Attack–Defense Trees

Patrick Schweitzer

Supervisor:
Prof. Dr. Sjouke Mauw (University of Luxembourg)

Daily advisor:
Dr. Barbara Kordy (University of Luxembourg)

uni.lu
UNIVERSITÉ DU
LUXEMBOURG

Fonds National de la
Recherche Luxembourg

The author was employed at the University of Luxembourg and received support from the Fonds National de la Recherche, Luxembourg (PHD-09-167) in the project "Security Analysis through Attack–Defense Trees".

UNIVERSITÉ DU
LUXEMBOURG

PhD-FSTC-2013-28
The Faculty of Sciences, Technology and Communication

DISSERTATION

Presented on 8 November 2013 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN

INFORMATIQUE

by

Patrick SCHWEITZER

Born on 25 January 1980 in Saarbrücken–Dudweiler (Germany)

ATTACK–DEFENSE TREES

Dissertation defense committee

Dr. Barbara Kordy, vice-chairman
Université du Luxembourg

Dr. Sjouke Mauw, dissertation supervisor
Professor, Université du Luxembourg

Dr. Christian W. Probst, member
Assistant Professor, The Technical University of Denmark

Dr. Yves Le Traon, chairman
Professor, Université du Luxembourg

Dr. Jan Willemson, member
Cybernetica, Estonia

Summary

The advent of the information age has notably amplified the importance of security. Unfortunately security considerations still widely occur as an afterthought. For many companies, security is not a requirement to conduct business and is therefore readily neglected. However the lack of security may obstruct, impede and even ruin an otherwise flourishing enterprise. Only when internal computer networks shut down, web portals are inaccessible, mail servers are attacked, or similar incidents affect the day to day business of an enterprise, security enters into the field of vision of companies. As such, security by design is only slowly becoming accepted practice.

Amongst security researchers, there is no dispute that a reasonable approach towards uninterrupted business activities includes security measures and controls from the beginning. To support these efforts, many security models have been developed. *Graphical security models* are a type of security model that help illustrate and guide the consideration of security throughout the lifecycle of a product, system or company. Their visual properties are especially well-suited to elucidate security requirements and corresponding security measures.

During the last four years, we have developed a new graphical security model called *attack–defense trees*. The new framework, presented in this thesis, generalizes the well-known attack trees model. Attack–defense trees formally extend attack trees and enhance them with defenses.

To be able to deploy attack–defense trees as a security support tool, we have equipped them with *three different syntaxes*: A visually appealing, graph-based syntax that is dedicated to representing security problems, an algebraic, term-based syntax that simplifies correct, formal and quantitative analysis of security scenarios and a textual syntax that is a compromise between succinct, visual representation and easy, computerized input.

We have also equipped attack–defense trees with a variety of semantics. This became necessary, since different applications require different interpretations of attack–defense trees. Besides the very specific and problem oriented *propositional*, *De Morgan* and *multiset* semantics, we have introduced *equational semantics*. The latter semantics is, in fact, an alternative, unified presentation of semantics based on equational theory. We have expressed the propositional and the multiset semantics in terms of the equational semantics. This facilitates algorithmic treatment since the two different semantics have a unified formal foundation.

To be able to perform quantitative security analysis, we have introduced the notion of an *attribute* for attack–defense trees. To guarantee that the evaluation of an attribute on two or more semantically equal attack–defense trees results in the same value, we have introduced the notion of a *compatibility condition* between semantics and attributes. We have also provided usability guidelines for attributes. These guidelines help a user to specify security-relevant questions that can unambiguously be answered using attributes.

We have performed several *case studies* that allowed us to test and improve the attack–defense tree methodology. We have provided detailed explanations for our design choices during the case studies as well as extensive applicability guidelines that serve a prospective user of the attack–defense tree methodology as a user manual.

We have demonstrated the usefulness of the formal foundations of attack–defense trees by *relating attack–defense terms to other scientific research disciplines*. Concretely, we have shown that attack–defense trees in the propositional semantics are computationally as complex as propositional attack trees. Moreover, we have described how to merge Bayesian networks with attack–defense trees and have illustrated that attack–defense trees in the propositional semantics are equivalent to a specific class of games frequently occurring in game theory.

Concluding the thesis, we have related the attack–defense tree methodology to other graphical security models in an extensive *literature overview* over similar methodologies.

Acknowledgments

Writing a thesis is, like life, a never-ending learning process. Moments of joy are equally part of the experience as failures and setbacks. Besides having to pull myself (and the horse I was sitting on) out of the self-induced chaos by my own hair, I was frequently helped and supported by numerous people all of whom I wish to thank.

First, I want to mention my daily advisor, Barbara Kordy. I cherish the unwaivering endurance that she showed throughout the years. Her devotion to teach me *the* precise and intelligible writing is admirable.

Next, I wish to extend my gratitude to Sjouke Mauw. It was a great pleasure to be part of his research group. His enthusiasm for and capability to inspire individuals to perform theoretical research continues to fascinate me. I appreciate the many valiant conversational attempts to enlighten me in aspects of Dutch humor, which *unfortunately* still remain a mystery to me.

I would also like to thank the members of my defense committee, Christian W. Probst, Yves Le Traon and Jan Willemson. I am grateful for the time and effort they spent on improving the quality of my research results. Furthermore, I would like to thank Björn Ottersten for valuable advice given as a member of my thesis supervisory committee.

Many members of SaToSS as well as SnT made life in Luxembourg extremely enjoyable. In particular, I would like to acknowledge Xihui Chen, Ton van Deursen, Naipeng Dong, Hugo Jonker, Piotr Kordy, Jean Lancrenon, Matthijs Melissen, Tim Muller, Marc Pouly, Saša Radomirović and Miguel Urquidi for scientific discussions, memorable Xmas parties and extended aquatic sessions.

I am honored to have such great friends as André Berthe, Johannes Hess and Sebastian Reißmann who managed to, time and again, show me different perspectives on life and unerringly urged me to go on.

I'd like to extend my special appreciation to Matty McConchie for thoroughly proofreading the entire manuscript.

I give heartfelt thanks to my parents, my Bomi, Annette and Renate for their never-ending moral support. Moreover, I wish to thank Tamaris Zwickler for always lending me a shoulder to cry on and for never complaining about the long hours that it took to compose this thesis.

I want to express my deepest gratitude to my brother Pascal, who not only fought his way through the first draft of this thesis but also encouraged me to wake up an hour earlier than him every morning. Finally, I wish to acknowledge my Volk for always smiling not matter what the circumstances.

Patrick Schweitzer
Luxembourg, November 2013

Contents

Nomenclature

○	Generic attack node	14
□	Generic defense node	14
$\triangle$	Generic subtree	101
$\langle\ \rangle$	Algebraic structure	34
$\{\!\!\{\ \}\!\!\}$	Multiset brackets	39
$\cdot$	Concatenation of strings	18
$\downarrow$	Projection of a configuration onto a subset	33
$\neg$	Unary negation function, also Boolean negation	33
$\overline{s}$	Negation of type s	19
$\otimes$	Distributive product of sets of bundles	39
$+_k$	Addition of any number of variables bounded by k	71
$\mathcal{M}^*$	Set of finite strings over $\mathcal{M}$	17
$\uplus$	Multiset union	39
$\approx$	Equivalent propositional formulas	32
$::=$	Production rule in BNF grammar notation	25
$\equiv$	Equivalent Boolean functions	33
$\equiv$	Equivalent De Morgan valuations	37
$\preceq$	Partial order	34
$\vdash$	Syntactic consequence	41
Σ	The AD–signature	19
β	Basic assignment function	59
ε	The empty string	17
$\equiv_{\mathcal{DM}}$	Equivalence in the De Morgan semantics	37
$\equiv_E$	Equivalence in the equational semantics	41
$\equiv_{\mathcal{M}}$	Equivalence in the multiset semantics	40
$\equiv_{\mathcal{P}}$	Equivalence in the propositional semantics	32
$\vee^{\mathrm{p}}$	Disjunctive refinement of proponent type	19
$\wedge^{\mathrm{p}}$	Conjunctive refinement of proponent type	19
$\vee^{\mathrm{o}}$	Disjunctive refinement of opponent type	19
$\wedge^{\mathrm{o}}$	Conjunctive refinement of opponent type	19
c^{p}	Countermeasure operator of proponent type	19
c^{o}	Countermeasure operator of opponent type	19
A	Attribute domain for ADTerms	58
$\mathcal{ADT}$	Grammar that generates all propositional ADTerms	45
$\mathcal{AT}$	Grammar that generates all propositional attack terms	141
$\mathcal{A}$	Set of query answers	144
$\mathbb{B}$	Set of all basic actions	19
$\mathbb{B}^{\mathrm{o}}$	Set of all basic actions of the opponent	18
$\mathbb{B}^{\mathrm{p}}$	Set of all basic actions of the proponent	18

