. 3.1.2. Before presenting the proof of Theorem 3.2, we formalise a lemma to be used later in the proof.

Lemma A.1 *Let $t \in Tree$ be a polarity-consistent tree. Then there exist Boolean assignments m^- and m^+ such that*

$$\min\{\mathcal{B}[\![t]\!]m \mid m \text{ Boolean assignment}\} = \mathcal{B}[\![t]\!]m^-$$

$$\max\{\mathcal{B}[\![t]\!]m \mid m \text{ Boolean assignment}\} = \mathcal{B}[\![t]\!]m^+$$

Proof. First let us show that there exists an assignment m^- that minimises $\mathcal{B}[\![t]\!]m$. We do it by constructing such an assignment, i.e., defining the value of m^- for each basic action.

Consider basic action $a \in Act$. The tree t is a polarity-consistent, hence a occurs in t either positively or negatively. Since we want to minimise the overall value and we have assumed that $ff < tt$, then

- if a occurs positively in t, we map it to the minimum value, $m^-(a) = ff$
- if a occurs negatively in t, we map it to the maximum value, $m^-(a) = tt$, so as to get the minimum value after applying negation.

Thus, we constructed an assignment m^- that minimised $\mathcal{B}[\![t]\!]m$.

In the same way we can construct an assignment m^+ that maximises $\mathcal{B}[\![t]\!]m$. In this case we map an action a to tt when it occurs positively in t, and we map it to ff when it occurs negatively in t. □

Theorem A.2 *Let $t \in Tree$ be a polarity-consistent attack tree. Then*

$$M(t) = INT(t)$$

Proof.[Theorem 3.2] The proof is organised by structural induction on the shape of the tree t.

Basis. The basis of the induction consists of the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Then, we have

$$\begin{aligned} M(t) &= (\min\{\mathcal{B}[\![\&_{\texttt{false}}]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{B}[\![\&_{\texttt{false}}]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (ff, ff) \end{aligned}$$

   and

$$INT(t) = (ff, ff)$$

   and we conclude $M(t) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the case $t = \&_{\texttt{false}}$.

3. Assume $t = a$. Then, we have

$$\begin{aligned} M(t) &= (\min\{\mathcal{B}[\![a]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{B}[\![a]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (\min\{ff, tt\}, \max\{ff, tt\}) \\ &= (ff, tt) \end{aligned}$$

   On the other hand, we have

$$INT(t) = (ff, tt)$$

   and the thesis follows immediately.

Step. The inductive step consists of the analysis of three cases, corresponding to the Boolean operators.

1. Assume $t = \&_\wedge(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{array}{rcl} M(t_i) & = & (\min\{\mathcal{B}[\![t_i]\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{B}[\![t_i]\!]m \mid m \text{ Boolean assignment}\}) = \\ INT(t_i) & = & (min_i, max_i) \end{array}$$

   for $i \in \{1, 2\}$, and we shall prove that $M(t) = INT(t)$.

   From Lemma A.1, the rules in Table 3.2 and the inductive hypothesis, we have

$$\begin{array}{rcl} M(t) & = & (\min\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!]m \mid m \text{ Boolean assignment}\}) \\ & = & (\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!]m^-, \mathcal{B}[\![\&_\wedge(t_1, t_2)]\!]m^+) \\ & = & (\mathcal{B}[\![t_1]\!]m^- \wedge \mathcal{B}[\![t_2]\!]m^-, \mathcal{B}[\![t_1]\!]m^+ \wedge \mathcal{B}[\![t_2]\!]m^+) \\ & = & (\min\{\mathcal{B}[\![t_1]\!]m \mid m \text{ Boolean assignment}\} \wedge \\ & & \min\{\mathcal{B}[\![t_2]\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{B}[\![t_1]\!]m \mid m \text{ Boolean assignment}\} \wedge \\ & & \max\{\mathcal{B}[\![t_2]\!]m \mid m \text{ Boolean assignment}\}) \\ & = & (min_1 \wedge min_2, max_1 \wedge max_2) \end{array}$$

   On the other hand, we have

$$\begin{array}{rcl} INT(t) & = & let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\} \\ & & in\ (min_1 \wedge min_2, max_1 \wedge max_2) \end{array}$$

   Hence, $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{array}{rcl} M(t') & = & (\min\{\mathcal{B}[\![t']\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{B}[\![t']\!]m \mid m \text{ Boolean assignment}\}) = \\ INT(t') & = & (min', max') \end{array}$$

   We shall show that $M(t) = INT(t)$.

   From Lemma A.1, the rules in Table 3.2, the inductive hypothesis, and

the fact that $\neg m^- = m^+, \neg m^+ = m^-$, we have

$$\begin{aligned} M(t) &= (\min\{\mathcal{B}[\![\&_\neg(t')]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{B}[\![\&_\neg(t')]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (\mathcal{B}[\![\&_\neg(t')]\!]m^-, \mathcal{B}[\![\&_\neg(t')]\!]m^+) \\ &= (\neg\mathcal{B}[\![t']\!]m^+, \neg\mathcal{B}[\![t']\!]m^-) \\ &= (\neg \max\{\mathcal{B}[\![t']\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \neg \min\{\mathcal{B}[\![t']\!]m \mid m \text{ Boolean assignment}\}) \\ &= (\neg max', \neg min') \end{aligned}$$

and

$$\begin{aligned} INT(t) &= let\ (min', max') = INT(t') \\ &\quad in\ (\neg max', \neg min') \end{aligned}$$

and the thesis follows.

□

A.2 Probabilistic Case

In the following we show Theorem 3.3, presented in Sect. 3.1.3.

Lemma A.3 *Let $t \in Tree$ be a polarity-consistent tree. Then there exist Boolean assignments m^- and m^+ such that*

$$\min\{\mathcal{P}[\![t]\!]m \mid m \text{ Boolean assignment}\} = \mathcal{P}[\![t]\!]m^-$$

$$\max\{\mathcal{P}[\![t]\!]m \mid m \text{ Boolean assignment}\} = \mathcal{P}[\![t]\!]m^+$$

Proof. The proof is done by constructing the Boolean assignments m^- and m^+. Let us first show the construction of m^-.

Consider basic action $a \in Act$. The tree t is a polarity-consistent, thus a occurs in t either positively or negatively. Moreover, we assumed that $p_1(a) \leq p_2(a)$, where $p_1(a)$ corresponds to $m(a) = ff$ and $p_2(a)$ corresponds to tt.

- If $p_1(a) = p_2(a)$, then we can map $m^-(a)$ to any Boolean value as it will give the same probabilistic value for a.
- If a occurs positively in t, we map it to the minimum value, $m^-(a) = ff$ in order to get the smallest probabilistic value.
- If a occurs negatively in t, we map it to the maximum values, $m^-(a) = tt$, so to get the minimum value after applying negation.

Thus, we constructed an assignment m^- that minimised $\mathcal{P}[\![t]\!]m$.

In the same way we can construct an assignment m^+ that maximises $\mathcal{P}[\![t]\!]m$. In this case we map an action a to tt, when it occurs positively in t, and we map it to ff, when it occurs negatively in t. □

Theorem A.4 *Let $t \in Tree$ be a polarity-consistent attack tree. Then*

$$M(t) = INT(t)$$

Proof.[Theorem 3.3] The proof is organised by structural induction on the shape of t.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Then, we have

$$\begin{array}{rcl} M(t) & = & (\min\{\mathcal{P}[\![\&_{\texttt{false}}]\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{P}[\![\&_{\texttt{false}}]\!]m \mid m \text{ Boolean assignment}\}) \\ & = & (0,0) \end{array}$$

   and

$$INT(t) = (0,0)$$

   and we conclude $M(t) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the case $t = \&_{\texttt{false}}$.

3. Assume $t = a$. Then, we have

$$\begin{array}{rcl} M(t) & = & (\min\{\mathcal{P}[\![a]\!]m \mid m \text{ Boolean assignment}\}, \\ & & \max\{\mathcal{P}[\![a]\!]m \mid m \text{ Boolean assignment}\}) \\ & = & (\min\{p_1(a), p_2(a)\}, \max\{p_1(a), p_2(a)\}) \\ & = & (p_1(a), p_2(a)) \end{array}$$

   for we have assumed $p_1(a) \leq p_2(a)$.

   On the other hand, we have

$$INT(t) = (p_1(a), p_2(a))$$

   and the thesis follows immediately.

Step. The inductive step consists of the analysis of three cases, corresponding to the probabilistic operators.

1. Assume $t = \&_\wedge(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t_i) &= (\min\{\mathcal{P}[\![t_i]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{P}[\![t_i]\!]m \mid m \text{ Boolean assignment}\}) = \\ INT(t_i) &= (min_i, max_i) \end{aligned}$$

for $i \in \{1, 2\}$, and we shall prove that $M(t) = INT(t)$.

From Lemma A.3, the rules in Table 3.4 and the inductive hypothesis, we have

$$\begin{aligned} M(t) &= (\min\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m^-, \mathcal{P}[\![\&_\wedge(t_1, t_2)]\!]m^+) \\ &= (\mathcal{P}[\![t_1]\!]m^- \cdot \mathcal{P}[\![t_2]\!]m^-, \mathcal{P}[\![t_1]\!]m^+ \cdot \mathcal{P}[\![t_2]\!]m^+) \\ &= (\min\{\mathcal{P}[\![t_1]\!]m \mid m \text{ Boolean assignment}\} \cdot \\ &\quad \min\{\mathcal{P}[\![t_2]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{P}[\![t_1]\!]m \mid m \text{ Boolean assignment}\} \cdot \\ &\quad \max\{\mathcal{P}[\![t_2]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (min_1 \cdot min_2, max_1 \cdot max_2) \end{aligned}$$

On the other hand, we have

$$\begin{aligned} INT(\&_\wedge(t_1, t_2)) &= \mathit{let}\ (min_i, max_i) = INT(t_i), i \in \{1, 2\} \\ &\quad \mathit{in}\ (min_1 \cdot min_2, max_1 \cdot max_2) \end{aligned}$$

Hence, we conclude that $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t') &= (\min\{\mathcal{P}[\![t']\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{P}[\![t']\!]m \mid m \text{ Boolean assignment}\}) = \\ INT(t') &= (min', max') \end{aligned}$$

and we shall prove $M(t) = INT(t)$.

From Lemma A.3, the rules in Table 3.4, the inductive hypothesis, and the fact that $\neg m^- = m^+, \neg m^+ = m^-$ we have

$$\begin{aligned} M(t) &= (\min\{\mathcal{P}[\![\&_\neg(t')]\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad \max\{\mathcal{P}[\![\&_\neg(t')]\!]m \mid m \text{ Boolean assignment}\}) \\ &= (\mathcal{P}[\![\&_\neg(t')]\!]m^-, \mathcal{P}[\![\&_\neg(t')]\!]m^+) \\ &= (1 - \mathcal{P}[\![t']\!]m^+, 1 - \mathcal{P}[\![t']\!]m^-) \\ &= (1 - \max\{\mathcal{P}[\![t']\!]m \mid m \text{ Boolean assignment}\}, \\ &\quad 1 - \min\{\mathcal{P}[\![t']\!]m \mid m \text{ Boolean assignment}\}) \\ &= (1 - max', 1 - min') \end{aligned}$$

On the other hand we have

$$\begin{aligned} INT(t) &= let\ (min', max') = INT(t') \\ &\quad in\ (1 - max', 1 - min') \end{aligned}$$

Thus, the thesis follows.

□

A.3 Boolean Case with Cost

We prove that the semantic evaluation given in Table 3.7 and the algorithmic evaluation presented in Table 3.8 are equivalent for linear attack trees.

First let us formalise a lemma to be used later in the proof of Theorem 3.5.

Lemma A.5

1. $MR^{--}(\{(b_1 \wedge b_2, c_1 + c_2) \mid (b_1, c_1) \in U_1, (b_2, c_2) \in U_2\}) =$
   $MR^{--}(\{(b_1 \wedge b_2, c_1 + c_2) \mid (b_1, c_1) \in MR^{--}(U_1), (b_2, c_2) \in MR^{--}(U_2)\})$
2. $MR^{+-}(\{(b_1 \wedge b_2, c_1 + c_2) \mid (b_1, c_1) \in U_1, (b_2, c_2) \in U_2\}) =$
   $MR^{+-}(\{(b_1 \wedge b_2, c_1 + c_2) \mid (b_1, c_1) \in MR^{+-}(U_1), (b_2, c_2) \in MR^{+-}(U_2)\})$
3. $MR^{--}(\{(b_1 \vee b_2, c_1 + c_2) \mid (b_1, c_1) \in U_1, (b_2, c_2) \in U_2\}) =$
   $MR^{--}(\{(b_1 \vee b_2, c_1 + c_2) \mid (b_1, c_1) \in MR^{--}(U_1), (b_2, c_2) \in MR^{--}(U_2)\})$
4. $MR^{+-}(\{(b_1 \vee b_2, c_1 + c_2) \mid (b_1, c_1) \in U_1, (b_2, c_2) \in U_2\}) =$
   $MR^{+-}(\{(b_1 \vee b_2, c_1 + c_2) \mid (b_1, c_1) \in MR^{+-}(U_1), (b_2, c_2) \in MR^{+-}(U_2)\})$
5. $MR^{--}(\{(\neg b, c) \mid (b, c) \in U\}) = MR^{--}(\{(\neg b, c) \mid (b, c) \in MR^{+-}(U)\})$
6. $MR^{+-}(\{(\neg b, c) \mid (b, c) \in U\}) = MR^{+-}(\{(\neg b, c) \mid (b, c) \in MR^{--}(U)\})$

Proof. The proof considers the six cases above distinctly.