BN_t	Bayesian network associated to the ADTerm t	149
$C[\,]$	Context: typed term with a hole	41
$\mathcal{DADT}$	Grammar that generates all propositional ADTerms in disjunctive form	46
D	Value domain for ADTerms	58
$E_{\mathcal{M}}$	Complete set of axioms for the multiset semantics	43
$E_{\mathcal{P}}$	Complete set of axioms for the propositional semantics	43
$\mathbb{E}$	Expected value	116
$\mathcal{E}$	Set of propositional formulas	45
$\mathtt{false}$	Boolean value false	31
$\mathcal{F}$	Set of all functional symbols in the AD–signature	19
$\tilde{\mathcal{F}}$	Flattened set of all functional symbols in the AD–signature	19
$\mathcal{G}$	Set of all two-player binary zero-sum games in extensive form	164
$\iota(T)$	ADTerm corresponding to the ADTree T	21
$\mathrm{I}(t)$	ADTree corresponding to the ADTerm t	21
$\mathcal{M}(S)$	Set of all multisets of elements of S	39
$\mathbb{N}$	Natural numbers.	
$\mathbb{N}^+$	Positive natural numbers excluding 0.	
o	Opponent	14
own	Owner of a question	75
par	Parents	148
$\mathcal{P}(S)$	Powerset of S	39
p	Proponent	14
$\mathbb{R}_{\geq 0}$	Positive real numbers including 0.	
$\mathbb{R}$	Real numbers.	
$\mathbb{R}_{\infty}$	Real numbers and ∞.	
$\mathbb{R}^+$	Positive real numbers excluding 0.	
$\mathbb{R}^+_{\infty}$	Positive real numbers and ∞ excluding 0.	
Sym	Set of all bijections from $\{1, \ldots, k\}$ to itself	41
$\mathcal{S}$	Set containing the types p and o	19
$\mathbb{T}_{\Sigma}$	Set of all ADTerms	20
$\mathbb{T}^{\mathrm{o}}_{\Sigma}$	Set of all ADTerms of opponent type	20
$\mathbb{T}^{\mathrm{p}}_{\Sigma}$	Set of all ADTerms of proponent type	20
$\mathbb{T}^{\mathrm{VAR}}_{\Sigma}$	Set of typed ADTerms over the variables in VAR	40
$\mathtt{T}$	ADTree in textual syntax	25
$\mathtt{true}$	Boolean value true	31
T	ADTree in graphical syntax	18
t	ADTerm, i.e., ADTree in term syntax	20
t_{α}	Term used to evaluate the attribute α on the ADTerm t	40
$t_{\mathcal{DM}}$	De Morgan ADTerm t	36
$t_{\mathcal{M}}$	ADTerm t in multiset semantics	40
$t_{\mathcal{P}}$	Propositional ADTerm t	30
var	Set of propositional variables	40
VAR	Set of typed variables	40
$\mathbf{x}$	Configuration	33

X^G	Set of propositional variables that correspond to all basic actions of the proponent type in the propositional semantics	45
x or x_b	Propositional variables	30
$x_{\text{Basic Action}}$	Propositional variable associated with a basic action labeled “Basic Action”	30
Y^G	Set of propositional variables that correspond to all basic actions of the proponent type in the propositional semantics	45
Attribute	Attributes are typeset in bold font	55
Consensus	Values resulting from the consensus meeting are typeset in bold font	112
Design Choice	Preferred design choices of the ADTree methodology are typeset in bold font	129
“Label”	Node labels are surrounded by quotation marks	15
`Value`	Attribute values are typeset in typewriter font	57

List of Figures

List of Tables

1

Introduction

Security has long become a part of our daily life. We have lockable drawers, doors, windows and safes in our homes. And we have a large corresponding bunch of keys. Our wallets are filled with identity documents, membership cards and bank notes that are all equipped with numerous security features.

Security has also ceased to only be physical. To be able to use bank and credit cards, we require personal identification numbers, we need to enter a pattern to access smart phones and almost all electronic services ask us to provide a password or another form of digital identification.

Compared to physical security, the field of information technology (IT) security is relatively young. In the early development stages, many computer systems have been claimed to be secure, but were compromised shortly after. The evolution of security continues to strive for an acceptable protection level without overly inconveniencing the end user. These efforts are complicated by the ever growing complexity of the systems that IT security aims to protect and the skill that potential attackers have. This battle between attackers and defenders is known as the IT arms race.

Naively, fighting against attacks on complex IT systems appears to be an insurmountable challenge. However, there are three developments that support security specialists in protecting systems from attacks.

Graphical models First, visual models support human creativity and understanding. The phrase *a picture says more than a thousand words* succinctly characterizes our brain's capabilities. When visualized, even the most intricate concepts and detailed facts can be relayed, explained and understood.

Formal models Second, another tool that helps us cope with complexity is the use of formal models. Long past being in their infancy, formal methods are precise and exact. Researchers have learned how to capitalize on formal methods by specifying only the necessary aspects and abstracting away from everything else when creating models.

Paradigm shift Third, a paradigm shift in IT security changes the focus of the models as well as the modeler: Security analysts have realized that the complete prevention of all attacks is costly if not even infeasible. Zero-day attacks, i.e., attacks that exploit previously unknown vulnerabilities, are inevitable and can only be thwarted heuristically. Therefore, security experts are now not only advocating measures that impede attacks from succeeding, but are also encouraging to mitigate the effect of any attack that has not been prevented. Abstractly speaking the paradigm shift introduces an additional layer of security. This, in turn, reemphasizes the importance of adequate graphical and formal security models since they are now used in two protection mechanisms.

In this introduction, we first elaborate on graphical models (Section 1.1) and formal models (Section 1.2) before we can formulate the research question (Section 1.3) and adequately express the contribution of this thesis (Section 1.4). We provide a more detailed outline of the thesis (Section 1.5) and shortly illustrate how other research interests have benefited this thesis (Section 1.6).

1.1 Graphical Security Modeling

Graphical security models, such as attack trees [SSSW98], provide a powerful method to visualize and examine security vulnerabilities of systems, organizations or even scenarios. In our framework, we understand a scenario as a series of actions and events that affect the security of systems or organizations. Graphical security models constitute a valuable support tool to facilitate threat assessment and risk analysis of real-life systems, during the planning, the development and the maintenance phase of a system's lifecycle. Graphical security models also provide an intuitive methodology to visualize possible attacks and countermeasures and to enable the computation of security related parameters.

Despite the advancement of many analysis possibilities that are now embedded in graphical security methodologies, their main focus still lies on creating a model that visually represents a system or a scenario. Hence, the most important requirement for a graphical methodology is its intuitiveness, especially with respect to presentation features. As such, they have become popular in the industrial sector as a means to visualize and support threat analysis and risk management. Due to their simplicity, graphical models are especially well-suited for an interdisciplinary context, where different cooperating partners may not be familiar with each other's conventions and working language [JEBR10]. Application domains of graphical models include security analysis of supervisory control and data acquisition (SCADA) systems, voting systems, vehicular communication systems, Internet related attacks, secure software engineering and socio-technical attacks [TLFH01, BFM04, ABD$^+$06, BM07, RVOC08, HAF$^+$09, TA10, EPPC11].

In 1991, Weiss [Wei91] introduced *threat logic trees* as the first graphical attack modeling technique. The apparent similarities of threat logic trees to fault trees [VGRH81] suggest that graphical security modeling has its roots in safety modeling. The essential difference between safety and security modeling is that security models consider preventing threats originating from active or malicious actions, whereas safety models address accidental system failures or events with potential impacts on the system environment [PCB13]. The nature of the different threat models is the reason that graphical security modeling and graphical safety modeling are nowadays considered as two largely separate disciplines, led by separated communities developing their own tools and methodologies. However, Weiss's approach can be considered to be the origin of graphical security modeling. Numerous subsequent models are based on Weiss's original idea. In the late 1990s, graphical security models were popularized by Schneier [SSSW98, Sch99]. He introduced the wide-spread graphical security methodology *attack trees*, a formalism that largely resembles Weiss's threat logic trees.

Graphical approaches have attracted the attention of numerous security and formal methods experts and are now quickly becoming a stand-alone research area with dedicated national and international research projects [SHI10a, TRE16, ANI14, ATR12]. As a result, a myriad of different approaches exist. Some of them extend the original methodology of threat logic trees in various dimensions. These dimensions include defensive components [BFP06], timed and ordered actions [WJ10] as well as dynamic aspects [PCB10a]. Other approaches are based on popular modeling techniques, such as UML [SO00]. Yet again, others are ad hoc approaches that arise from attempting to model the security of specific applications in particular contexts [ABS06].

In security analysis, an emerging subfield is concerned with quantifying security. In other words, besides qualitative questions, such as "Is it possible to attack a system?", the focus of the subfield lies on quantitative questions, such as "What are the minimal costs to protect a server?" The result of quantification is often a single key figure that can provide an indication on the general state of the system, rather than a complex mathematical function that describes the system's behavior. Security quantities are deduced from the available raw data. In practice, numerous different methods to compute security related parameters, such as the costs, the impact or likelihood of an attack, the efficiency of necessary protection measures or the environmental damage of an attack, have been developed.