1. Let us represent the sets U_1 and U_2 as the union of two partitions each:

$$U_1 = MR^{--}(U_1) \cup U_1 \backslash MR^{--}(U_1)$$

$$U_2 = MR^{--}(U_2) \cup U_2 \backslash MR^{--}(U_2)$$

Let us first recall the definition of the function MR^{--}. The function computes the set of all pairs that have lower value for both arguments with respect to the other pairs in the set:

$$MR^{--}(U_i) = \{(x, y) \in U_i \mid \forall (x', y') \in U_i : (x \sqsubseteq x' \vee y \sqsubset y') \wedge (x \sqsubset x' \vee y \sqsubseteq y')\}$$

Thus, the set $U_i \backslash MR^{--}(U_i)$ contains all pairs that have higher values for both arguments with respect to the pairs in the set $MR^{--}(U_i)$:

$$U_i \backslash MR^{--}(U_i) = \{(x,y) \in U_i \mid \exists (x',y') \in MR^{--}(U_i) : (x \sqsupset x' \wedge y \sqsupseteq y') \vee (x \sqsupseteq x' \wedge y \sqsupset y')\}$$

The proof is carried out by splitting the origin of the pairs (b_1, c_1) and (b_2, c_2) in the left-hand side with respect to the four possible partitions of U_1 and U_2.

(a) Assume $(b_1, c_1) \in MR^{--}(U_1)$ and $(b_2, c_2) \in MR^{--}(U_2)$. Then the identity of the two sides of the equation follows trivially.

(b) Assume $(b_1, c_1) \in U_1 \backslash MR^{--}(U_1)$ and $(b_2, c_2) \in U_2 \backslash MR^{--}(U_2)$. Applying the definition of $U_i \backslash MR^{--}(U_i)$ we can write
$\exists (b_1', c_1') \in MR^{--}(U_1)$ and $\exists (b_2', c_2') \in MR^{--}(U_2)$ such that
$((b_1 \sqsupset b_1' \wedge c_1 \sqsupseteq c_1') \vee (b_1 \sqsupseteq b_1' \wedge c_1 \sqsupset c_1')) \wedge$
$((b_2 \sqsupset b_2' \wedge c_2 \sqsupseteq c_2') \vee (b_2 \sqsupseteq b_2' \wedge c_2 \sqsupset c_2'))$

Based on distributivity of $\wedge$ over $\vee$, we can write
$(b_1 \sqsupset b_1' \wedge c_1 \sqsupseteq c_1') \wedge (b_2 \sqsupset b_2' \wedge c_2 \sqsupseteq c_2') \quad \vee$
$(b_1 \sqsupset b_1' \wedge c_1 \sqsupseteq c_1') \wedge (b_2 \sqsupseteq b_2' \wedge c_2 \sqsupset c_2') \quad \vee$
$(b_1 \sqsupseteq b_1' \wedge c_1 \sqsupset c_1') \wedge (b_2 \sqsupset b_2' \wedge c_2 \sqsupseteq c_2') \quad \vee$
$(b_1 \sqsupseteq b_1' \wedge c_1 \sqsupset c_1') \wedge (b_2 \sqsupseteq b_2' \wedge c_2 \sqsupset c_2')$

As conjunction over Booleans is monotonic and summation over non-negative rational numbers is monotonic, we have
$((b_1 \wedge b_2 \sqsupset b_1' \wedge b_2') \wedge (c_1 + c_2 \sqsupseteq c_1' + c_2')) \quad \vee$
$((b_1 \wedge b_2 \sqsupset b_1' \wedge b_2') \wedge (c_1 + c_2 \sqsupset c_1' + c_2')) \quad \vee$
$((b_1 \wedge b_2 \sqsupset b_1' \wedge b_2') \wedge (c_1 + c_2 \sqsupset c_1' + c_2')) \quad \vee$
$((b_1 \wedge b_2 \sqsupseteq b_1' \wedge b_2') \wedge (c_1 + c_2 \sqsupset c_1' + c_2'))$

From here we can see that the tuple $(b_1 \wedge b_2, c_1 + c_2)$ has higher values on both arguments with respect to the tuple $(b_1' \wedge b_2', c_1' + c_2')$. Thus, after applying the function MR^{--} the former will be discarded.

We proved that the point-wise combination of pairs from the sets $U_1 \backslash MR^{--}(U_1), U_2 \backslash MR^{--}(U_2)$ will be discarded in favour of the point-wise combination of pairs from the sets $MR^{--}(U_1), MR^{--}(U_2)$ when applying the outer MR^{--}.

(c) Assume $(b_1, c_1) \in U_1 \backslash MR^{--}(U_1)$ and $(b_2, c_2) \in MR^{--}(U_2)$. This case is analogous to the previous case hence omitted.

(d) Assume $(b_1, c_1) \in MR^{--}(U_1)$ and $(b_2, c_2) \in U_2 \backslash MR^{--}(U_2)$. This case is analogous to the previous case hence omitted.

Thus, we showed that when applying the outer MR^{--} only the combinations of pairs from the sets $MR^{--}(U_1)$ and $MR^{--}(U_2)$ are kept. Thus, we can consider these sets only instead of the sets U_1 and U_2 without loosing any pairs in the final result.

2. The proof is analogous to the proof of the first point of the lemma, where the sets U_1 and U_2 are represented as

$$U_1 = MR^{+-}(U_1) \cup U_1 \backslash MR^{+-}(U_1)$$

$$U_2 = MR^{+-}(U_2) \cup U_2 \backslash MR^{+-}(U_2)$$

and

$$\begin{aligned} U_i \backslash MR^{+-}(U_i) = \quad & \{(x,y) \in U_i \mid \exists (x',y') \in MR^{+-}(U_i) : \\ & (x \sqsubset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsubseteq x' \wedge y \sqsupset y')\} \end{aligned}$$

3. The proof is analogous to the proof of the first point of the lemma, using the fact that disjunction over Booleans is monotonic.

4. The proof is analogous to the previous cases, using the fact that disjunction over Booleans is monotonic.

5. The proof is analogous to the previous cases, where the set U is represented as

$$U = MR^{+-}(U) \cup U \backslash MR^{+-}(U)$$

and

$$\begin{aligned} U \backslash MR^{+-}(U) = \quad & \{(x,y) \in U \mid \exists (x',y') \in MR^{+-}(U) : \\ & (x \sqsubset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsubseteq x' \wedge y \sqsupset y')\} \end{aligned}$$

Moreover, we use the fact that negation over Booleans is anti-monotonic.

6. The proof is analogous to the previous cases, where the set U is represented as

$$U = MR^{--}(U) \cup U \backslash MR^{--}(U)$$

and

$$\begin{aligned} U \backslash MR^{--}(U) = \quad & \{(x,y) \in U \mid \exists (x',y') \in MR^{--}(U) : \\ & (x \sqsupset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsupseteq x' \wedge y \sqsupset y')\} \end{aligned}$$

and the fact that negation over Booleans is anti-monotonic.

□

Theorem A.6 *Let $t \in Tree$ be a linear attack tree. Then*

$$M(t, yield(t)) = INT(t)$$

Proof.[Theorem 3.5] The proof is organised by structural induction on the shape of the tree t.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Then we have

$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (\mathcal{B}[\![\&_{\texttt{false}}]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}), \\ &\quad MR^{+-}(\{\ (\mathcal{B}[\![\&_{\texttt{false}}]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\})) \\ &= (MR^{--}(\{(\mathit{ff}, 0)\}), MR^{+-}(\{(\mathit{ff}, 0)\})) \\ &= (\{(\mathit{ff}, 0)\}, \{(\mathit{ff}, 0)\}) \end{aligned}$$

   for $\mathcal{B}[\![\&_{\texttt{false}}]\!]m$ is independent from any assignment m.

   On the other hand, we have

$$INT(t) = (\{(\mathit{ff}, 0)\}, \{(\mathit{ff}, 0)\})$$

   and we conclude that $M(t, yield(t)) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the previous case.

3. Assume $t = a$. Then we have

$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (\mathcal{B}[\![a]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}), \\ &\quad MR^{+-}(\{\ (\mathcal{B}[\![a]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\})) \\ &= (MR^{--}(\{(\mathit{ff}, 0), (\mathit{tt}, c(a))\}), \\ &\quad MR^{+-}(\{(\mathit{ff}, 0), (\mathit{tt}, c(a))\})) \end{aligned}$$

   and

$$\begin{aligned} INT(t) &= (MR^{--}(\{(\mathit{ff}, 0), (\mathit{tt}, c(a))\}), \\ &\quad MR^{+-}(\{(\mathit{ff}, 0), (\mathit{tt}, c(a))\})) \end{aligned}$$

   and the thesis follows immediately.

Step. The inductive step consists of the case analysis of three cases, corresponding to the Boolean operators.

1. Assume $t = \&_{\wedge}(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t_i, yield(t_i)) &= (MR^{--}(\{\ (\mathcal{B}[\![t_i]\!]m, \mathfrak{b}) \mid cost(m, yield(t_i)) \leq \mathfrak{b}\}), \\ &\quad MR^{+-}(\{\ (\mathcal{B}[\![t_i]\!]m, \mathfrak{b}) \mid cost(m, yield(t_i)) \leq \mathfrak{b}\})) = \\ INT(t_i) &= (V_i, W_i) \end{aligned}$$

for $i \in \{1,2\}$, and we shall prove that $M(t) = INT(t)$.

Since t is a linear tree no action occurs both in t_1 and t_2. Thus, we can write the budget $\mathfrak{b}$ corresponding to $cost(m, yield(t)) \leq \mathfrak{b}$ as a summation of budgets $\mathfrak{b}_1$ and $\mathfrak{b}_2$ corresponding to $cost(m, yield(t_1)) \leq \mathfrak{b}_1$ and $cost(m, yield(t_2)) \leq \mathfrak{b}_2$, respectively.

From the mentioned fact, Lemma A.5, and the inductive hypothesis, we have

$$\begin{aligned}
M(t, yield(t)) &= (MR^{--}(\{\ (\mathcal{B}[\![\&_\wedge(t_1,t_2)]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}),\\
&\quad\ MR^{+-}(\{\ (\mathcal{B}[\![\&_\wedge(t_1,t_2)]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}))\\
&= (MR^{--}(\{(\mathcal{B}[\![t_1]\!]m \wedge \mathcal{B}[\![t_2]\!]m, \mathfrak{b}_1 + \mathfrak{b}_2) \mid\\
&\quad\ cost(m, yield(t_1)) \leq \mathfrak{b}_1, cost(m, yield(t_2)) \leq \mathfrak{b}_2\}),\\
&\quad\ MR^{+-}(\{(\mathcal{B}[\![t_1]\!]m \wedge \mathcal{B}[\![t_2]\!]m, \mathfrak{b}_1 + \mathfrak{b}_2) \mid\\
&\quad\ cost(m, yield(t_1)) \leq \mathfrak{b}_1, cost(m, yield(t_2)) \leq \mathfrak{b}_2\}))\\
&= let\ U_i = \{(\mathcal{B}[\![t_i]\!]m, \mathfrak{b}_i) \mid cost(m, yield(t_i)) \leq \mathfrak{b}_i\}\\
&\quad\ in\ (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in U_1, (b',c') \in U_2\}),\\
&\quad\ \ MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in U_1, (b',c') \in U_2\}))\\
&= (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in MR^{--}(U_1),\\
&\qquad\qquad (b',c') \in MR^{--}(U_2)\}),\\
&\quad\ MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in MR^{+-}(U_1),\\
&\qquad\qquad (b',c') \in MR^{+-}(U_2)\}))\\
&= (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in V_1, (b',c') \in V_2\}),\\
&\quad\ MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in W_1, (b',c') \in W_2\}))
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
INT(t) &= let\ (V_i, W_i) = INT(t_i), i \in \{1,2\}\\
&\quad\ in\ (MR^{--}(\{(b \wedge b', c + c') \mid (b,c) \in V_1, (b',c') \in V_2\}),\\
&\quad\ \ MR^{+-}(\{(b \wedge b', c + c') \mid (b,c) \in W_1, (b',c') \in W_2\}))
\end{aligned}$$

and we conclude that $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{aligned}
M(t', yield(t')) &= (MR^{--}(\{\ (\mathcal{B}[\![t']\!]m, \mathfrak{b}) \mid cost(m, yield(t')) \leq \mathfrak{b}\}),\\
&\qquad\ MR^{+-}(\{\ (\mathcal{B}[\![t']\!]m, \mathfrak{b}) \mid cost(m, yield(t')) \leq \mathfrak{b}\})) =\\
INT(t') &= (V', W')
\end{aligned}$$

From Lemma A.5 and the inductive hypothesis, we have

$$\begin{aligned}
M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (\mathcal{B}[\![\&_{\neg}(t')]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}),\\
&\quad\ MR^{+-}(\{\ (\mathcal{B}[\![\&_{\neg}(t')]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}))\\
&= (MR^{--}(\{\ (\neg\mathcal{B}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}),\\
&\quad\ MR^{+-}(\{\ (\neg\mathcal{B}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}))\\
&= \mathit{let}\ U = \{(\mathcal{B}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}\\
&\quad\ \mathit{in}\ (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in U\}),\\
&\qquad MR^{+-}(\{\ (\neg b, c) \mid (b, c) \in U\}))\\
&= (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in MR^{+-}(U)\}),\\
&\quad\ MR^{+-}(\{\ (\neg b, c) \mid (b, c) \in MR^{--}(U)\}))\\
&= (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in W\}),\\
&\quad\ MR^{+-}(\{\ (\neg b, c) \mid (b, c) \in V\}))
\end{aligned}$$

and

$$\begin{aligned}
\mathit{INT}(t) &= \mathit{let}\ (V, W) = \mathit{INT}(t)\\
&\quad\ \mathit{in}\ (MR^{--}(\{(\neg b, c) \mid (b, c) \in W\}),\\
&\qquad MR^{+-}(\{(\neg b, c) \mid (b, c) \in V\}))
\end{aligned}$$

and we conclude that $M(t) = \mathit{INT}(t)$.