While the great advantage of a graphical modeling approach lies in its user friendliness and its capability to intuitively visualize scenarios, graphical modeling is first and foremost an ad hoc procedure. It is often neither structured nor systematic. At best the approaches are loosely governed by modeling rules. These, however, are indispensable when the models grow in size. Therefore, graphical approaches by themselves do not suffice.

1.2 Formal Security Modeling

When modeling a mature and complex system with a purely visual approach that depicts every part of the system, the model can become cluttered, muddled and confusing. If there exists only a detailed graphical model, and no alternative representation, it may be difficult to quickly decide whether a costly defensive measure implemented in the past is still necessary today. Problems like "What are the best defensive measures currently worth investing in?" or "How can newly discovered attacks and implemented defenses be efficiently and systematically documented?" require a structured, systematic approach.

Formal approaches, such as term rewriting [BN98], are, by definition, designed to overcome the mentioned problems. Correctly specified formal models are structured, systematic, have a clear methodology, sound mathematical foundations and are often well-equipped to handle qualitative and quantitative analysis [Win90]. They are precise and unambiguous and typically constructive. Contrary to intuitive models, formal frameworks aim at preventing ambiguity and are able to support automated processing and automated quantitative evaluation [Win90].

A formal model consists of a clearly defined syntax and an explicit semantics. Attributes, sometimes also called metrics, are used to answer questions such as: "Is it possible to protect the system?", "How much would it cost to prevent one or all attacks?" or "How long does it take to secure the entire system?"

Despite all benefits of formal approaches, they are, not seldom, unintuitive, difficult to understand and hard to apply. Most of the formal methodologies are specialized, using their own notation and, therefore, require expert knowledge [BH95, BH06]. This knowledge is not only necessary for creating models, but even for understanding and interpreting them. As a result, it has to be acquired before a formal approach can be applied successfully [BH06].

Over the course of the last two decades there has been a growing tendency to advocate graphical models as part of formal approaches instead of treating them separately [BH06]. This development comes in many flavors. It is, for example, possible to formalize an already existing, purely visual methodology [GGR93]. Similarly, techniques have been developed to better illustrate and visualize formal methods [BH06]. Naturally, it is also possible to construct entirely new approaches having both the formalization and the intuitive visualization, in mind.

1.3 The Research Question

In this thesis, we combine a graphical with a formal approach. The abundance of related approaches in Chapter 7 demonstrates that the idea of improving an already existing graphical and formal security formalism is not a novel idea. It is a strategy as old as formalisms themselves. One of the reasons to continually improve a security methodology is that none of the existing approaches can model all vulnerable systems or security scenarios. Most methodologies are, in fact, limited to small, dedicated application domains. As such, they do not meet the demands of security analysts for a general, versatile formalism.

Nonetheless, all-purpose security methodologies exist. One such methodology is the formalism of attack trees that uses an underlying graph structure. Attack trees have proven their worth in numerous case studies and applications and are widely accepted in the industry, for references see Section 7.4.1.

The attack tree formalism can informally be described as follows. For a security scenario, the main goal of the attacker is depicted by the root node of a tree. Two types of refinement relations help to detail the model. Disjunctive refinements elaborate on attack options and conjunctive refinements to specify necessary steps, called actions, of the attacker. Both types of refinement are recursively applied. The procedure may stop at any time when the actions are sufficiently specified. This allows for a customizable level of abstraction. Generally speaking, the more refinements are depicted in a tree, the more precise the model. We illustrate an attack tree in Figure 1.1. Attack actions are depicted by labeled, red, circular nodes. The root is disjunctively refined into two subgoals. The Disjunctive Subgoal 2 is conjunctively refined into two subgoals. To distinguish conjunctive from disjunctive refinements, an arc connects the edges between a node and its conjunctively refining children.

Initially, the attack tree formalism lacked formalization and could not provide a precise meaning for a given scenario [SSSW98]. This was remedied by Mauw and Oostdijk who have formalized syntax and semantics of attack trees in 2005 [MO05]. They also analyzed under which conditions quantification with the help of attack trees is consistent with a simple bottom-up procedure. More recently, Whitley et al. extended and diversified the quantification procedure by proposing a novel way to compose disjunctive nodes [WWPP11].

In 2010, Willemson and Jürgenson proposed to extend attack trees by defining an order on the nodes [WJ10]. Their approach allowed them to improve quantitative analysis. They also showed that their extension is consistent with the semantics defined by Mauw and Oostdijk.

A notable step that structurally extends attack trees was proposed by Bistarelli et al. in 2006 [BDP06]. Their approach includes defenses as leaves into the attack tree formalism and views interactions between attackers and defenders as a game. In the same year, Edge et al. proposed protection trees. Instead of attack nodes, these trees only contain defensive nodes. Consequently, protection trees can be seen as being dual to attack trees. Over the last years, further approaches have incorporated defenses in the attack tree formalism. Baca and Petersen have proposed countermeasure graphs [BP10], which are similar to defense trees. Countermeasure graphs extend attack trees by attaching countermeasures to the leaves of the tree. Baca and Petersen use hierarchical cumulative voting in their quantification methods. Another approach that combines attack trees with defenses has been prosed by [RKT12a]. Their so-called attack countermeasure trees distinguish between attack events, detection events and mitigation events, which all appear as leaf nodes.

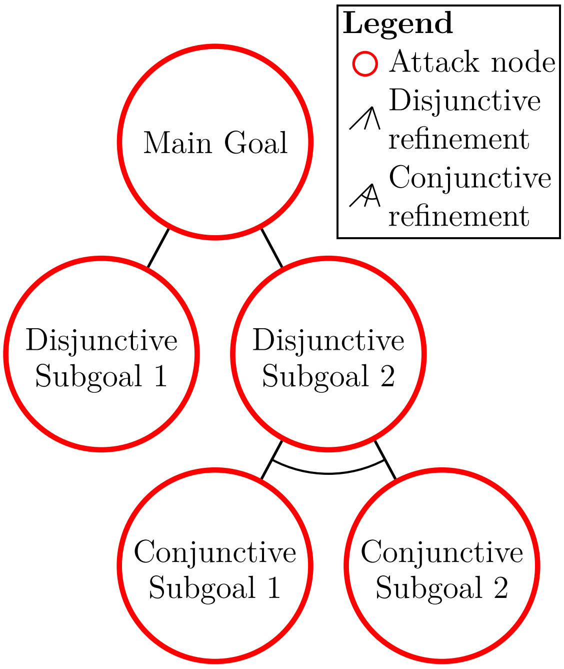


Figure 1.1: A generic attack tree.

In summary, attack trees in general and defenses in particular have already gained wide interest. They have proven their value in practice, as we show in a more thorough analysis of related approaches, which can be found in Chapter 7. However, no attack tree-based security analysis methodology with interleaved defenses existed at the start of this research project. Moreover, the existing approaches that include defenses do not differentiate between semantics and quantitative analysis and are not checking for consistency. Our aim is to overcome these deficiencies and amend the attack tree approach with novel features. We pose the following research question.

Research question: How can we extend attack trees with defenses to be able to provide a security analysis methodology that is both formal and graphical?

There is a long and a short answer to this question. The short one is the *attack–defense tree methodology.* The long answer is this thesis.

For the development of a new graphical security methodology we follow certain design criteria. By choosing attack trees as the underlying model, we have already limited the scope of this thesis to focus on modeling of *attack and defense* scenarios. We understand attacks and defenses in a general sense: they consist of any malicious action of an attacker who wants to harm or damage another party or its assets as well as any defense or countermeasure that could be used to prevent or mitigate the aforementioned malicious actions.

The choice of the attack tree formalism calls for a justification. First, attack trees are already well-known in the security area. Hence, by basing a model on attack trees, we can benefit from their familiarity. Consequently, as an extension of attack trees, attack–defense trees are more likely to be accepted by end users compared to an entirely new methodology. Additionally, any existing attack tree model can simply be reused and expanded. Second, their simplicity makes it ideal for graphical security modeling. As already mentioned before, combining a visual and a formal model should respect the main concepts of both worlds to maximally benefit from the fusion. By slightly extending the attack tree formalism and supplying it with formal foundations, we benefit in two ways. On the one hand, the models remain intuitive and can be interpreted by anyone, not only by specialists. On the other hand, the formal component of the model is ideally suited for algorithmic treatment and computer support.

Besides being based on attack trees, there were several other design choices that we followed when creating the versatile *attack–defense tree methodology.*

1. The methodology is supposed to capture attacker as well as *defender* actions by allowing alternation between attack and defense nodes. Consequently, it is possible to analyze which sets of defenses are optimal from different perspectives.

2. The approach should not only capture potential attack and defense actions, but it should allow for *interleaving* them. Interleaving enables the user to consider the evolution of a system's security. It especially clarifies why certain defenses were put in place and whether they are still necessary or whether they have become obsolete due to more advanced and more powerful attacks or defenses.

3. The modeling capabilities should be extended while the *complexity* of the formalism should be restricted to a minimum. This entails that the simple tree structure of attack trees is kept.

4. The model description should closely resemble *natural language.* Artificial constructions should be avoided in the visual representation of the model. In other words, any visual representation feature should serve a purpose and not (only) exist for technical purposes.

5. Finally, as already stated in the main research question, we chose to *extend the attack tree approach.* Moreover, we require it to be backward compatible, so that any attack tree is also a model in our formalism. This implies that

the attack tree displayed in Figure 1.1 has an identical representation in the attack–defense tree formalism.

1.4 Contribution

We designed a graphical model called *attack–defense trees* consisting of a *syntax*, a *semantics* and formal analysis techniques for *quantitative analysis*. Compared to other models, the formalism is unique since it is the only tree-based formalism that allows for interleaving of attacks and defenses. It is the only formalism that is equipped with *various semantics* to handle attacks as well as defenses. Moreover, the attack–defense tree methodology is the only attack tree-based formalism with a strict *separation of semantics and quantitative analysis*. It is both *visual and formal* and a translation between the two representations is provided. The methodology has been *tested* in practical case studies and *applied* in theoretical research.

A complete list of the author's scientific contributions can be found at the end of the thesis. This publication index is separated from the general bibliography and includes further research activities performed by the author during the course of his doctoral studies. All references to the author's own work start with a year, contrary to other references, which end in a year.

1.5 Thesis Structure

To answer the research question while respecting the restrictions detailed in Section 1.3, we

- define a syntax for attack–defense trees in Chapter 2;
- define a semantics for attack–defense trees in Chapter 3;
- treat quantitative analysis for attack–defense trees in Chapter 4. This chapter is based on the previous two chapters, however, some parts of it can be understood with only knowledge of the syntax for attack–defense trees;
- verify the usefulness of the methodology twofold. We examine the methodology's usefulness practically with the help of case studies in Chapter 5. We also demonstrate the usefulness of the formalization by relating attack–defense trees to concepts of other mathematical disciplines in Chapter 6;
- provide an extensive literature review that surveys attack and defense modeling approaches based on directed acyclic graphs in Chapter 7.

The dependencies between the chapters are illustrated in Figure 1.2. The thesis generally assumes knowledge equivalent to graduate level studies in theoretical computer science and basic knowledge of formal methods. Moreover, to understand the description of the syntax and the semantics a sound knowledge on term rewriting and typed terms is beneficial. For an introduction to term rewriting we refer to the book by Baader and Nipkow [BN98], for an introduction to typed terms, we refer to [CK00]. For a more extensive treatment, we refer to [Zan91, Zan00, Ohl02]. Finally, basic knowledge about general Bayesian networks [Pea88] is beneficial when they are used in combination with attack–defense trees in Section 6.2.

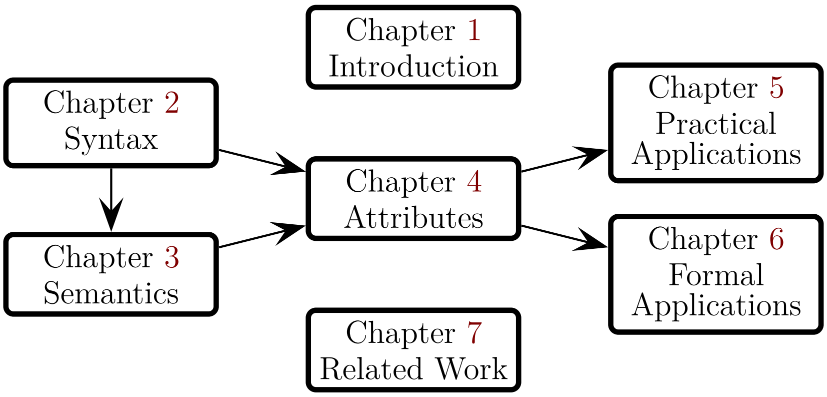


Figure 1.2: Dependencies between the chapters of the thesis.

Chapter 2: Syntax In this chapter, we introduce three different kinds of syntax for attack–defense trees. More precisely, we describe the visual attack–defense tree description, the algebraic attack–defense terms and a textual syntax that retains a formal character while also providing a visually appealing layout. We specify the connections and transformations between the different syntaxes and provide an introductory example that we turn into a running example. Finally, we explain benefits of the choices we made while designing the attack–defense tree language.

The syntax description is based on work with Barbara Kordy, Saša Radomirović and Sjouke Mauw. Attack–defense trees and attack–defense terms have been introduced in the conference paper [10KMRS] as well as the subsequent and expanded journal publication [12KMRS]. The textual syntax and related concepts were developed in collaboration with Sjouke Mauw.

Chapter 3: Semantics After introducing the syntax, we describe various semantics for attack–defense trees along with several usage scenarios. More specifically, we define the propositional semantics, suitable to answer recursively defined binary questions. We specify the class of De Morgan semantics. Each De Morgan semantics is an extension of the propositional semantics and is suitable in scenarios where we have two or more discrete outcomes. Finally, we provide the multiset semantics that are adequate when repetitive actions need to be considered.

We then introduce an axiomatic approach to semantics, by defining semantics with the help of an equational theory. This approach is then applied to express the propositional and the multiset semantics in terms of an equational theory. The so-called equational semantics are structures that allow us to relate different semantics to each other and are suitable to be implemented in algorithms and software.

The De Morgan semantics have been introduced in [11KPS]. The rest of this chapter is, to a small extent, based on [10KMRS] and, to a large extent, based on [12KMRS].

Chapter 4: Attributes This chapter covers attributes, which serve to model quantitative aspects on attack–defense trees. We provide a historical perspective on attributes before we introduce them formally as questions together with a bottom-up evaluation algorithm. We illustrate the computation with the help of several examples and advance a compatibility criterion that relates attributes and semantics.

Then we turn our attention to practical considerations and provide an empirical analysis on how to provide unambiguous answers to informal questions. We provide guidelines on how to transform the intuitive questions into an appropriate formal form. This transformation is also valid on attack trees. We conclude with methods that explain how to construct new attributes from existing ones.

The historical overview is based on [12BKMS]. The definitions and the compatibility criterion in this chapter originate from [10KMRS] and [12KMRS]. The examples of the attributes arose from work with Barbara Kordy and Jean-Paul Weber. The empirical analysis of attributes in the literature is based on a conference paper [12KMS] and the associated technical report [12KMSTec] while the constructions of new attributes are unpublished work.

Chapter 5: Practical Applications During the development cycle, attack–defense trees have continually been evaluated with the help of targeted case studies. This chapter illustrates the realization and the results of the case studies. We first investigated what kind of case study is suitable in the context of attack–defense trees. We then conducted internal case studies of which we present one. The experience gained from the initial case studies was then incorporated into comprehensive case studies on attributes.

As the outcome of the experiments, we provide observations that we made during the case studies and guidelines that assist during the application of the attack–defense tree methodology. Finally, we present a software tool that facilitates the work with the attack–defense tree methodology.

This chapter is based on results from a start-up workshop with an industrial partner and several internal case studies. One of these case studies concerning attributes has been published as a journal paper [12BKMS]. This journal paper also contains the guidelines on how to apply the attack–defense tree methodology. The software tool was designed in cooperation with Barbara Kordy and implemented by Piotr Kordy [12KSADTMan]. The presentation of the software tool is based on an early version of [13KKMS] and the extended technical report [13KKMSTec].

Chapter 6: Formal Applications In this chapter, we demonstrate the usefulness of the formalization of attack–defense trees. We analyze computational complexity aspects of the propositional as well as the class of De Morgan semantics and elaborate on implications for query evaluations and the applicability of present and future attack tree algorithms for attack–defense trees.

We also show how to combine the attack–defense tree methodology with other methodologies. More concretely, we combine attack–defense trees with Bayesian networks to be able to quantify probabilistic scenarios with dependent actions. Moreover, we present a summary of an explicit connection between attack–defense trees and game theory by showing that attack–defense trees in the propositional semantics are equivalent to a specific class of games.

The results about the complexity considerations have been published in a conference paper [11KPS]. The concept of combining Bayesian networks and attack–defense trees is under submission [13KPSPro] and the relation between attack–defense trees and a certain class of games has been a collaboration with Matthijs Melissen and has been published in a conference paper [10KMMS] and in extended form in a technical report [10KMMSTec].

Chapter 7: Related Work This chapter surveys related graphical security models. The emphasis lies on DAG-based models while other popular graphical security models are only shortly summarized.

The chapter is based on a survey which was submitted for publication [13KPSFor].

1.6 Further Research

Apart from graphical security models the author worked in three distinct research areas. We summarize the contributions and elaborate on the impact of the extracurricular activities on the work in graphical security models.

Trust models In asymmetric interactions over the Internet, an agent (the subject) is dependent on another agent (the target), but not vice versa. The subject should, therefore, form an opinion about the target, before possibly initiating an interaction. The scenario wherein a subject only relies on information obtained from past interactions with one target is well-studied and understood. In [12MS], we generalize the setting to allow for targets consisting of several parties. In particular, we formally derive conjunction and disjunction of trust opinions. In a follow-up paper [13MS], we generalize the initial setting differently by allowing recommendations (statements of a third party) as a source of information. We formalize a set of assumptions about trust with recommendations which allows us to identify and analyze the family of valid models that admit recommendations.