□

A.4 Probabilistic Case with Cost

We prove that the semantic evaluation given in Table 3.9 and the algorithmic evaluation presented in Table 3.10 are equivalent for linear attack trees. First, let us formalise a lemma to be used later in the proof.

Lemma A.7

1. $MR^{--}(\{(p_1 \cdot p_2, c_1 + c_2) \mid (p_1, c_1) \in U_1, (p_2, c_2) \in U_2\}) =$
   $MR^{--}(\{(p_1 \cdot p_2, c_1 + c_2) \mid (p_1, c_1) \in MR^{--}(U_1), (p_2, c_2) \in MR^{--}(U_2)\})$
2. $MR^{+-}(\{(p_1 \cdot p_2, c_1 + c_2) \mid (p_1, c_1) \in U_1, (p_2, c_2) \in U_2\}) =$
   $MR^{+-}(\{(p_1 \cdot p_2, c_1 + c_2) \mid (p_1, c_1) \in MR^{+-}(U_1), (p_2, c_2) \in MR^{+-}(U_2)\})$
3. $MR^{--}(\{(1 - (1 - p_1) \cdot (1 - p_2), c_1 + c_2) \mid (p_1, c_1) \in U_1, (p_2, c_2) \in U_2\}) =$
   $MR^{--}(\{(1 - (1 - p_1) \cdot (1 - p_2), c_1 + c_2) \mid$
   $(p_1, c_1) \in MR^{--}(U_1), (p_2, c_2) \in MR^{--}(U_2)\})$
4. $MR^{+-}(\{(1 - (1 - p_1) \cdot (1 - p_2), c_1 + c_2) \mid (p_1, c_1) \in U_1, (p_2, c_2) \in U_2\}) =$
   $MR^{+-}(\{(1 - (1 - p_1) \cdot (1 - p_2), c_1 + c_2) \mid$
   $(p_1, c_1) \in MR^{+-}(U_1), (p_2, c_2) \in MR^{+-}(U_2)\})$
5. $MR^{--}(\{(1 - p, c) \mid (p, c) \in S\}) = MR^{--}(\{(1 - p, c) \mid (p, c) \in MR^{+-}(U)\})$
6. $MR^{+-}(\{(1 - p, c) \mid (p, c) \in S\}) = MR^{+-}(\{(1 - p, c) \mid (p, c) \in MR^{--}(U)\})$

Proof. The proof closely follows the proof of Lemma A.5. We give the detailed proof for the first equation only, while highlighting the considerations for the proof for the other equations.

1. Let us represent the sets U_1 and U_2 as the union of two partitions each:

$$U_1 = MR^{--}(U_1) \cup U_1 \backslash MR^{--}(U_1)$$

$$U_2 = MR^{--}(U_2) \cup U_2 \backslash MR^{--}(U_2)$$

Let us first recall the definition of the function MR^{--}. The function computes the set of all pairs that have lower value for both arguments with respect to the other pairs in the set.

$$MR^{--}(U_i) = \{(x, y) \in U_i \mid \forall (x', y') \in U_i : (x \sqsubseteq x' \vee y \sqsubset y') \wedge (x \sqsubset x' \vee y \sqsubseteq y')\}$$

Thus, the set $U_i \backslash MR^{--}(U_i)$ contains all pairs that have higher values for both arguments with respect to the pairs in the set $MR^{--}(U_i)$.

$$U_i \backslash MR^{--}(U_i) = \{(x, y) \in U_i \mid \exists (x', y') \in MR^{--}(U_i) : (x \sqsupset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsupseteq x' \wedge y \sqsupset y')\}$$

The proof is carried out by splitting the origin of the pairs (b_1, c_1) and (b_2, c_2) in the left-hand side with respect to the four possible partitions of U_1 and U_2.

   (a) Assume $(p_1, c_1) \in MR^{--}(U_1)$ and $(p_2, c_2) \in MR^{--}(U_2)$. Then the identity of the two sides of the equation follows trivially.

(b) Assume $(p_1, c_1) \in U_1 \backslash MR^{--}(U_1)$ and $(p_2, c_2) \in U_2 \backslash MR^{--}(U_2)$. Applying the definition of $U_i \backslash MR^{--}$ we can write
$\exists (p_1', c_1') \in MR^{--}(U_1)$ and $\exists (p_2', c_2') \in MR^{--}(U_2)$ such that
$((p_1 \sqsupset p_1' \wedge c_1 \sqsupseteq c_1') \ \vee \ (p_1 \sqsupseteq b_1' \wedge c_1 \sqsupset c_1')) \wedge$
$((p_2 \sqsupset p_2' \wedge c_2 \sqsupseteq c_2') \ \vee \ (p_2 \sqsupseteq b_2' \wedge c_2 \sqsupset c_2'))$
Based on distributivity of $\wedge$ over $\vee$, we can write
$(p_1 \sqsupset p_1' \wedge c_1 \sqsupseteq c_1') \wedge (p_2 \sqsupset p_2' \wedge c_2 \sqsupseteq c_2') \quad \vee$
$(p_1 \sqsupset p_1' \wedge c_1 \sqsupseteq c_1') \wedge (p_2 \sqsupseteq p_2' \wedge c_2 \sqsupset c_2') \quad \vee$
$(p_1 \sqsupseteq p_1' \wedge c_1 \sqsupset c_1') \wedge (p_2 \sqsupset p_2' \wedge c_2 \sqsupseteq c_2') \quad \vee$
$(p_1 \sqsupseteq p_1' \wedge c_1 \sqsupset c_1') \wedge (p_2 \sqsupseteq p_2' \wedge c_2 \sqsupset c_2')$
As multiplication over [0,1] is monotonic and summation over non-negative rational numbers is monotonic, we have
$((p_1 \cdot p_2 \sqsupset p_1' \cdot p_2') \ \wedge \ (c_1 + c_2 \sqsupseteq c_1' + c_2')) \quad \vee$
$((p_1 \cdot p_2 \sqsupset p_1' \cdot p_2') \ \wedge \ (c_1 + c_2 \sqsupset c_1' + c_2')) \quad \vee$
$((p_1 \cdot p_2 \sqsupset p_1' \cdot p_2') \ \wedge \ (c_1 + c_2 \sqsupset c_1' + c_2')) \quad \vee$
$((p_1 \cdot p_2 \sqsupseteq p_1' \cdot p_2') \ \wedge \ (c_1 + c_2 \sqsupset c_1' + c_2'))$
From here we can see that the tuple $(p_1 \cdot p_2, c_1 + c_2)$ has higher values on both arguments with respect to the tuple $(p_1' \cdot p_2', c_1' + c_2')$. Thus, after applying the function MR^{--} the former will be discarded.
We proved that the point-wise combination of pairs from the sets $U_1 \backslash MR^{--}(U_1), U_2 \backslash MR^{--}(U_2)$ will be discarded in favour of the point-wise combination of pairs from the sets $MR^{--}(U_1), MR^{--}(U_2)$ when applying the outer MR^{--}.

(c) Assume $(p_1, c_1) \in U_1 \backslash MR^{--}(U_1)$ and $(p_2, c_2) \in MR^{--}(U_2)$. This case is analogous to the previous case hence omitted.

(d) Assume $(p_1, c_1) \in MR^{--}(U_1)$ and $(p_2, c_2) \in U_2 \backslash MR^{--}(U_2)$. This case is analogous to the previous case hence omitted.

Thus, we showed that when applying the outer MR^{--} only the combinations of pairs from the sets $MR^{--}(U_1)$ and $MR^{--}(U_2)$ are kept. Thus, we can consider these sets only instead of the sets U_1 and U_2 without loosing any pairs in the final result.

2. The proof is analogous to the proof of the first point of the lemma, where the sets U_1 and U_2 are represented as

$$U_1 = MR^{+-}(U_1) \cup U_1 \backslash MR^{+-}(U_1)$$
$$U_2 = MR^{+-}(U_2) \cup U_2 \backslash MR^{+-}(U_2)$$

and

$$U_i \backslash MR^{+-}(U_i) = \{(x, y) \in U_i \mid \exists (x', y') \in MR^{+-}(U_i) : (x \sqsubset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsubseteq x' \wedge y \sqsupset y')\}$$

3. The proof is analogous to the proof of the first point of the lemma, using the fact that $1-(1-p)\cdot(1-p)$ over [0,1] is monotonic.

4. The proof is analogous to the previous cases, using the fact that $1-(1-p)\cdot(1-p)$ over [0,1] is monotonic.

5. The proof is analogous to the previous cases, where the sets U is represented as
$$U = MR^{+-}(U) \cup U_1 \backslash MR^{+-}(U)$$
and
$$\begin{aligned} U\backslash MR^{+-}(U) = \quad & \{(x,y) \in U \mid \exists (x',y') \in MR^{+-}(U) : \\ & (x \sqsubset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsubseteq x' \wedge y \sqsupset y')\} \end{aligned}$$
Moreover, we use the fact that minus over [0,1] is anti-monotonic.

6. The proof is analogous to the previous cases, where the sets U is represented as
$$U = MR^{--}(U) \cup U_1 \backslash MR^{--}(U)$$
and
$$\begin{aligned} U_i\backslash MR^{--}(U_i) = \quad & \{(x,y) \in U_i \mid \exists (x',y') \in MR^{--}(U_i) : \\ & (x \sqsupset x' \wedge y \sqsupseteq y') \ \vee \ (x \sqsupseteq x' \wedge y \sqsupset y')\} \end{aligned}$$
and the fact that minus over [0,1] is anti-monotonic.

□

Theorem A.8 *Let $t \in Tree$ be a linear attack tree. Then*

$$M(t, yield(t)) = INT(t)$$

Proof.[Theorem 3.6] The proof is organised by structural induction on the shape of the tree t.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\mathtt{false}}$. Then we have
$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (\mathcal{P}[\![\&_{\mathtt{false}}]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}), \\ &\quad\ MR^{+-}(\{\ (\mathcal{P}[\![\&_{\mathtt{false}}]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\})) \\ &= (MR^{--}(\{(0,0)\}), MR^{+-}(\{(0,0)\})) \\ &= (\{(0,0)\}, \{0,0)\}) \end{aligned}$$

for $\mathcal{P}[\![\&_{\texttt{false}}]\!]m$ is independent from any assignment m.

On the other hand, we have

$$INT(t) = (\{(0,0)\}, \{(0,0)\})$$

and we conclude that $M(t, yield(t)) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the previous case.

3. Assume $t = a$. Then we have

$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (\mathcal{P}[\![a]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\}), \\ &\quad\ MR^{+-}(\{\ (\mathcal{P}[\![a]\!]m, \mathfrak{b}) \mid cost(m, yield(t)) \leq \mathfrak{b}\})) \\ &= (MR^{--}(\{(p_1(a), 0), (p_2(a), c(a))\}), \\ &\quad\ MR^{+-}(\{(p_1(a), 0), (p_2(a), c(a))\})) \end{aligned}$$

and

$$\begin{aligned} INT(t) &= (MR^{--}(\{(p_1(a), 0), (p_2(a), c(a))\}), \\ &\quad\ MR^{+-}(\{(p_1(a), 0), (p_2(a), c(a))\})) \end{aligned}$$

and the thesis follows immediately.

Step. The inductive step consists of the case analysis of three cases, corresponding to the Boolean operators.

1. Assume $t = \&_{\wedge}(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t_i, yield(t_i)) &= (MR^{--}(\{\ (\mathcal{P}[\![t_i]\!]m, \mathfrak{b}) \mid cost(m, yield(t_i)) \leq \mathfrak{b}\}), \\ &\quad\ MR^{+-}(\{\ (\mathcal{P}[\![t_i]\!]m, \mathfrak{b}) \mid cost(m, yield(t_i)) \leq \mathfrak{b}\})) = \\ INT(t_i) &= (V_i, W_i) \end{aligned}$$

for $i \in \{1, 2\}$, and we shall prove that $M(t) = INT(t)$.

Since t is a linear tree no action occurs both in t_1 and t_2. Thus, we can write the budget $\mathfrak{b}$ corresponding to $cost(m, yield(t)) \leq \mathfrak{b}$ as a summation of budgets $\mathfrak{b}_1$ and $\mathfrak{b}_2$ corresponding to $cost(m, yield(t_1)) \leq \mathfrak{b}_1$ and $cost(m, yield(t_2)) \leq \mathfrak{b}_2$, respectively.