Research in trust models has helped the author gain a better understanding for formal methods and security modeling in general.

Graph theory In [10SS], we show that any face hitting set of size n of a connected planar graph with a minimum degree of at least 3 is contained in a connected subgraph of size $5n-6$. Furthermore, we show that this bound is tight by providing a lower bound in the form of a family of graphs. Our proof is valid for simple graphs with loops and generalizes to graphs embedded in surfaces of arbitrary genus.

In a paper currently under submission [13MSS], we have proven two conjectures about competition graphs. The competition graph of a directed acyclic graph D is the undirected graph on the same vertex set as D in which two distinct vertices are adjacent if they have a common out-neighbor in D. The competition number of an undirected graph G is the least number of isolated vertices that have to be added to G to make it the competition graph of a directed acyclic graph. The first conjecture by Opsut [Ops82] is proven by showing that the competition number of every quasi-line graph is at most 2. Recall that a quasi-line graph, also called a locally co-bipartite graph, is a graph for which the neighborhood of every vertex can partitioned into at most two cliques. To prove this conjecture we devise an alternative characterization of quasi-line graphs to the one by Chudnovsky and Seymour [CS05]. The second conjecture by Kim [Kim05] is proven by showing that the competition number of any graph is at most one greater than the number of holes in the graph. Our methods are even applicable to prove a strengthened form of this conjecture. Specifically, we show that the competition number of any graph is at most one greater than the dimension of the subspace of the cycle space spanned by the holes.

Working in mathematics and especially graph theory has helped the author to improve his knowledge of precise mathematical formalization.

Modeling uncertainty In the paper [10KS], we analyze the regulation of emissions of non-uniformly mixed pollutants under uncertainty. Commonly, regulation with a permit market carries the risk of hot spot formation, which can be reduced by dividing the regulation area into trading zones. The trading zone approach has been extensively discussed for the full-information case. We consider incomplete information concerning the emitters' abatement costs, their locations and pollution dispersion. We derive the optimal number of trading zones and the optimal number of permits per zone and analyze under which conditions a system of independent trading zones is superior to other policy measures. Our results show that appropriately sized permit markets are well-suited to regulate non-uniformly mixed pollutants under informational constraints if firms are not excessively heterogeneous. Only for substantial heterogeneity and a highly non-linear damage function can it be optimal to use command-and-control strategies.

This work has helped the author increase his knowledge about modeling in general and modeling with probabilistic variables in particular. It also introduced the author to academic research and research presentation.

2

Syntax and Definitions

The growing complexity of systems complicates their reliable operation. Since empirical testing was deemed to be insufficient, formal methods were devised to overcome this new challenge [CW96]. At the core of any formal method lies the syntax. The syntax describes a model's fundamental building blocks and stipulates how to combine them into a model. The syntax does not provide any interpretation, syntactical rules merely govern the visual appearance of the model. All syntaxes depicted in this thesis are typeset with a set of LaTeX macros, publicly available in a documented LaTeX package [12SADTSty].

In this chapter, we advance three different syntaxes for attack–defense methodology. First, in Section 2.1, we introduce the graphical syntax called attack–defense trees. The graphical syntax is well-suited to provide a quick and intuitive overview over the entire model. It is based on the syntax of attack trees, as introduced in [SSSW98]. Then, in Section 2.2 we introduce a term-based syntax called attack–defense terms. This syntax is based on the mathematical concept of terms. Terms are a convenient mathematical framework to devise properties of and prove results within the model. In Section 2.3 we show how attack–defense trees and attack–defense terms are linked by formalizing transformation rules between the two syntaxes. Sometimes, a trade-off between a visual and purely symbolic syntax can be helpful. We, therefore, present a third syntax, the textual syntax, in Section 2.4. Its primary use can be seen to serve as notation use in email exchanges between people that are unfamiliar with the notation of the term-based syntax. Finally, in Section 2.5, we elaborate on several choices that we were faced with during the design phase and the creation of the syntax of the attack–defense tree methodology. We elaborate on the choices and how they affected the syntax.

2.1 ADTrees

As already mentioned, attack–defense trees are a methodology to represent attack–defense scenarios. Attack–defense trees extend the well-known model of attack trees [SSSW98, Sch99] by incorporating defensive measures. In this section we first introduce the concept of attack–defense trees intuitively. We then illustrate their usage on an extended example, before we formally define attack–defense trees.

2.1.1 Defining ADTrees

An *attack–defense tree* (ADTree) is a node-labeled rooted tree describing measures an attacker may take in order to attack a system and defenses that a defender can employ to protect the system. ADTrees have nodes of two *opposite types*: *attack* nodes and *defense* nodes, which correspond to, respectively, an attacker's and a defender's goals or subgoals.

Two key features of an ADTree are the ability to represent *refinements* and *countermeasures*. Every node may have one or more children of the parent's type representing a refinement into *subgoals* of the node's goal. If a node does not have any children of the parent's type, it is called a *non-refined* node. Non-refined nodes represent so-called *basic actions*, i.e., actions which can be fully understood and easily quantified.

Every node may also have one child of opposite type of that of the parent, representing a countermeasure. Thus, an attack node may have several children which refine the attack and one child which defends against the attack. The defending child in turn may have several children which refine the defense and one child that is an attack node and counters the defense.

The refinement of a node of an ADTree is either disjunctive or conjunctive. The goal of a disjunctively refined node is achieved when *at least one* of its children's goals is achieved. The goal of a conjunctively refined node is achieved when *all* of its children's goals are achieved. It is also possible to view the two refinements as *different ways* and *different steps* to achieve the goal represented by the parent node, where no indication of the steps' order is necessary.

The purpose of ADTrees is to model attack–defense scenarios. An attack–defense scenario can be seen as a game between two players, the *proponent* (denoted by p) and the *opponent* (denoted by o). The root of an ADTree represents the main goal of the proponent. When the root is an attack node, the proponent is an attacker and the opponent is a defender. Conversely, when the root is a defense node, the proponent is a defender and the opponent is an attacker.

When drawing ADTrees, we depict attack nodes by (red) circles (○) and defense nodes by (green) rectangles (□), as shown in Figure 2.1. Refinement relations are indicated by solid edges between nodes and countermeasures are indicated by dotted edges. We depict a conjunctive refinement of a node by an arc over all edges connecting the node and its children of equal type.

2.1.2 An Introductory Example

To construct an ADTree, we first consider the goal of the attack–defense scenario. It forms the root of the tree. The goal is refined into subgoals represented by children of the root which in turn may get further refined. Counteracting measures employed or expected to be employed against a goal are included by inserting one child of the opposite type. This child node is then refined into specific goals and actions of the other type. Should there again be a measure which counteracts this goal, then the measure is inserted as a child of the measure and further refined, if necessary.

Example 2.1 To demonstrate the features of ADTrees, we consider the following fictitious scenario concerning data confidentiality in a data hosting center. The ADTree representing the scenario is shown in Figure 2.1. Its root node is a defense. Thus the proponent is the defender and his main goal expressed by the tree is the protection of “Data Confidentiality”. To ease matching of the textual scenario descriptions with the corresponding figures, we signify node labels with quotation marks.

In order to protect the confidentiality of costumer data, the hosting company needs to invest in “Network Security” as well as in physical security measures. These measures break up into several aspects that need to be taken care of. However, even if both of physical and network security were to be infallible, the company’s employees would still be a weak point. Two common options to subvert a company through its employees are “Corruption” and “Social Engineering”. These attacks can be mitigated through “Screening” the employees and “Sensitivity Training” against social engineering techniques.

Network security is a complex problem. It is beyond the purpose of this introductory example to show all possible attacks and defenses. Some standard measures employed in the context of network security are access control systems, firewalls and intrusion detection. Of these, we display the evolution of access control through the use of passwords. In many access controlled services, passwords used to be free of any restrictions regarding the type of characters they need to contain. Consequently, a significant number of passwords chosen consisted of a name or dictionary word since these are much easier to remember than a random sequence of characters. This has led to access control breaches through so-called dictionary attacks. In order to prevent these attacks, computer systems nowadays “Require Strong Passwords”, which need to contain letters, numbers and non-alphanumeric characters. This mechanism, however, causes people to write up their passwords on easily accessible sticky notes or to reuse the same strong password for different accounts and services. Thus, the strong password required for the data center may be recovered by attacking an unrelated and possibly weaker system on which the target user has an account or by finding a sticky note.

Regarding physical security, a building can be broken into through back doors, fire escapes or windows. It is, therefore, common to reinforce windows and to protect other entrances with locks. The locks can be circumvented by forcing them open or by acquiring a key. An increasingly common practice is, therefore, to employ security guards that monitor the building. In order to effectively monitor the building, a security guard will typically have the keys not only to the building itself, but also to all rooms in the building. This makes the security guard a possible attack vector. He could be bribed, subdued or his keys could be stolen in some manner. To overpower the guard, it would be necessary to outnumber him and then use a weapon to incapacitate him. To prevent these three attacks, video cameras with remote surveillance could be employed. These, in turn, could be sabotaged by an attacker.