From the mentioned fact, Lemma A.7, and the inductive hypothesis, we

have

$$\begin{array}{rcl}
M(t, \mathit{yield}(t)) & = & (MR^{--}(\{\ (\mathcal{P}[\![\&_\wedge(t_1,t_2)]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t)) \leq \mathfrak{b}\}), \\
& & \ MR^{+-}(\{\ (\mathcal{P}[\![\&_\wedge(t_1,t_2)]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t)) \leq \mathfrak{b}\})) \\
& = & (MR^{--}(\{(\mathcal{P}[\![t_1]\!]m \cdot \mathcal{P}[\![t_2]\!]m, \mathfrak{b}_1 + \mathfrak{b}_2) \mid \\
& & \ \mathit{cost}(m, \mathit{yield}(t_1)) \leq \mathfrak{b}_1, \mathit{cost}(m, \mathit{yield}(t_2)) \leq \mathfrak{b}_2\}), \\
& & MR^{+-}(\{(\mathcal{P}[\![t_1]\!]m \cdot \mathcal{P}[\![t_2]\!]m, \mathfrak{b}_1 + \mathfrak{b}_2) \mid \\
& & \ \mathit{cost}(m, \mathit{yield}(t_1)) \leq \mathfrak{b}_1, \mathit{cost}(m, \mathit{yield}(t_2)) \leq \mathfrak{b}_2\})) \\
& = & \mathit{let}\ U_i = \{(\mathcal{P}[\![t_i]\!]m, \mathfrak{b}_i) \mid \mathit{cost}(m, \mathit{yield}(t_i)) \leq \mathfrak{b}_i\} \\
& & \mathit{in}\ (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in U_1, (p',c') \in U_2\}), \\
& & \quad MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in U_1, (p',c') \in U_2\})) \\
& = & (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in MR^{--}(U_1), \\
& & \qquad\qquad (p',c') \in MR^{--}(U_2)\}), \\
& & \ MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in MR^{+-}(U_1), \\
& & \qquad\qquad (p',c') \in MR^{+-}(U_2)\})) \\
& = & (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in V_1, (p',c') \in V_2\}), \\
& & \ MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in W_1, (p',c') \in W_2\}))
\end{array}$$

On the other hand, we have

$$\begin{array}{rcl}
\mathit{INT}(t) & = & \mathit{let}\ (V_i, W_i) = \mathit{INT}(t_i), i \in \{1,2\} \\
& & \mathit{in}\ (MR^{--}(\{(p \cdot p', c + c') \mid (p,c) \in V_1, (p',c') \in V_2\}), \\
& & \quad MR^{+-}(\{(p \cdot p', c + c') \mid (p,c) \in W_1, (p',c') \in W_2\}))
\end{array}$$

and we conclude that $M(t) = \mathit{INT}(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{array}{rcl}
M(t', \mathit{yield}(t')) & = & (MR^{--}(\{\ (\mathcal{P}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}), \\
& & \ MR^{+-}(\{\ (\mathcal{P}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\})) = \\
\mathit{INT}(t') & = & (V', W')
\end{array}$$

From Lemma A.7 and the inductive hypothesis, we have

$$\begin{aligned}
M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (\mathcal{P}[\![\&_\neg(t')]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}),\\
&\quad\ MR^{+-}(\{\ (\mathcal{P}[\![\&_\neg(t')]\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}))\\
&= (MR^{--}(\{\ (1 - \mathcal{P}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}),\\
&\quad\ MR^{+-}(\{\ (1 - \mathcal{P}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}))\\
&= \mathit{let}\ U = \{(\mathcal{P}[\![t']\!]m, \mathfrak{b}) \mid \mathit{cost}(m, \mathit{yield}(t')) \leq \mathfrak{b}\}\\
&\quad\ \mathit{in}\ (MR^{--}(\{\ (1 - p, c) \mid (p, c) \in U\}),\\
&\qquad MR^{+-}(\{\ (1 - p, c) \mid (p, c) \in U\}))\\
&= (MR^{--}(\{\ (1 - p, c) \mid (p, c) \in MR^{+-}(U)\}),\\
&\quad\ MR^{+-}(\{\ (1 - p, c) \mid (p, c) \in MR^{--}(U)\}))\\
&= (MR^{--}(\{\ (1 - p, c) \mid (p, c) \in W\}),\\
&\quad\ MR^{+-}(\{\ (1 - p, c) \mid (p, c) \in V\}))
\end{aligned}$$

and

$$\begin{aligned}
\mathit{INT}(t) &= \mathit{let}\ (V, W) = \mathit{INT}(t)\\
&\quad\ \mathit{in}\ (MR^{--}(\{(1 - p, c) \mid (p, c) \in W\}),\\
&\qquad MR^{+-}(\{(1 - p, c) \mid (p, c) \in V\}))
\end{aligned}$$

and we conclude that $M(t) = \mathit{INT}(t)$.

□

A.5 Probabilistic Case with Multiple Costs

In the following we prove Theorem 3.7 given in Sect 3.3. First let us formalise a lemma to be used later in the proof.

Lemma A.9

1. $MR^{--\cdots-}(\{(p \cdot p', c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in U_1, (p', c_1', \cdots, c_n') \in U_2\}) =$
   $MR^{--\cdots-}(\{(p \cdot p', c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in MR^{--\cdots-}(U_1), (p', c_1', \cdots, c_n') \in MR^{--\cdots-}(U_2)\})$
2. $MR^{+-\cdots-}(\{(p \cdot p', c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in U_1, (p', c_1', \cdots, c_n') \in U_2\}) =$
   $MR^{+-\cdots-}(\{(p \cdot p', c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in MR^{+-\cdots-}(U_1), (p', c_1', \cdots, c_n') \in MR^{+-\cdots-}(U_2)\})$
3. $MR^{--\cdots-}(\{(1 - (1-p)(1-p'), c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in U_1, (p', c_1', \cdots, c_n') \in U_2\}) =$
   $MR^{--\cdots-}(\{(1 - (1-p)(1-p'), c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in MR^{--\cdots-}(U_1), (p', c_1', \cdots, c_n') \in MR^{--\cdots-}(U_2)\})$
4. $MR^{+-\cdots-}(\{(1 - (1-p)(1-p'), c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in U_1, (p', c_1', \cdots, c_n') \in U_2\}) =$
   $MR^{+-\cdots-}(\{(1 - (1-p)(1-p'), c_1 + c_1', \cdots, c_n + c_n') \mid (p, c_1, \cdots, c_n) \in MR^{+-\cdots-}(U_1), (p', c_1', \cdots, c_n') \in MR^{+-\cdots-}(U_2)\})$
5. $MR^{--\cdots-}(\{(1-p, c_1, \cdots, c_n) \mid (p, c_1, \cdots, c_n) \in U\}) =$
   $MR^{--\cdots-}(\{(1-p, c_1, \cdots, c_n) \mid (p, c_1, \cdots, c_n) \in MR^{+-\cdots-}(U)\})$
6. $MR^{+-\cdots-}(\{(1-p, c_1, \cdots, c_n) \mid (p, c_1, \cdots, c_n) \in U\}) =$
   $MR^{+-\cdots-}(\{(1-p, c_1, \cdots, c_n) \mid (p, c_1, \cdots, c_n) \in MR^{--\cdots-}(U)\})$

Proof. The proof is analogous to the proof of Lemma A.7, hence omitted. □

Theorem A.10 *Let $t \in Tree$ be a linear attack tree. Then*

$$M^*(t, yield(t)) = INT^*(t)$$

Proof.[Theorem 3.7] The proof is organised by structural induction on the shape of the tree t using Lemma A.9. It is analogous to the proof of Theorem 3.6. □

Appendix B

Detailed Evaluation of Attack Trees

This appendix contains the detailed evaluation of the attack trees discussed in the examples of Ch. 3. Section B.1 shows the algorithmic evaluation of an attack tree with probabilities, displayed in Figure B.1, while Sect. B.2 gives the algorithmic evaluation of an attack tree with probability and a single-cost associated with basic actions, displayed in Figure B.2.

B.1 Probability Evaluation of Attack Trees

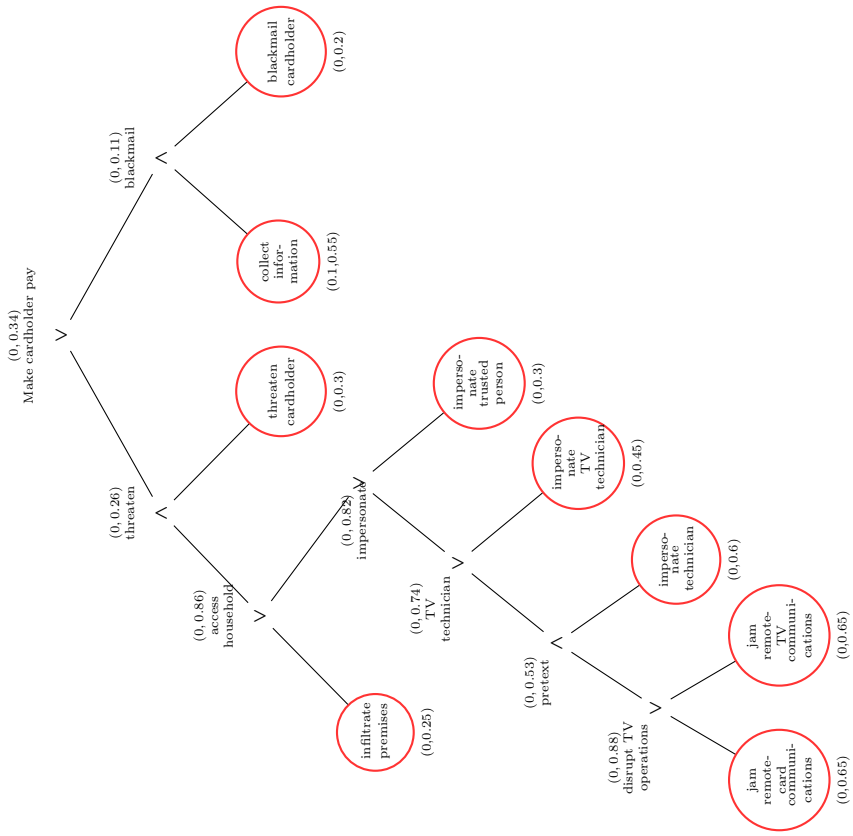


Figure B.1: The probabilistic algorithmic evaluation on the tree of Figure 3.1.

B.2 Cost Evaluation of Attack Trees

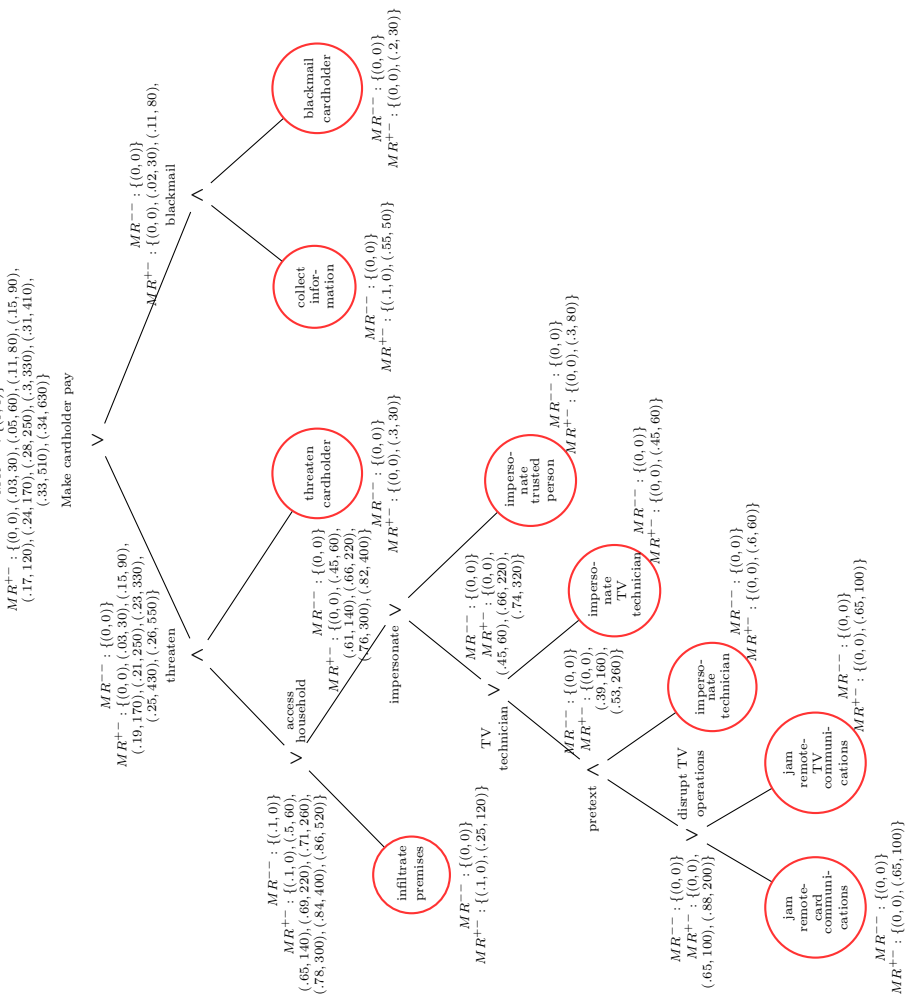


Figure B.2: The probabilistic algorithmic evaluation on the tree of Figure 3.1.

Appendix C

Soundness of the Algorithmic Evaluation for Attack-Defence Trees

This appendix contains the proof of equivalence of the evaluation techniques for attack-defence trees presented in Ch. 4.

C.1 Boolean Case

Here, we show the equivalence of the semantic evaluation given in Table 4.2 and the algorithmic evaluation presented in Table 4.3, for attack-defence trees.

Theorem C.1 *Let $t \in Tree$ be a polarity-consistent attack-defence stree. Then*

$$M(t) = INT(t)$$

Proof.[Theorem 4.2] The proof is organised by structural induction on the shape of the tree t. In order to simplify the presentation, we will write B.a. as a short-hand for Boolean assignment.

Basis. The basis of the induction consists of the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Then, we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{B}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{B}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\mathit{ff}, \mathit{ff}) \end{aligned}$$

   and

$$INT(t) = (\mathit{ff}, \mathit{ff})$$

   and we conclude $M(t) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the case $t = \&_{\texttt{false}}$.

3. Assume $t = a$. There are two possible cases.

   - If $a \in Act_p$, then we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\max\{m_p(a) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{m_p(a) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{m_p(a) \mid m_p \text{ B.a.}\}, \\ &\quad \max\{m_p(a) \mid m_p \text{ B.a.}\}) \\ &= (\mathit{ff}, \mathit{tt}) \end{aligned}$$

     and

$$INT(t) = (\mathit{ff}, \mathit{tt})$$

   - If $a \in Act_o$, then we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\max\{m_o(a) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{m_o(a) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\mathit{tt} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\mathit{ff} \mid m_p \text{ B.a.}\}) \\ &= (\mathit{tt}, \mathit{ff}) \end{aligned}$$

     and

$$INT(t) = (\mathit{tt}, \mathit{ff})$$

   Thus, we conclude that $M(t) = INT(t)$.

Step. The inductive step consists of the analysis of three cases, corresponding to the Boolean operators.