The scenario as described thus far is obviously incomplete. It is clear, however, how to extend the ADTree shown in Figure 2.1 with new attacks and defenses when adapting the scenario. In Section 5.4.3 we discuss when it is reasonable to stop extending the tree and when to continued providing more details.

ADTrees were conceived as an extension of attack trees, which intend to describe the security of systems. Other elaborated tree models include “How to attack a bank account?” [10KMRS], “How to attack a server?” [12KMS] and “How to

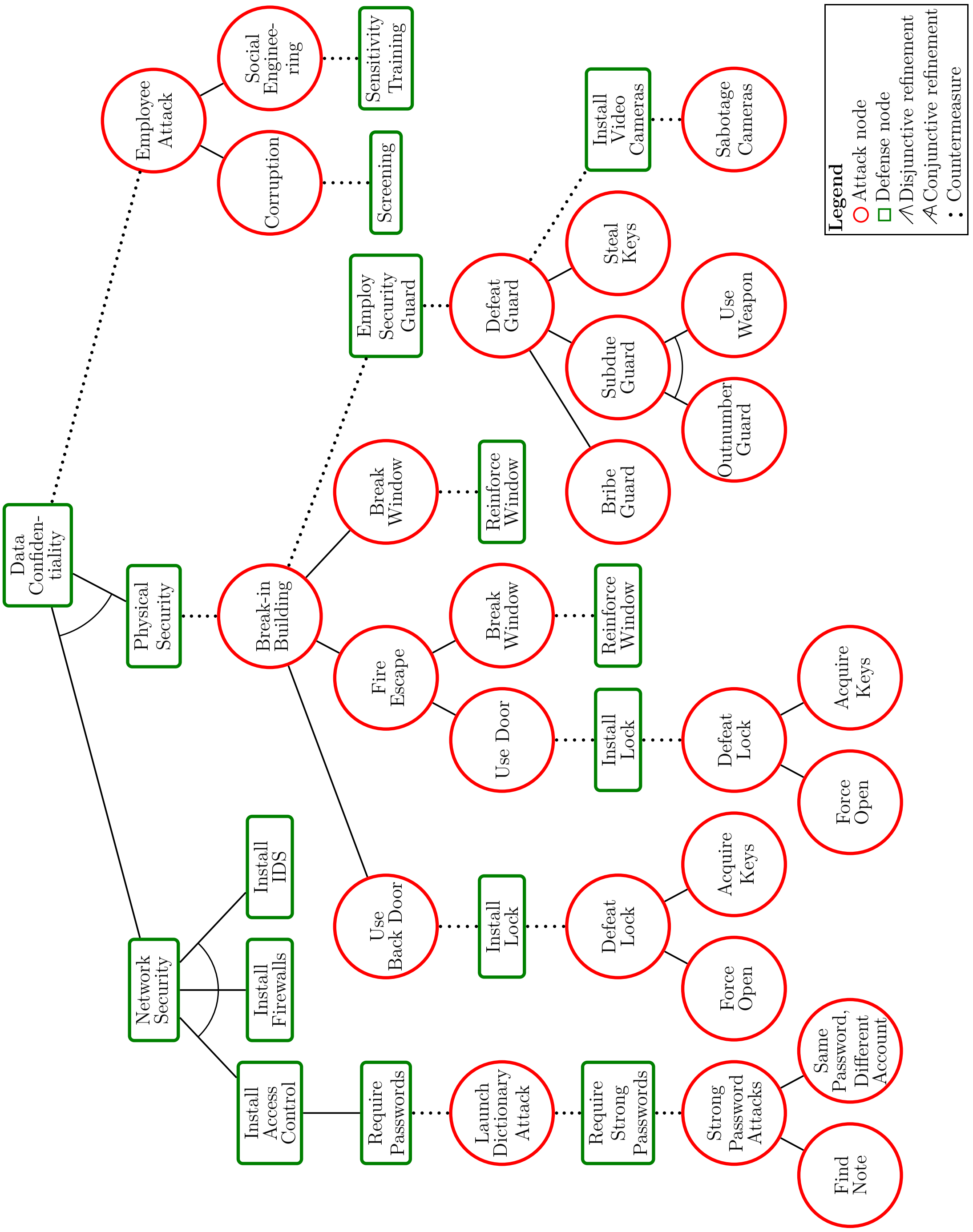


Figure 2.1: An ADTree modeling the scenario of protecting data confidentiality.

perform a distributed denial of service attack on an RFID system?" [12BKMS]. However, the application field of ADTrees is versatile. They are not restricted to only model computer security. We have created toy ADTree models for "How to steal a lion?", "How to break and enter into a warehouse?" and "How to open a locked door without raising suspicion?", which can all be found at [12KSADTLib].

2.1.3 A Running Example

The initial Example 2.1 given in the previous section visually demonstrates all graphical features of ADTrees. However, it is rather large and unwieldy. We, therefore, use a subtree as a running example. This subtree example has the added advantage that it demonstrates that ADTrees can also be rooted in attack nodes. This means that, since in this scenario the root is of attack type, the proponent is now the attacker and the opponent the defender.

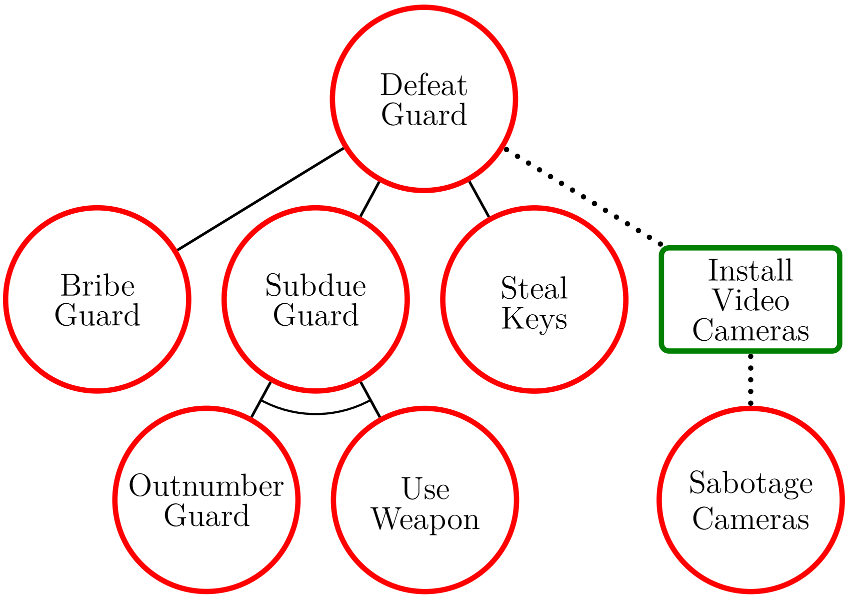


Figure 2.2: An ADTree for defeating a guard.

Example 2.2 Imagine we are only interested in defeating the guard mentioned in Example 2.1. In this case we can use the subtree rooted in the node "Defeat Guard" to illustrate the scenario. We use the following abbreviations: "Defeat Guard" (DG), "Bribe Guard" (BG), "Subdue Guard" (SG), "Outnumber Guard" (OG), "Use Weapon" (UW), "Steal Keys" (SK), "Install Video Cameras" (IC), "Sabotage Cameras" (SC).

2.1.4 A Formal Definition of ADTrees

The formal definition of an ADTree is based on the notion of finite ordered trees, as introduced in [CDG+07].

Given a set $\mathcal{M}$, we denote by $\mathcal{M}^*$ the set of all finite strings over $\mathcal{M}$ and by ε the empty string. We define a *finite ordered tree T over a set of labels L* as a function $T\colon \operatorname{Pos}(T) \to L$, where $\operatorname{Pos}(T)$ is a finite prefix-closed subset of $(\mathbb{N}^+)^*$

which is closed under decrementing the last entry. It is called the set of positions of T.

We depict T as a graph in the following manner. The positions in $\mathrm{Pos}(T)$ are drawn as nodes labeled with elements of L. The position ε is the root node of the graph, depicted as the topmost node. The positions $p \cdot 1, p \cdot 2, \dots, p \cdot k$ for some $k \in \mathbb{N}^+$, are the children of the node corresponding to the position p. (We use $\cdot$ to denote concatenation). Since T is ordered, the node corresponding to the position $p \cdot i$ is drawn left of the node depicting position $p \cdot j$ whenever $i < j$.

An ADTree is then formally defined as follows:

Definition 2.3 (ADTree) An attack–defense tree (ADTree) is a finite ordered tree T over the set of labels $L_T = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}} \cup \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, where $\mathbb{B}^{\mathrm{p}}$ denotes the set of basic actions of the proponent and $\mathbb{B}^{\mathrm{o}}$ denotes the set of basic actions of the opponent, together with a function $\lambda\colon \mathrm{Pos}(T) \to \{\bigcirc, \square\}$ which satisfies the following conditions for every $p \in \mathrm{Pos}(T)$.