1. Assume $t = \&_\wedge(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{array}{rcl} M(t_i) & = & (\min\{\max\{\mathcal{B}[\![t_i]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{B}[\![t_i]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) = \\ INT(t_i) & = & (min_i, max_i) \end{array}$$

for $i \in \{1, 2\}$, and we shall prove that $M(t) = INT(t)$.

From Lemma A.1, the rules in Table 4.2, the inductive hypothesis and the fact that $Act_p \cap Act_o = \emptyset$, we have

$$\begin{array}{rcl} M(t) & = & (\min\{\max\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (\min\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}, \\ & & \max\{\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}) \\ & = & (\mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p^-, m_o^+), \\ & & \mathcal{B}[\![\&_\wedge(t_1, t_2)]\!](m_p^+, m_o^-)) \\ & = & (\mathcal{B}[\![t_1]\!](m_p^-, m_o^+) \wedge \mathcal{B}[\![t_2]\!](m_p^-, m_o^+), \\ & & \mathcal{B}[\![t_1]\!](m_p^+, m_o^-) \wedge \mathcal{B}[\![t_2]\!](m_p^+, m_o^-)) \\ & = & (\min\{\mathcal{B}[\![t_1]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\} \wedge \min\{\mathcal{B}[\![t_2]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}, \\ & & \max\{\mathcal{B}[\![t_1]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\} \wedge \max\{\mathcal{B}[\![t_2]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}) \\ & = & (\min\{\max\{\mathcal{B}[\![t_1]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\} \wedge \\ & & \min\{\max\{\mathcal{B}[\![t_2]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{B}[\![t_1]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\} \wedge \\ & & \max\{\min\{\mathcal{B}[\![t_2]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (min_1 \wedge min_2, max_1 \wedge max_2) \end{array}$$

On the other hand, we have

$$\begin{array}{rcl} INT(t) & = & let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\} \\ & & in\ (min_1 \wedge min_2, max_1 \wedge max_2) \end{array}$$

Hence, $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{array}{rcl} M(t') & = & (\min\{\max\{\mathcal{B}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{B}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) = \\ INT(t') & = & (min', max') \end{array}$$

We shall show that $M(t) = INT(t)$.

From Lemma A.1, the rules in Table 4.2, the inductive hypothesis, and the fact that $\neg m^- = m^+, \neg m^+ = m^-$, we have

$$\begin{aligned}
M(t) &= (\min\{\max\{\mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\},\\
&\quad \max\{\min\{\mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\})\\
&= (\min\{\mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\},\\
&\quad \max\{\mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\})\\
&= (\mathcal{B}[\![\&_\neg(t')]\!](m_p^-, m_o^+),\\
&\quad \mathcal{B}[\![\&_\neg(t')]\!](m_p^+, m_o^-))\\
&= (\neg\mathcal{B}[\![t']\!](m_p^+, m_o^-),\\
&\quad \neg\mathcal{B}[\![t']\!](m_p^-, m_o^+))\\
&= (\neg \max\{\mathcal{B}[\![t']\!](m_p, m_o^-) \mid m_p \text{ B.a.}\},\\
&\quad \neg \min\{\mathcal{B}[\![t']\!](m_p, m_o^+) \mid m_p \text{ B.a.}\})\\
&= (\neg \max\{\min\{\mathcal{B}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\},\\
&\quad \neg \min\{\max\{\mathcal{B}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\})\\
&= (\neg max', \neg min')
\end{aligned}$$

and

$$\begin{aligned}
INT(t) &= let\ (min', max') = INT(t')\\
&\quad in\ (\neg max', \neg min')
\end{aligned}$$

and the thesis follows.

4. Assume $t = \&_\sim(t')$. This case is analogous to the case $t = \&_\neg(t')$ hence omitted.

□

C.2 Probabilistic Case

In the following we show Theorem 4.3, presented in Sect. 4.1.3.

Theorem C.2 *Let $t \in Tree$ be a polarity-consistent attack-defence tree. Then*

$$M(t) = INT(t)$$

Proof.[Theorem 4.3] The proof is organised by structural induction on the shape of the tree t. For simplifying the presentation, we will use B.a. as shorthand for Boolean assignment.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Then, we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{P}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{P}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (0, 0) \end{aligned}$$

   and

$$INT(t) = (0, 0)$$

   and we conclude $M(t) = INT(t)$.

2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the case $t = \&_{\texttt{false}}$.

3. Assume $t = a$. There are two possible cases for a.

   - If $a \in Act_p$, then we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\max\{\mathcal{P}[\![a]\!]m_p \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{P}[\![a]\!]m_p \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\mathcal{P}[\![a]\!]m_p \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\mathcal{P}[\![a]\!]m_p \mid m_p \text{ B.a.}\}) \\ &= (p_1(a), p_2(a)) \end{aligned}$$

     for we have assumed $p_1(a) \leq p_2(a)$.
     On the other hand, we have

$$INT(t) = (p_1(a), p_2(a))$$

   - If $a \in Act_o$, then we have

$$\begin{aligned} M(t) &= (\min\{\max\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\max\{\mathcal{P}[\![a]\!]m_o \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{\mathcal{P}[\![a]\!]m_o \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{\max\{p_1(a), p_2(a)\} \mid m_p \text{ B.a.}\}, \\ &\quad \max\{\min\{p_1(a), p_2(a)\} \mid m_p \text{ B.a.}\}) \\ &= (\min\{p_2(a) \mid m_p \text{ B.a.}\}, \\ &\quad \max\{p_1(a) \mid m_p \text{ B.a.}\}) \\ &= (p_2(a), p_1(a)) \end{aligned}$$

     for we have assumed $p_1(a) \leq p_2(a)$.
     On the other hand, we have

$$INT(t) = (p_2(a), p_1(a))$$

Thus, the thesis follows immediately.

Step. The inductive step consists of the analysis of three cases, corresponding to the probabilistic operators.

1. Assume $t = \&_\wedge(t_1, t_2)$. By inductive hypothesis it holds that

$$\begin{array}{rcl} M(t_i) & = & (\min\{\max\{\mathcal{P}[\![t_i]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{P}[\![t_i]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) = \\ INT(t_i) & = & (min_i, max_i) \end{array}$$

for $i \in \{1, 2\}$, and we shall prove that $M(t) = INT(t)$.

From Lemma A.3, the rules in Table 4.4 and the inductive hypothesis, we have

$$\begin{array}{rcl} M(t) & = & (\min\{\max\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (\min\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}, \\ & & \max\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}) \\ & = & (\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p^-, m_o^+), \\ & & \mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p^+, m_o^-)) \\ & = & (\mathcal{P}[\![t_1]\!](m_p^-, m_o^+) \cdot \mathcal{P}[\![t_2]\!](m_p^-, m_o^+), \\ & & \mathcal{P}[\![t_1]\!](m_p^+, m_o^-) \cdot \mathcal{P}[\![t_2]\!](m_p^+, m_o^-)) \\ & = & (\min\{\mathcal{P}[\![t_1]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\} \cdot \min\{\mathcal{P}[\![t_2]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}, \\ & & \max\{\mathcal{P}[\![t_1]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\} \cdot \max\{\mathcal{P}[\![t_2]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}) \\ & = & (\min\{\max\{\mathcal{P}[\![t_1]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\} \cdot \\ & & \min\{\max\{\mathcal{P}[\![t_2]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{P}[\![t_1]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\} \cdot \\ & & \max\{\min\{\mathcal{P}[\![t_2]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (min_1 \cdot min_2, max_1 \cdot max_2) \end{array}$$

On the other hand, we have

$$\begin{array}{rcl} INT(t) & = & let\ (min_i, max_i) = INT(t_i), i \in \{1, 2\} \\ & & in\ (min_1 \cdot min_2, max_1 \cdot max_2) \end{array}$$

Hence, $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. By inductive hypothesis it holds that

$$\begin{array}{rcl} M(t') & = & (\min\{\max\{\mathcal{P}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{P}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) = \\ INT(t') & = & (min', max') \end{array}$$

and we shall prove $M(t) = INT(t)$.

From Lemma A.3, the rules in Table 4.4, the inductive hypothesis, and the fact that $\neg m^- = m^+, \neg m^+ = m^-$ we have

$$\begin{array}{rcl} M(t) & = & (\min\{\max\{\mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & \max\{\min\{\mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (\min\{\mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}, \\ & & \max\{\mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}) \\ & = & (\mathcal{P}[\![\&_\neg(t')]\!](m_p^-, m_o^+), \\ & & \mathcal{P}[\![\&_\neg(t')]\!](m_p^+, m_o^-)) \\ & = & (1 - \mathcal{P}[\![t']\!](m_p^+, m_o^-), \\ & & 1 - \mathcal{P}[\![t']\!](m_p^-, m_o^+)) \\ & = & (1 - \max\{\mathcal{P}[\![t']\!](m_p, m_o^-) \mid m_p \text{ B.a.}\}, \\ & & 1 - \min\{\mathcal{P}[\![t']\!](m_p, m_o^+) \mid m_p \text{ B.a.}\}) \\ & = & (1 - \max\{\min\{\mathcal{P}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}, \\ & & 1 - \min\{\max\{\mathcal{P}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \mid m_p \text{ B.a.}\}) \\ & = & (1 - max', 1 - min') \end{array}$$

On the other handle, we have

$$\begin{array}{rcl} INT(t) & = & let\ (min', max') = INT(t') \\ & & in\ (1 - max', 1 - min') \end{array}$$

and the thesis follows.

4. Assume $t = \&_\sim(t')$. This case is analogous to the case $t = \&_\neg(t')$ hence omitted. As we have discussed in Sect. 4.1.2 the operators $\&_\neg$ and $\&_\sim$ unfold identically, even if the reasoning supporting them is different.

□

C.3 Boolean Case with Cost

We prove that the semantic evaluation given in Table 4.7 and the algorithmic evaluation presented in Table 4.8 are equivalent for linear attack-defence trees.

Theorem C.3 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M(t, yield(t)) = INT(t)$$

Proof.[Theorem 3.5] The proof is organised by structural induction on the shape of the tree t. In order to simplify the presentation, we will write B.a. as short-hand for Boolean assignment.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\texttt{false}}$. Let us first evaluate the functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$.
$$f_1^{m_p}(t) = \max\{\mathcal{B}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = \mathit{ff}$$
$$f_2^{m_p}(t) = \min\{\mathcal{B}[\![\&_{\texttt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = \mathit{ff}$$
We have
$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_{\texttt{false}}), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (f_2^{m_p}(\&_{\texttt{false}}), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}))\\ &= (MR^{--}(\{\ (\mathit{ff}, \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (\mathit{ff}, \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}))\\ &= (\{(\mathit{ff}, 0)\}, \{(\mathit{ff}, 0)\}) \end{aligned}$$
On the other hand, we have
$$INT(\&_{\text{false}}) = (\{(\mathit{ff}, 0)\}, \{(\mathit{ff}, 0)\})$$
and we conclude that $M(t, yield(t)) = INT(t)$
2. Assume $t = \&_{\texttt{true}}$. This case is analogous to the previous case.
3. Assume $t = a$. There are two possible cases.
   - If $a \in Act_p$, then the evaluation of functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$ is
   $$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = \max\{m_p(a) \mid m_o \text{ B.a.}\}\\ &= m_p(a)\\ f_2^{m_p}(t) &= \min\{\mathcal{B}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = \min\{m_p(a) \mid m_o \text{ B.a.}\}\\ &= m_p(a) \end{aligned}$$
   and we have
   $$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (f_1^{m_p}(a), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (f_2^{m_p}(a), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}))\\ &= (MR^{--}(\{\ (m_p(a), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (m_p(a), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}))\\ &= (MR^{--}(\{(\mathit{ff}, 0), (tt, c(a))\}),\\ &\quad\ MR^{+-}(\{(\mathit{ff}, 0), (tt, c(a))\})) \end{aligned}$$

and

$$\begin{aligned} INT(t) &= (MR^{--}(\{(\mathit{ff},0),(tt,c(a))\}),\\ &\quad\ MR^{+-}(\{(\mathit{ff},0),(tt,c(a))\})) \end{aligned}$$

- If $a \in Act_o$, then the evaluation of functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$ is

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{B}[\![a]\!](m_p,m_o) \mid m_o \text{ B.a.}\} = \max\{m_o(a) \mid m_o \text{ B.a.}\}\\ &= \max\{\mathit{ff},tt\} = tt\\ f_2^{m_p}(t) &= \min\{\mathcal{B}[\![a]\!](m_p,m_o) \mid m_o \text{ B.a.}\} = \min\{m_o(a) \mid m_o \text{ B.a.}\}\\ &= \min\{\mathit{ff},tt\} = \mathit{ff} \end{aligned}$$

and we have

$$\begin{aligned} M(t,\mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(a),\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t)) \le \mathfrak{b}_p\}),\\ &\qquad MR^{+-}(\{\ (f_2^{m_p}(a),\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t)) \le \mathfrak{b}_p\}))\\ &= (MR^{--}(\{\ (tt,\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t)) \le \mathfrak{b}_p\}),\\ &\qquad MR^{+-}(\{\ (\mathit{ff},\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t)) \le \mathfrak{b}_p\}))\\ &= (MR^{--}(\{(tt,0)\}),\\ &\qquad MR^{+-}(\{(\mathit{ff},0)\})) \end{aligned}$$

On the other hand, we have

$$INT(t) = (MR^{--}(\{(tt,0)\}), MR^{+-}(\{(\mathit{ff},0)\}))$$

Thus, the thesis follows immediately.

Step. The inductive step consists of the analysis of three cases, corresponding to the Boolean operators.

1. Assume $t = \&_\wedge(t_1,t_2)$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t_i,\mathit{yield}(t_i)) &= (MR^{--}(\{\ (f_1^{m_p}(t_i),\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t_i)) \le \mathfrak{b}_p\}),\\ &\qquad MR^{+-}(\{\ (f_2^{m_p}(t_i),\mathfrak{b}_p) \mid \mathit{cost}(m_p,\mathit{yield}(t_i)) \le \mathfrak{b}_p\})) =\\ INT(t_i) &= (V_i,W_i) \end{aligned}$$

for $i \in \{1,2\}$, and we shall prove that $M(t) = INT(t)$.

Let us first evaluate functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$. From Lemma A.1, we have

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{B}[\![\&_\wedge(t_1,t_2)]\!](m_p,m_o) \mid m_o \text{ B.a.}\}\\ &= \mathcal{B}[\![\&_\wedge(t_1,t_2)]\!](m_p,m_o^+)\\ &= \mathcal{B}[\![t_1]\!](m_p,m_o^+) \wedge \mathcal{B}[\![t_2]\!](m_p,m_o^+)\\ &= \max\{\mathcal{B}[\![t_1]\!](m_p,m_o) \mid m_o \text{ B.a.}\} \wedge \max\{\mathcal{B}[\![t_2]\!](m_p,m_o) \mid m_o \text{ B.a.}\}\\ &= f_1^{m_p}(t_1) \wedge f_1^{m_p}(t_2) \end{aligned}$$

Analogously, we have

$$f_2^{m_p}(t) = f_2^{m_p}(t_1) \wedge f_2^{m_p}(t_2)$$

Let us now evaluate $M(t)$ and $INT(t)$.

Since t is a linear tree no action occurs both in t_1 and t_2. Thus, we can write the budget $\mathfrak{b}_p$ corresponding to $cost(m_{,p}\, yield(t)) \leq \mathfrak{b}_p$ as a summation of budgets $\mathfrak{b}_p^1$ and $\mathfrak{b}_p^2$ corresponding to $cost(m_p, yield(t_1)) \leq \mathfrak{b}_p^1$ and $cost(m_p, yield(t_2)) \leq \mathfrak{b}_p^2$, respectively.

From the mentioned fact, Lemma A.5 and the inductive hypothesis, we have

$$\begin{aligned}
M(t, yield(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_\wedge(t_1,t_2)), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}),\\
&\quad\ MR^{+-}(\{\ (f_2^{m_p}(\&_\wedge(t_1,t_2)), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}))\\
&= (MR^{--}(\{\ (f_1^{m_p}(t_1) \wedge f_1^{m_p}(t_2), \mathfrak{b}_p^1 + \mathfrak{b}_p^2) \mid\\
&\qquad cost(m_p, yield(t_1)) \leq \mathfrak{b}_p^1, cost(m_p, yield(t_2)) \leq \mathfrak{b}_p^2\}),\\
&\quad\ MR^{+-}(\{\ (f_2^{m_p}(t_1) \wedge f_2^{m_p}(t_2), \mathfrak{b}_p^1 + \mathfrak{b}_p^2) \mid\\
&\qquad cost(m_p, yield(t_1)) \leq \mathfrak{b}_p^1, cost(m_p, yield(t_2)) \leq \mathfrak{b}_p^2\}),\\
&= let\ (X_i, Y_i) = (\{\ (f_1^{m_p}(t_i), \mathfrak{b}_p^i) \mid cost(m_p, yield(t_i)) \leq \mathfrak{b}_p^i\}),\\
&\qquad\qquad\qquad\qquad \{\ (f_2^{m_p}(t_i), \mathfrak{b}_p^i) \mid cost(m_p, yield(t_i)) \leq \mathfrak{b}_p^i\})\\
&\quad\ in\ (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in X_1, (b',c') \in X_2\}),\\
&\qquad MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in Y_1, (b',c') \in Y_2\}))\\
&= (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in MR^{--}(X_1),\\
&\qquad\qquad\qquad\qquad (b',c') \in MR^{--}(X_2)\}),\\
&\quad\ MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in MR^{+-}(Y_1),\\
&\qquad\qquad\qquad\qquad (b',c') \in MR^{+-}(Y_2)\}))\\
&= (MR^{--}(\{\ (b \wedge b', c + c') \mid (b,c) \in V_1, (b',c') \in V_2\}),\\
&\quad\ MR^{+-}(\{\ (b \wedge b', c + c') \mid (b,c) \in W_1, (b',c') \in W_2\}))
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
INT(t) &= let\ (V_i, W_i) = INT(t_i), i \in \{1,2\}\\
&\quad\ in\ (MR^{--}(\{(b \wedge b', c + c') \mid (b,c) \in V_1, (b',c') \in V_2\}),\\
&\qquad MR^{+-}(\{(b \wedge b', c + c') \mid (b,c) \in W_1, (b',c') \in W_2\}))
\end{aligned}$$

and we conclude that $M(t) = INT(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t', \mathit{yield}(t')) &= (MR^{--}(\{\ (f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\})) = \\ \mathit{INT}(t') &= (V, W) \end{aligned}$$

Let us first evaluate functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$. From Lemma A.1 and the fact that $\neg m^+ = m^-$, we have

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\}\\ &= \mathcal{B}[\![\&_\neg(t')]\!](m_p, m_o^+)\\ &= \neg\mathcal{B}[\![t']\!](m_p, m_o^-)\\ &= \neg \min\{\mathcal{B}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\}\\ &= \neg f_2^{m_p}(t') \end{aligned}$$

Analogously, we have

$$f_2^{m_p}(t) = \neg f_1^{m_p}(t')$$

Let us now evaluate $M(t)$ and $\mathit{INT}(t)$.

From Lemma A.5 and the inductive hypothesis, we have

$$\begin{aligned} M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_\neg(t')), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (f_2^{m_p}(\&_\neg(t')), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}))\\ &= (MR^{--}(\{\ (\neg f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\ &\quad\ MR^{+-}(\{\ (\neg f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\ &= \mathit{let}\ (X, Y) = (\{\ (f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\ &\qquad\qquad\qquad\qquad \{\ (f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\})\\ &\quad\ \mathit{in}\ (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in Y\}),\\ &\qquad MR^{+-}(\{\ (\neg b, c) \mid (b, c) \in X\}))\\ &= (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in MR^{+-}(Y)\}),\\ &\quad\ MR^{+-}(\{\ (\neg b, c + c') \mid (b, c) \in MR^{--}(X)\}))\\ &= (MR^{--}(\{\ (\neg b, c) \mid (b, c) \in W\}),\\ &\quad\ MR^{+-}(\{\ (\neg b, c) \mid (b, c) \in V\})) \end{aligned}$$

On the other hand, we have

$$\begin{aligned} \mathit{INT}(\&_\neg(t)) &= \mathit{let}\ (V, W) = \mathit{INT}(t)\\ &\quad\ \mathit{in}\ (MR^{--}(\{(\neg b, c) \mid (b, c) \in W\}),\\ &\qquad MR^{+-}(\{(\neg b, c) \mid (b, c) \in V\})) \end{aligned}$$

and we conclude that $M(t) = \mathit{INT}(t)$.

4. Assume $t = \&_{\sim}(t')$. This case is analogous to the case $t = \&_{\neg}(t')$ hence omitted.

□

C.4 Probabilistic Case with Cost

We prove that the semantic evaluation given in Table 4.9 and the algorithmic evaluation presented in Table 4.10 are equivalent for linear attack-defence trees.

Theorem C.4 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M(t, yield(t)) = INT(t)$$

Proof.[Theorem 4.5] The proof is organised by structural induction on the shape of the tree t. In order to simplify the presentation, we will write B.a. as short-hand for Boolean assignment.

Basis. The basis of the induction consists in the analysis of three cases.

1. Assume $t = \&_{\mathtt{false}}$. Let us first evaluate the functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$.

$$f_1^{m_p}(t) = \max\{\mathcal{P}[\![\&_{\mathtt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = 0$$

$$f_2^{m_p}(t) = \min\{\mathcal{P}[\![\&_{\mathtt{false}}]\!](m_p, m_o) \mid m_o \text{ B.a.}\} = 0$$

We have

$$\begin{aligned} M(t, yield(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_{\mathtt{false}}), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (f_2^{m_p}(\&_{\mathtt{false}}), \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{\ (0, \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (0, \mathfrak{b}_p) \mid cost(m_p, yield(t)) \leq \mathfrak{b}_p\})) \\ &= (\{(0,0)\}, \{(0,0)\}) \end{aligned}$$

On the other hand, we have

$$INT(\&_{\mathtt{false}}) = (\{(0,0)\}, \{(0,0)\})$$

and we conclude that $M(t, yield(t)) = INT(t)$.

2. Assume $t = \&_{\mathtt{true}}$. This case is analogous to the previous case.

3. Assume $t = a$. There are two possible cases.

- If $a \in Act_p$, then the evaluation of functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$ is

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \max\{\mathcal{P}[\![a]\!]m_p \mid m_o \text{ B.a.}\} = \mathcal{P}[\![a]\!]m_p \\ f_2^{m_p}(t) &= \min\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \min\{\mathcal{P}[\![a]\!]m_p \mid m_o \text{ B.a.}\} = \mathcal{P}[\![a]\!]m_p \end{aligned}$$

  and we have

$$\begin{aligned} M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (f_2^{m_p}(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{\ (\mathcal{P}[\![a]\!]m_p, \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (\mathcal{P}[\![a]\!]m_p, \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{(p_1(a), 0), (p_2(a), c(a))\}), \\ &\quad MR^{+-}(\{(p_1(a), 0), (p_2(a), c(a))\})) \end{aligned}$$

  and

$$\begin{aligned} \mathit{INT}(t) &= (MR^{--}(\{(p_1(a), 0), (p_2(a), c(a))\}), \\ &\quad MR^{+-}(\{(p_1(a), 0), (p_2(a), c(a))\})) \end{aligned}$$

- If $a \in Act_o$, then the evaluation of functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$ is

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \max\{\mathcal{P}[\![a]\!]m_o \mid m_o \text{ B.a.}\} = \max\{p_1(a), p_2(a)\} = p_2(a) \\ f_2^{m_p}(t) &= \min\{\mathcal{P}[\![a]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \min\{\mathcal{P}[\![a]\!]m_o \mid m_o \text{ B.a.}\} = \min\{p_1(a), p_2(a)\} = p_1(a) \end{aligned}$$

  and we have

$$\begin{aligned} M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (f_2^{m_p}(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{\ (p_2(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (p_1(a), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{(p_2(a), 0)\}), \\ &\quad MR^{+-}(\{(p_1(a), 0)\})) \end{aligned}$$

  On the other hand, we have

$$\mathit{INT}(t) = (MR^{--}(\{(p_2(a), 0)\}), MR^{+-}(\{(p_1(a), 0)\}))$$

Thus, the thesis follows immediately.

Step. The inductive step consists of the analysis of three cases, corresponding to the probabilistic operators.

1. Assume $t = \&_\wedge(t_1, t_2)$. Then, by inductive hypothesis it holds that

$$\begin{aligned} M(t_i, \mathit{yield}(t_i)) &= (MR^{--}(\{\ (f_1^{m_p}(t_i), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t_i)) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (f_2^{m_p}(t_i), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t_i)) \leq \mathfrak{b}_p\})) = \\ \mathit{INT}(t_i) &= (V_i, W_i) \end{aligned}$$

for $i \in \{1, 2\}$, and we shall prove that $M(t) = \mathit{INT}(t)$.

Let us first evaluate functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$. From Lemma A.3, we have

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \mathcal{P}[\![\&_\wedge(t_1, t_2)]\!](m_p, m_o^+) \\ &= \mathcal{P}[\![t_1]\!](m_p, m_o^+) \cdot \mathcal{P}[\![t_2]\!](m_p, m_o^+) \\ &= \max\{\mathcal{P}[\![t_1]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \cdot \max\{\mathcal{P}[\![t_2]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= f_1^{m_p}(t_1) \cdot f_1^{m_p}(t_2) \end{aligned}$$

Analogously, we have

$$f_2^{m_p}(t) = f_2^{m_p}(t_1) \cdot f_2^{m_p}(t_2)$$

Let us now evaluate $M(t)$ and $\mathit{INT}(t)$.

Since t is a linear tree no action occurs both in t_1 and t_2. Thus, we can write the budget $\mathfrak{b}_p$ corresponding to $\mathit{cost}(m_{,p}\, \mathit{yield}(t)) \leq \mathfrak{b}_p$ as a summation of budgets $\mathfrak{b}_p^1$ and $\mathfrak{b}_p^2$ corresponding to $\mathit{cost}(m_p, \mathit{yield}(t_1)) \leq \mathfrak{b}_p^1$ and $\mathit{cost}(m_p, \mathit{yield}(t_2)) \leq \mathfrak{b}_p^2$, respectively.