1. If there exists $i \in \mathbb{N}^+$, such that $p \cdot i \in \mathrm{Pos}(T)$ and $\lambda(p \cdot i) = \lambda(p)$, then

$$T(p) \in \begin{cases} \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\} & \text{if} \quad \lambda(p) = \lambda(\varepsilon); \\ \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\} & \text{else.} \end{cases}$$

2. If there exists $i \in \mathbb{N}^+$, such that $p \cdot i \in \mathrm{Pos}(T)$ and $\lambda(p \cdot i) \neq \lambda(p)$, then $\forall j > i : p \cdot j \notin \mathrm{Pos}(T)$.

3. If $p \cdot 1 \in \mathrm{Pos}(T)$ and $\lambda(p \cdot 1) \neq \lambda(p)$ or there exists no i, such that $p \cdot i \in \mathrm{Pos}(T)$, then

$$T(p) \in \begin{cases} \mathbb{B}^{\mathrm{p}} & \text{if} \quad \lambda(p) = \lambda(\varepsilon); \\ \mathbb{B}^{\mathrm{o}} & \text{else.} \end{cases}$$

Finally, we require that on every path from a leaf to the root a basic action occurs at most once.

The function λ allows us to distinguish between attack nodes (○) and defense nodes (□). The value $\lambda(\varepsilon)$ determines for the considered tree which player (attacker or defender) is the proponent and which is the opponent. By comparing the values of λ applied to a parent node with the values of λ applied to its children, we can decide which nodes are refined and which non-refined. A node p is refined if it has the child $p \cdot 1$ and it holds that $\lambda(p) = \lambda(p \cdot 1)$. Then, Condition 1 of Definition 2.3 holds and the node p of an ADTree is either conjunctively or disjunctively refined ($T(p) \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$). Condition 2 states that a node can have at most one child $p \cdot j$ of opposite type. Such a child satisfies $\lambda(p) \neq \lambda(p \cdot j)$ and is always depicted as the rightmost child node of p. Finally, Condition 3 says that a non-refined node either has no children or exactly one child of opposite type and no children of the same type. For every non-refined node it holds that $T(p) \in \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$.

In the formal definition of ADTrees, refined nodes are labeled with the associated refining symbols. In practice, such nodes are typically labeled with descriptive names of the goals or subgoals they represent, as shown in Figure 2.1.

2.2 ADTerms

In order to formally analyze ADTrees, we define an abstract syntax which we call attack–defense terms [10KMRS]. To be able to capture ADTrees rooted in an attacker's node as well as a defender's node, we distinguish between the proponent, which we recall refers to the root player, and the opponent, which is the other player. For instance, for the ADTree in Figure 2.1, the proponent is the attacker and the opponent is the defender. Conversely, if the root of an ADTree is a defense node, the proponent is the defender and the opponent is the attacker. Therefore, the root of an ADTree always represents the main goal of the proponent.

Attack–defense terms are typed terms over a particular signature called the attack–defense signature (AD–signature). To define the AD–signature, we make use of the notion of an unranked function. An *unranked function* F with domain D and range R denotes a family of functions $(F_k)_{k\in\mathbb{N}^+}$, where $F_k\colon D^k \to R$, for $k \geq 1$. Often, when the rank of a function F_k can be deduced from the context, we abuse notation and use the unranked function symbol F, instead of a specific ranked function F_k.

Definition 2.4 (AD–signature) The AD–signature is a pair $\Sigma = (\mathcal{S}, \mathcal{F})$, where

- $\mathcal{S} = \{\mathrm{p}, \mathrm{o}\}$ is the set of types of the proponent and the opponent and
- $\mathcal{F} = \{(\vee^{\mathrm{p}}_k)_{k\in\mathbb{N}^+}, (\wedge^{\mathrm{p}}_k)_{k\in\mathbb{N}^+}, (\vee^{\mathrm{o}}_k)_{k\in\mathbb{N}^+}, (\wedge^{\mathrm{o}}_k)_{k\in\mathbb{N}^+}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\} \cup \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$ is a set of function symbols, such that $\{(\vee^{\mathrm{p}}_k)_{k\in\mathbb{N}^+}, (\wedge^{\mathrm{p}}_k)_{k\in\mathbb{N}^+}, (\vee^{\mathrm{o}}_k)_{k\in\mathbb{N}^+}, (\wedge^{\mathrm{o}}_k)_{k\in\mathbb{N}^+}, \mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\}$, $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are pairwise disjoint.

For technical reasons, we define the flattened set $\tilde{\mathcal{F}} = \bigcup_{k\in\mathbb{N}^+}\{\vee^{\mathrm{p}}_k\} \cup \bigcup_{k\in\mathbb{N}^+}\{\wedge^{\mathrm{p}}_k\} \cup \bigcup_{k\in\mathbb{N}^+}\{\vee^{\mathrm{o}}_k\} \cup \bigcup_{k\in\mathbb{N}^+}\{\wedge^{\mathrm{o}}_k\} \cup \{\mathrm{c}^{\mathrm{p}}, \mathrm{c}^{\mathrm{o}}\} \cup \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$. Every function symbol $F \in \tilde{\mathcal{F}}$ is equipped with a mapping $\mathrm{rnk}\colon \tilde{\mathcal{F}} \to \mathcal{S}^* \times \mathcal{S}$, called *rank*. The rank of a function symbol F is a pair $\mathrm{rnk}(F) = (\mathrm{arity}(F), \mathrm{type}(F))$, where the first component describes the *arity* of F and the second specifies its *type*. Given $F \in \tilde{\mathcal{F}}$ and $s \in \mathcal{S}$, we say that F is of type s if $\mathrm{type}(F) = s$. For the function symbols in $\tilde{\mathcal{F}}$ we define,

$$\begin{aligned}
&\mathrm{rnk}(b) = (\varepsilon, \mathrm{p}), \quad \text{for} \quad b \in \mathbb{B}^{\mathrm{p}}, && \mathrm{rnk}(b) = (\varepsilon, \mathrm{o}), \quad \text{for} \quad b \in \mathbb{B}^{\mathrm{o}},\\
&\mathrm{rnk}(\vee^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), && \mathrm{rnk}(\vee^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}),\\
&\mathrm{rnk}(\wedge^{\mathrm{p}}_k) = (\mathrm{p}^k, \mathrm{p}), && \mathrm{rnk}(\wedge^{\mathrm{o}}_k) = (\mathrm{o}^k, \mathrm{o}),\\
&\mathrm{rnk}(\mathrm{c}^{\mathrm{p}}) = (\mathrm{p\,o}, \mathrm{p}), && \mathrm{rnk}(\mathrm{c}^{\mathrm{o}}) = (\mathrm{o\,p}, \mathrm{o}),
\end{aligned}$$

where $k \in \mathbb{N}^+$.

The elements of $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are typed constants, which we call *basic actions of proponent type* and *basic actions of opponent type*, respectively. Since they do not take any argument as input, but provide an ADTerm as output, their ranks are $(\varepsilon, \mathrm{p})$ for basic actions of the proponent and $(\varepsilon, \mathrm{o})$, for basic actions of the opponent. We denote the set of all basic actions by $\mathbb{B} = \mathbb{B}^{\mathrm{p}} \cup \mathbb{B}^{\mathrm{o}}$. The unranked functions $\vee^{\mathrm{p}}$, $\wedge^{\mathrm{p}}$, $\vee^{\mathrm{o}}$ and $\wedge^{\mathrm{o}}$ represent disjunctive ($\vee$) and conjunctive ($\wedge$) refinement operators for the proponent and the opponent, respectively. The arity of a concrete operator, e.g., $\vee^{\mathrm{p}}_k$ is given k times the type of the root symbol (here p), followed by an argument of the same type. We set $\overline{\mathrm{p}} = \mathrm{o}$ and $\overline{\mathrm{o}} = \mathrm{p}$. The binary functions c^s, for $s \in \mathcal{S}$, represent countermeasures and are used to connect components of type s with components of the opposite type $\overline{s}$. The rank of c^{p} is $(\mathrm{p}\,\mathrm{o}, \mathrm{p})$ since the operator's inputs are a term of proponent type and a term of opponent type and it yields a term of proponent type. Similarly, the rank of c^{o} is $(\mathrm{o}\,\mathrm{p}, \mathrm{o})$.

The concept of AD–signatures allows us to define terms that are equipped with the typing function type while not containing any variables. These kind of terms are usually called *typed ground terms*.

Definition 2.5 (ADTerms) Finite, typed ground terms over the AD–signature Σ are called attack–defense terms (ADTerms). The set of all ADTerms is denoted by $\mathbb{T}_\Sigma$.

For $s \in \mathcal{S}$, we denote by $\mathbb{T}_\Sigma^s$ the set of all ADTerms with the root symbol of type s, i.e., the symbol at position ε is of type s. We have $\mathbb{T}_\Sigma = \mathbb{T}_\Sigma^{\mathrm{p}} \cup \mathbb{T}_\Sigma^{\mathrm{o}}$. The elements of $\mathbb{T}_\Sigma^{\mathrm{p}}$ and $\mathbb{T}_\Sigma^{\mathrm{o}}$ are called *ADTerms of proponent type* and *of opponent type*, respectively. The ADTerms of proponent type constitute formal representations of ADTrees. Finally, ADTerms which involve only operators $\vee^{\mathrm{p}}$ and $\wedge^{\mathrm{p}}$ are called *attack terms* and correspond to attack trees introduced in [SSSW98] and formalized in [MO05].