From the mentioned fact, Lemma A.7 and the inductive hypothesis, we

have

$$
\begin{aligned}
M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_\wedge(t_1,t_2)), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}),\\
&\quad MR^{+-}(\{\ (f_2^{m_p}(\&_\wedge(t_1,t_2)), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t)) \leq \mathfrak{b}_p\}))\\
&= (MR^{--}(\{\ (f_1^{m_p}(t_1) \cdot f_1^{m_p}(t_2), \mathfrak{b}_p^1 + \mathfrak{b}_p^2) \mid\\
&\qquad \mathit{cost}(m_p, \mathit{yield}(t_1)) \leq \mathfrak{b}_p^1, \mathit{cost}(m_p, \mathit{yield}(t_2)) \leq \mathfrak{b}_p^2\}),\\
&\quad MR^{+-}(\{\ (f_2^{m_p}(t_1) \cdot f_2^{m_p}(t_2), \mathfrak{b}_p^1 + \mathfrak{b}_p^2) \mid\\
&\qquad \mathit{cost}(m_p, \mathit{yield}(t_1)) \leq \mathfrak{b}_p^1, \mathit{cost}(m_p, \mathit{yield}(t_2)) \leq \mathfrak{b}_p^2\}),\\
&= \mathit{let}\ (X_i, Y_i) = (\{\ (f_1^{m_p}(t_i), \mathfrak{b}_p^i) \mid \mathit{cost}(m_p, \mathit{yield}(t_i)) \leq \mathfrak{b}_p^i\}),\\
&\qquad\qquad \{\ (f_2^{m_p}(t_i), \mathfrak{b}_p^i) \mid \mathit{cost}(m_p, \mathit{yield}(t_i)) \leq \mathfrak{b}_p^i\})\\
&\quad \mathit{in}\ (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in X_1, (p',c') \in X_2\}),\\
&\qquad MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in Y_1, (p',c') \in Y_2\}))\\
&= (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in MR^{--}(X_1),\\
&\qquad\qquad (p',c') \in MR^{--}(X_2)\}),\\
&\quad MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in MR^{+-}(Y_1),\\
&\qquad\qquad (p',c') \in MR^{+-}(Y_2)\}))\\
&= (MR^{--}(\{\ (p \cdot p', c + c') \mid (p,c) \in V_1, (p',c') \in V_2\}),\\
&\quad MR^{+-}(\{\ (p \cdot p', c + c') \mid (p,c) \in W_1, (p',c') \in W_2\}))
\end{aligned}
$$

On the other hand, we have

$$
\begin{aligned}
\mathit{INT}(t) \quad = \quad & \mathit{let}\ (V_i, W_i) = \mathit{INT}(t_i), i \in \{1,2\}\\
& \mathit{in}\ (MR^{--}(\{(p \cdot p', c + c') \mid (p,c) \in V_1, (p',c') \in V_2\}),\\
& \quad MR^{+-}(\{(p \cdot p', c + c') \mid (p,c) \in W_1, (p',c') \in W_2\}))
\end{aligned}
$$

and we conclude that $M(t) = \mathit{INT}(t)$.

2. Assume $t = \&_\vee(t_1, t_2)$. This case is analogous to the previous case hence omitted.

3. Assume $t = \&_\neg(t')$. Then, by inductive hypothesis it holds that

$$
\begin{aligned}
M(t', \mathit{yield}(t')) &= (MR^{--}(\{\ (f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}),\\
&\quad MR^{+-}(\{\ (f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\})) =\\
\mathit{INT}(t') &= (V, W)
\end{aligned}
$$

Let us first evaluate functions $f_1^{m_p}(t)$ and $f_2^{m_p}(t)$. From Lemma A.3 and

the fact that $\neg m^+ = m^-$, we have

$$\begin{aligned} f_1^{m_p}(t) &= \max\{\mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= \mathcal{P}[\![\&_\neg(t')]\!](m_p, m_o^+) \\ &= 1 - \mathcal{P}[\![t']\!](m_p, m_o^-) \\ &= 1 - \min\{\mathcal{P}[\![t']\!](m_p, m_o) \mid m_o \text{ B.a.}\} \\ &= 1 - f_2^{m_p}(t') \end{aligned}$$

Analogously, we have

$$f_2^{m_p}(t) = 1 - f_1^{m_p}(t')$$

Let us now evaluate $M(t)$ and $INT(t)$.

From Lemma A.7 and the inductive hypothesis, we have

$$\begin{aligned} M(t, \mathit{yield}(t)) &= (MR^{--}(\{\ (f_1^{m_p}(\&_\neg(t')), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (f_2^{m_p}(\&_\neg(t')), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\})) \\ &= (MR^{--}(\{\ (1 - f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}), \\ &\quad MR^{+-}(\{\ (1 - f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}), \\ &= \mathit{let}\ (X, Y) = (\{\ (f_1^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}), \\ &\qquad\qquad \{\ (f_2^{m_p}(t'), \mathfrak{b}_p) \mid \mathit{cost}(m_p, \mathit{yield}(t')) \leq \mathfrak{b}_p\}) \\ &\quad \mathit{in}\ (MR^{--}(\{\ (1-p, c) \mid (p, c) \in Y\}), \\ &\qquad MR^{+-}(\{\ (1-p, c) \mid (p, c) \in X\})) \\ &= (MR^{--}(\{\ (1-p, c) \mid (p, c) \in MR^{+-}(Y)\}), \\ &\quad MR^{+-}(\{\ (1-p, c+c') \mid (p, c) \in MR^{--}(X)\})) \\ &= (MR^{--}(\{\ (1-p, c) \mid (p, c) \in W\}), \\ &\quad MR^{+-}(\{\ (1-p, c) \mid (p, c) \in V\})) \end{aligned}$$

On the other hand, we have

$$\begin{aligned} \mathit{INT}(\&_\neg(t)) &= \mathit{let}\ (V, W) = \mathit{INT}(t) \\ &\quad \mathit{in}\ (MR^{--}(\{(1-p, c) \mid (p, c) \in W\}), \\ &\qquad MR^{+-}(\{(1-p, c) \mid (p, c) \in V\})) \end{aligned}$$

and we conclude that $M(t) = \mathit{INT}(t)$.

4. Assume $t = \&_\sim(t')$. This case is analogous to the case $t = \&_\neg(t')$ hence omitted.

□

C.5 Probabilistic Case with Multiple Costs

In the following we prove Theorem 4.6 given in Sect. 4.3.

Theorem C.5 *Let $t \in Tree$ be a linear attack-defence tree. Then*

$$M^*(t, yield(t)) = INT^*(t)$$

Proof.[Theorem 4.6] The proof is organised by structural induction on the shape of the tree t using Lemma A.9. It is analogous to the proof of Theorem 4.5. □

Appendix D

Detailed Evaluation of Attack-Defence Trees

This appendix contains the detailed evaluation of the attack-defence trees discussed in the examples of Ch. 4. Section D.1 shows the algorithmic evaluation of an attack-defence tree with probabilities, displayed in Figure D.1, while Sect. D.2 gives the algorithmic evaluation of an attack-defence tree with probability and a single-cost associated with basic actions, displayed in Figure D.2.

D.1 Probability Evaluation of Attack-Defence Trees

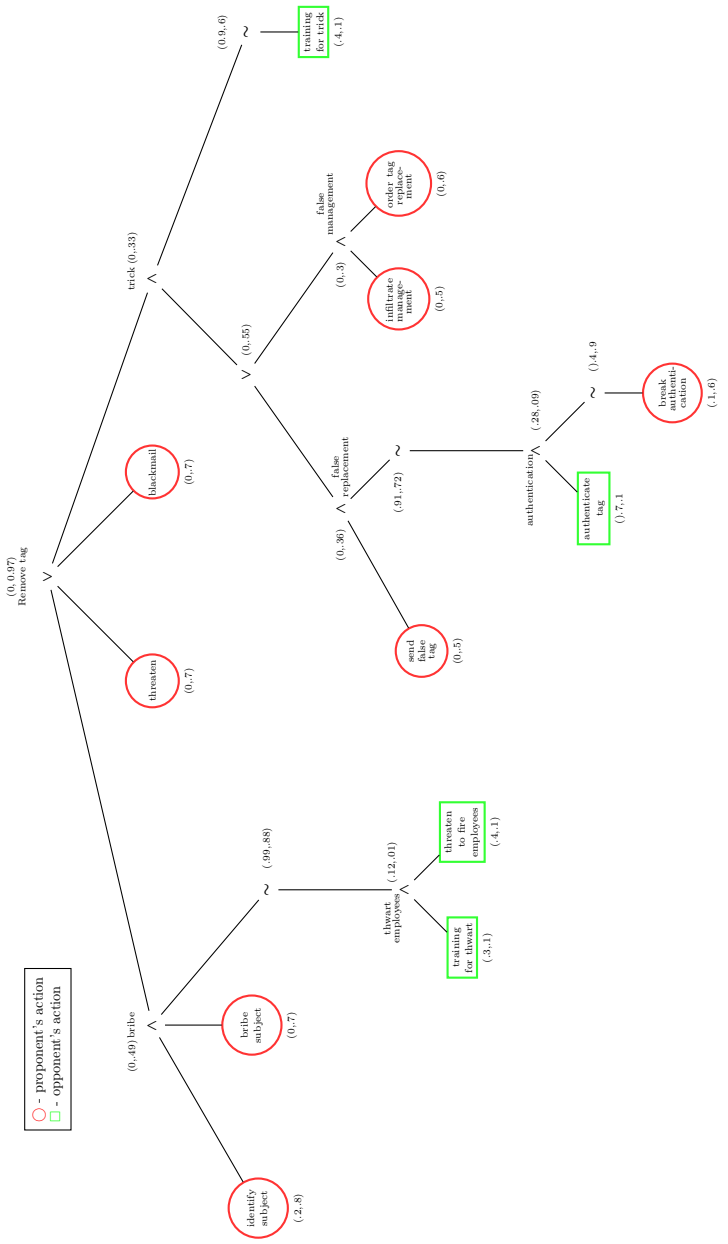


Figure D.1: The probabilistic algorithmic evaluation on the tree of Figure 4.1.

D.2 Cost Evaluation of Attack-Defence Trees

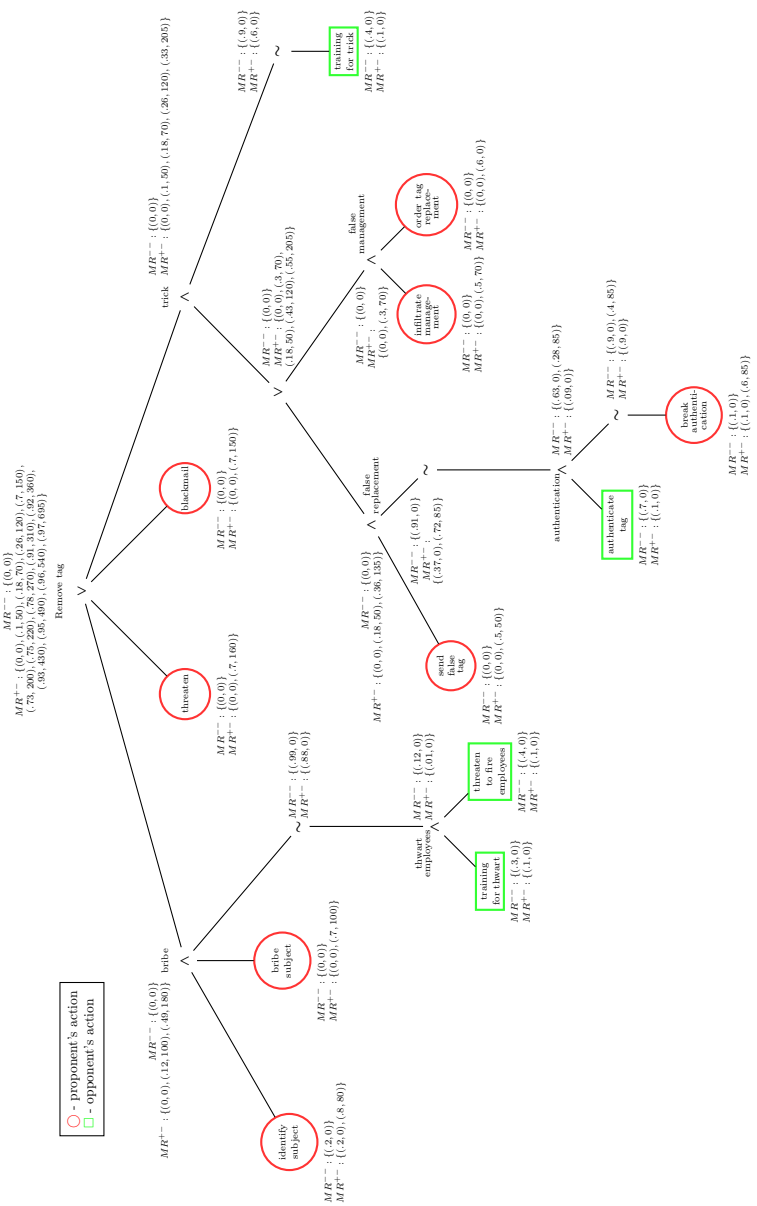


Figure D.2: The algorithmic evaluation on the tree of Figure 4.1.

Bibliography

[AHK03] Suzana Andova, Holger Hermanns, and Joost-Pieter Katoen. Discrete-time rewards model-checked. In *Formal Modeling and Analysis of Timed Systems: First International Workshop, FORMATS 2003, Marseille, France, September 6-7, 2003. Revised Papers*, pages 88–104, 2003.

[AHPS14] Florian Arnold, Holger Hermanns, Reza Pulungan, and Mariëlle Stoelinga. Time-dependent analysis of attacks. In *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 285–305, 2014.

[AINP15] Zaruhi Aslanyan, Marieta Georgieva Ivanova, Flemming Nielson, and Christian W. Probst. Modelling and analysing socio-technical systems. In *Proceedings of the 1st International Workshop on Socio-Technical Perspective in IS Development (STPIS'15) co-located with the 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015), Stockholm, Sweden, June 9, 2015.*, pages 121–124, 2015.

[Ame] Amenaza. SecurITree. `http://www.amenaza.com`.

[AN14] Zaruhi Aslanyan and Flemming Nielson. Pareto efficient solutions of attack trees. *Secure IT Systems*, page 279, 2014.

[AN15] Zaruhi Aslanyan and Flemming Nielson. Pareto efficient solutions of attack-defence trees. In *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*

2015, London, UK, April 11-18, 2015, Proceedings, pages 95–114, 2015.

[AN17] Zaruhi Aslanyan and Flemming Nielson. Model checking exact cost for attack scenarios. In *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, 2017.

[ANP16] Zaruhi Aslanyan, Flemming Nielson, and David Parker. Quantitative verification and synthesis of attack-defence scenarios. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 105–119, 2016.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM conference on Computer and communications security*, CCS'02, pages 217–224. ACM, 2002.

[Bai98] C. Baier. On algorithmic verification methods for probabilistic systems. Habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim, 1998.

[BDP06] Stefano Bistarelli, Marco Dall'Aglio, and Pamela Peretti. Strategic games on defense trees. In *Formal Aspects in Security and Trust, Fourth International Workshop, FAST 2006*, pages 1–15, 2006.

[BFM04] Eric J Byres, Matthew Franz, and Darrin Miller. The use of attack trees in assessing vulnerabilities in scada systems. In *Proceedings of the International Infrastructure Survivability Workshop*. Citeseer, 2004.

[BFP06] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense trees for economic evaluation of security investments. In *Availability, Reliability and Security*, pages 416–423, 2006.

[BHHK00] Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the logical characterisation of performability properties. In *Automata, Languages and Programming, 27th International Colloquium, ICALP 2000, Geneva, Switzerland, July 9-15, 2000, Proceedings*, pages 780–792, 2000.

[BK08] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, May 2008.

[BKMS12] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. Attribute decoration of attack-defense trees. *IJSSE*, 3(2):1–35, 2012.

[BLMW13] Fabrizio Biondi, Axel Legay, Pasquale Malacaria, and Andrzej Wąsowski. Quantifying Information Leakage of Randomized Protocols. In *14th International Conference Verification, Model Checking, and Abstract Interpretation (VMCAI'13)*, volume 7737 of *LNCS*, pages 68–87. Springer, 2013.

[BLP+06] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational choice of security measures via multi-parameter attack trees. In *Critical Information Infrastructures Security, First International Workshop, CRITIS 2006, Samos, Greece, August 31 - September 1, 2006, Revised Papers*, pages 235–248, 2006.

[BM07] Ahto Buldas and Triinu Mägi. Practical security analysis of e-voting systems. In *Advances in Information and Computer Security, Second International Workshop on Security, IWSEC 2007, Nara, Japan, October 29-31, 2007, Proceedings*, pages 320–335, 2007.

[BT91] Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, August 1991.

[Buc99] Peter Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theor. Comput. Sci.*, 215(1-2):263–287, 1999.

[Buc08] Peter Buchholz. Bisimulation relations for weighted automata. *Theor. Comput. Sci.*, 393(1-3):109–123, 2008.

[CCF04] Sean Convery, David Cook, and Matthew Franz. An attack tree for the border gateway protocol. *Work in Progress*, 2004.

[CFK+13a] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[CFK+13b] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, pages 266–277, 2013.

[DA98] Luca De Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford, CA, USA, 1998. AAI9837082.

[EDRM] K.S. Edge, G.C. Dalton, R.A. Raines, and R.F. Mills. Using attack and protection trees to analyze threats and defenses to homeland

security. In *Military Communications Conference, MILCOM 2006. IEEE*, pages 1–7.

[ERG+07] Kenneth S. Edge, Richard A. Raines, Michael R. Grimaila, Rusty O. Baldwin, Robert W. Bennington, and Christopher E. Reuter. The use of attack and protection trees to analyze security for an online banking system. In *40th Hawaii International International Conference on Systems Science (HICSS-40 2007), CD-ROM / Abstracts Proceedings, 3-6 January 2007, Waikoloa, Big Island, HI, USA*, page 144, 2007.

[FKNP11] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011, Bertinoro, Italy, June 13-18, 2011. Advanced Lectures*, pages 53–113, 2011.

[Gir95] Jean-Yves Girard. Linear logic: Its syntax and semantics. In *Proceedings of the Workshop on Advances in Linear Logic*, pages 1–42. Cambridge University Press, 1995.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Asp. Comput.*, 6(5):512–535, 1994.

[HKKS16] Holger Hermanns, Julia Krämer, Jan Krcál, and Mariëlle Stoelinga. The value of attack-defence diagrams. In *Principles of Security and Trust - 5th International Conference, POST 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 163–185, 2016.

[IEMR10] George C. Dalton II, Kenneth S. Edge, Robert F. Mills, and Richard A. Raines. Analysing security risks in computer and radio frequency identification (RFID) networks using attack and protection trees. *IJSN*, 5(2/3):87–95, 2010.

[IPHK15a] Marieta Georgieva Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller. Attack tree generation by policy invalidation. In *Information Security Theory and Practice - 9th IFIP WG 11.2 International Conference, WISTP 2015 Heraklion, Crete, Greece, August 24-25, 2015 Proceedings*, pages 249–259, 2015.

[IPHK15b] Marieta Georgieva Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller. Transforming graphical system

models to graphical attack models. In *Graphical Models for Security - Second International Workshop, GraMSec 2015, Verona, Italy, July 13, 2015, Revised Selected Papers*, pages 82–96, 2015.

[Iso] Isograph. AttackTree+. `http://www.isograph.com/software/attacktree/`.

[JKM+15] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, pages 339–353, 2015.

[JSW02] S Jha, O Sheyner, and J Wing. Two formal analyses of attack graphs. In *Proceedings 15th IEEE Computer Security Foundations Workshop CSFW15*, pages 49–63, 2002.

[JW07] Aivo Jürgenson and Jan Willemson. Processing multi-parameter attacktrees with estimated parameter values. In *Advances in Information and Computer Security, Second International Workshop on Security, IWSEC 2007, Nara, Japan, October 29-31, 2007, Proceedings*, pages 308–319, 2007.

[JW08] Aivo Jürgenson and Jan Willemson. Computing exact outcomes of multi-parameter attack trees. In *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*, pages 1036–1051, 2008.

[JW09] Aivo Jürgenson and Jan Willemson. Serial model for attack tree computations. In *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, pages 118–128, 2009.

[JW10] Aivo Jürgenson and Jan Willemson. On fast and approximate attack tree computations. In *Information Security, Practice and Experience, 6th International Conference, ISPEC 2010, Seoul, Korea, May 12-13, 2010. Proceedings*, pages 56–66, 2010.

[Kha09] Parvaiz Ahmed Khand. System level security modeling using attack trees. In *Computer, Control and Communication, 2009. IC4 2009. 2nd International Conference on*, pages 1–6. IEEE, 2009.

[KKMS13] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. Adtool: Security analysis with attack-defense trees. In *Quantitative Evaluation of Systems - 10th International Conference, QEST*

2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings, pages 173–176, 2013.

[KMMS10] Barbara Kordy, Sjouke Mauw, Matthijs Melissen, and Patrick Schweitzer. Attack-defense trees and two-player binary zero-sum extensive form games are equivalent. In *Decision and Game Theory for Security - First International Conference, GameSec 2010, Berlin, Germany, November 22-23, 2010. Proceedings*, pages 245–256, 2010.

[KMRS10] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Foundations of attack-defense trees. In *Formal Aspects of Security and Trust - 7th International Workshop, FAST 2010*, pages 80–95, 2010.

[KMRS14] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Attack-defense trees. *J. Log. Comput.*, 24(1):55–87, 2014.

[KMS12] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative questions on attack-defense trees. In *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, pages 49–64, 2012.

[KNP07] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, pages 220–270, 2007.

[KNP11] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.

[KP12] Marta Z. Kwiatkowska and David Parker. Advances in probabilistic model checking. In *Software Safety and Security - Tools for Analysis and Verification*, pages 126–151. 2012.

[KPS11] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational aspects of attack-defense trees. In *Security and Intelligent Information Systems - International Joint Conferences, SIIS 2011, Warsaw, Poland, June 13-14, 2011, Revised Selected Papers*, pages 103–116, 2011.

[KPS14a] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. Dag-based attack and defense modeling: Don't miss the forest for the attack trees. *Computer Science Review*, 13-14:1–38, 2014.

[KPS14b] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. A probabilistic framework for security scenarios with dependent actions. In *Integrated Formal Methods - 11th International Conference, IFM 2014, Bertinoro, Italy, September 9-11, 2014, Proceedings*, pages 256–271, 2014.

[KPW16] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. Prism-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 560–566, 2016.

[KRS15] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In *Formal Modeling and Analysis of Timed Systems - 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*, pages 156–171, 2015.

[LGCM10] Julien Legriel, Colas Le Guernic, Scott Cotton, and Oded Maler. Approximating the pareto front of multi-criteria optimization problems. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 69–83, 2010.

[LL11] Wen-ping Lv and Wei-min Li. Space based information system security risk evaluation based on improved attack trees. In *Multimedia Information Networking and Security (MINES), 2011 Third International Conference on*, pages 480–483. IEEE, 2011.

[LW05] Kong-wei Lye and M. Jeannette Wing. Game strategies in network security. *International Journal of Information Security*, 4(1):71–86, 2005.

[MAV+13] Y. A. Mahmood, Alireza Ahmadi, Ajit Kumar Verma, Ajit Srividya, and Uday Kumar. Fuzzy fault tree analysis: a review of concept and application. *Int. J. Systems Assurance Engineering and Management*, 4(1):19–32, 2013.

[MDTG07] John Mallios, Stelios Dritsas, Bill Tsoumas, and Dimitris Gritzalis. Attack modeling of sip-oriented SPIT. In *Critical Information Infrastructures Security, Second International Workshop, CRITIS 2007, Málaga, Spain, October 3-5, 2007. Revised Papers*, pages 299–310, 2007.

[Mel10] Per Håkon Meland. Seamonster. 2007-2010.

[MG13] Franco Mastroddi and Stefania Gemma. Analysis of pareto frontiers for multidisciplinary design optimization of aircraft. *Aerospace Science and Technology*, 28(1):40–55, 2013.

[MGE+08] Per Håkon Meland, Spampinato Daniele Giuseppe, Hagen Eilev, Baadshaug Egil Trygve, Krister Kris-Mikael, and Velle Ketil Sandanger. Seamonster: Providing tool support for security modeling. In *NISK 2008. Norsk informasjonssikkerhetskonferanse, Universitetet i Agder, Kampus Gimlemoen, November 17-19, 2008*, 2008.

[MO05] Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, pages 186–198, 2005.

[MRY11] Chris YT Ma, Nageswara SV Rao, and David KY Yau. A game theoretic study of attack and defense in cyber-physical systems. In *Computer Communications Workshops (INFOCOM WKSHPS), 2011 IEEE Conference on*, pages 708–713. IEEE, 2011.

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A scalable approach to attack graph generation. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS'06, pages 336–345. ACM, 2006.

[Pau14] Stéphane Paul. Towards automating the construction & maintenance of attack trees: a feasibility study. In *Proceedings First International Workshop on Graphical Models for Security, GraMSec 2014, Grenoble, France, April 12, 2014.*, pages 31–46, 2014.

[PB10] Ludovic Piètre-Cambacédès and Marc Bouissou. Beyond attack trees: Dynamic security modeling with boolean logic driven markov processes (BDMP). In *Eighth European Dependable Computing Conference, EDCC-8 2010*, pages 199–208, 2010.

[PRI] PRISM Model Checker. Available at `http://www.prismmodelchecker.org`.

[PS98] Cynthia Phillips and Laura Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proceedings of the 1998 workshop on New security paradigms NSPW 98*, volume pages, pages 71–79, 1998.

[RKT10a] Arpan Roy, Dong Seong Kim, and Kishor S Trivedi. Act: attack countermeasure trees for information assurance analysis. In *INFOCOM IEEE Conference on Computer Communications Workshops, 2010*, pages 1–2. IEEE, 2010.

[RKT10b] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Cyber security analysis using attack countermeasure trees. In *Proceedings of the 6th Cyber Security and Information Intelligence Research Workshop, CSIIRW 2010*, page 28, 2010.

[RKT12] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Attack countermeasure trees (ACT): towards unifying the constructs of attack and defense trees. *Security and Communication Networks*, 5(8):929–943, 2012.

[Sah08] Diptikalyan Saha. Extending logical attack graphs for efficient vulnerability analysis. In *Proceedings of the 2008 ACM Conference on Computer and Communications Security, CCS 2008, Alexandria, Virginia, USA, October 27-31, 2008*, pages 63–74, 2008.

[Sch99] Bruce Schneier. Attack Trees: Modeling Security Threats. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29, 1999.

[Sch04] Bruce Schneier. *Secrets and Lies: Digital Security in a Networked World*. John Wiley & Sons, 2004.

[Sha53] Lloyd S Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

[SHJ+02] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated Generation and Analysis of Attack Graphs. In *2002 IEEE Symposium on Security and Privacy*, pages 273–284, 2002.

[SSSW98] Chris Slater, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a secure system engineering methodolgy. In *Proceedings of the 1998 Workshop on New Security Paradigms, Charlottsville, VA, USA, September 22-25, 1998*, pages 2–10, 1998.

[SW04] Oleg Sheyner and Jeannette Wing. Tools for Generating and Analyzing Attack Graphs. In *2nd International Symposium on Formal*

Methods for Components and Objects (FMCO'03), volume 3188 of *LNCS*, pages 344–371. Springer, 2004.

[The14] The TREsPASS Project. `https://www.trespass-project.eu`, 2014.

[TLG07] Chee-Wooi Ten, Chen-Ching Liu, and Manimaran Govindarasu. Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8. IEEE, 2007.

[Umm11] Michael Ummels. *Stochastic multiplayer games: theory and algorithms*. PhD thesis, RWTH Aachen University, 2011.

[VBY13] Roberto Vigo, Alessandro Bruni, and Ender Yüksel. Security games for cyber-physical systems. In *Secure IT Systems - 18th Nordic Conference, NordSec 2013, Ilulissat, Greenland, October 18-21, 2013, Proceedings*, pages 17–32, 2013.

[VNN14] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Automated generation of attack trees. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 337–350, 2014.

[VRHG81] W.E. Vesely, N.H. Roberts, D.F. Haasl, and F.F Goldberg. *Fault Tree Handbook*. Number v. 88 in Fault Tree Handbook. Systems and Reliability Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1981.

[Wei91] Jonathan D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, pages 572–581, 1991.

[Yag06] Ronald R. Yager. OWA trees and their role in security modeling using attack trees. *Inf. Sci.*, 176(20):2933–2959, 2006.

[ZKSY14] Saman A. Zonouz, Himanshu Khurana, William H. Sanders, and Timothy M. Yardley. RRE: A game-theoretic intrusion response and recovery engine. *IEEE Trans. Parallel Distrib. Syst.*, 25(2):395–406, 2014.