Example 2.6 Consider the ADTree given in Figure 2.2. To construct the corresponding ADTerm t we can limit the set of basic actions of the proponent to $\{\mathrm{BG}, \mathrm{OG}, \mathrm{UW}, \mathrm{SK}, \mathrm{SC}\} \subset \mathbb{B}^{\mathrm{p}}$ and the set of basic actions of the opponent to $\{\mathrm{IC}\} \subset \mathbb{B}^{\mathrm{o}}$. Then t is given by $t = \mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC}))$, The subterms $\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK})$, $\wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW})$, BG, OG, UW, SK and SC as well as the entire ADTerm t, are of proponent type. Term t also contains a subterm of opponent type, namely $\mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC})$. Note that the names of refined nodes in the ADTree, such as “Defeat Guard” and “Subdue Guard”, do not appear in the ADTerm. Instead, these nodes are represented with the corresponding refining symbols $\vee^{\mathrm{p}}$ and $\wedge^{\mathrm{p}}$.

Note that it is also possible to represent all ADTerms with the help of a grammar. From [GTWW77] we know that every term-algebra can be transformed into a context-free grammar.

Definition 2.7 (ADTerms grammar) Let $\mathbb{B}^{\mathrm{p}}$ be all basic actions of proponent type and $\mathbb{B}^{\mathrm{o}}$ be all basic actions of opponent type. We define the following production rules.

$$\begin{array}{llclclcl} P: & \mathbb{B}^{\mathrm{p}} & \mid & \vee^{\mathrm{p}}(P,\ldots,P) & \mid & \wedge^{\mathrm{p}}(P,\ldots,P) & \mid & \mathrm{c}^{\mathrm{p}}(P,O) \\ O: & \mathbb{B}^{\mathrm{o}} & \mid & \vee^{\mathrm{o}}(O,\ldots,O) & \mid & \wedge^{\mathrm{o}}(O,\ldots,O) & \mid & \mathrm{c}^{\mathrm{o}}(O,P). \end{array}$$

Then, the expressions generated from the start symbol P represent all ADTerms of proponent type and the expressions generated from the start symbol O represent all ADTerms of opponent type. The set of all ADTerms is the union of the ADTerms of proponent type and the ADTerms of opponent type.

2.3 Transformations between ADTrees and ADTerms

Both the intuitive model, detailed in Section 2.1, and the formal model, detailed in Section 2.2, can be used as different representations of the same scenario. The

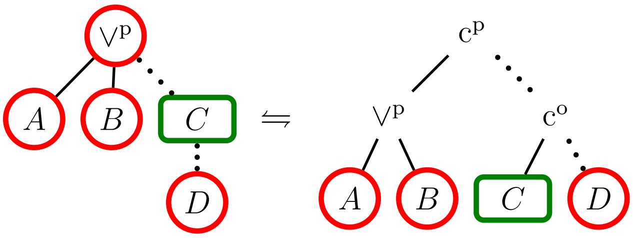


Figure 2.3: An ADTree (left) and a parse tree (right) of the corresponding ADTerm $\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(A, B), \mathrm{c}^{\mathrm{o}}(C, D))$.

intuitive ADTree model allows for quick visual inspections of the scenario, whereas the formal ADTerms allow for easier automated manipulation. We illustrate that ADTrees can be transformed into ADTerms and vice versa with the help of Figure 2.3.

As one might expect, the transformation involves the parse tree of an ADTerm. Due to the requirement of backwards compatibility with attack terms, this, however, is not the only step involved in the transformation. The complete procedure works as follows. Starting from an ADTree, we replace every node which has a countermeasure as a child by an additional node. This additional node is a countermeasure that has exactly two children: the original node's subtree without the countermeasure subtree and the countermeasure subtree. The modified tree then corresponds to a parse tree of the ADTerm that represents the same scenario as the initial ADTree.

Table 2.1 shows formally how ADTrees can be transformed into ADTerms and Table 2.2 shows the inverse direction. Given an ADTree T, we denote by $\iota(T)$ the ADTerm representing T. Given an ADTerm t, we denote by $\mathrm{I}(t)$ the corresponding ADTree. In Tables 2.1 and 2.2, we assume that the proponent is an attacker. If the proponent is a defender, circular nodes have to be replaced with rectangular nodes and vice versa. To improve readability, we omit the arcs, denoting conjunctions, in the cases where $f \in \{\wedge^{\mathrm{p}}, \wedge^{\mathrm{o}}\}$.

Remark 2.8 Since ADTerms do not contain labels of refined nodes, these will be lost when transforming ADTrees into ADTerms. However, since all relevant information is contained in the non-refined node, the labels of refined nodes may possibly be reconstructed from the labels of the child nodes or may simply be omitted.

2.4 Textual Syntax

A third way to represent ADTrees is to use a linear ASCII description. It can be employed for quick and easy display of ADTrees, for instance in an email exchange. For this purpose the following ASCII characters are used:

`( ) [ ] | - = #.`

To depict the type of a node we use the brackets. More specifically, the label of an attack node is enclosed between round brackets "`( )`", the label of a defense node in square brackets "`[ ]`". The "`|`" symbol helps to depict the structure of

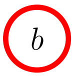
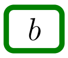
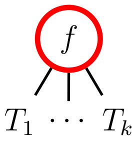
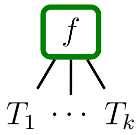
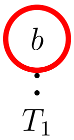
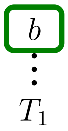
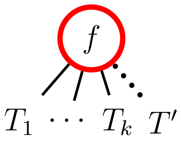
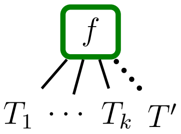
T	 where $b \in \mathbb{B}^{\mathrm{p}}$	 where $b \in \mathbb{B}^{\mathrm{o}}$	 where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	 where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\iota(T)$	b	b	$f(\iota(T_1), \ldots, \iota(T_k))$	$f(\iota(T_1), \ldots, \iota(T_k))$
T	 where $b \in \mathbb{B}^{\mathrm{p}}$	 where $b \in \mathbb{B}^{\mathrm{o}}$	 where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	 where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\iota(T)$	$\mathrm{c}^{\mathrm{p}}(b, \iota(T_1))$	$\mathrm{c}^{\mathrm{o}}(b, \iota(T_1))$	$\mathrm{c}^{\mathrm{p}}(f(\iota(T_1), \ldots, \iota(T_k)), \iota(T'))$	$\mathrm{c}^{\mathrm{o}}(f(\iota(T_1), \ldots, \iota(T_k)), \iota(T'))$

Table 2.1: Transformation rules from ADTrees to ADTerms.

the tree and the remaining symbols are used to express the connection between a parent node and its children. Each node is written on its own line adhering to the following structure.

The first line corresponds to the root node and contains only the label and the type of the root. All other lines correspond to intermediate nodes or leaves. They start with a number of “|” symbols, followed by “-”, “=” or “#” and end with the label of the node enclosed in brackets. On each such line the depth of the node in the tree is reflected by the amount of “|” symbols. The symbols “-”, “=” and “#” illustrate that this node is linked to its parent via a disjunctive refinement, a conjunctive refinement or a countermeasure, respectively.

The vertical structure of the ASCII description illustrates the parent–child relation between the nodes. The tree is set up from top to bottom and from left to right. Children of nodes are considered before siblings. The structure is created recursively. First, the line corresponding to the root is created. Then, for every parent node, lines corresponding to its children are inserted underneath the line corresponding to the parent. This construction is repeated until we reach lines corresponding to leaves. In other words, we mimic the execution of depth-first search.

The linear ASCII syntax for the ADTree from Example 2.2 is given in Figure 2.4.

Since white space should be insignificant in the textual ADTree syntax, there are representations of ADTrees that do not look intuitive. We, therefore, give some conventions for a proper layout:

- Every node is followed by a new line.
- The bar “|” and the connector symbols “-”, “=” and “#” are aligned vertically.
- Two vertically aligned bars may be connected to form a continuous figure,

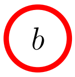
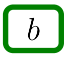
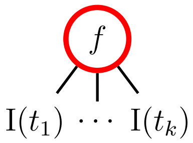
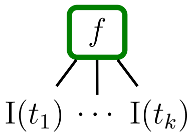
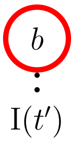
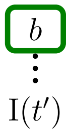
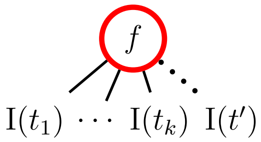
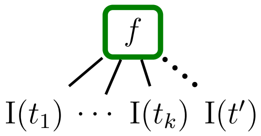
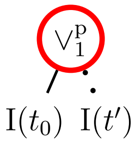
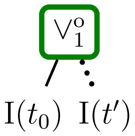
t	$b \in \mathbb{B}^{\mathrm{p}}$	$b \in \mathbb{B}^{\mathrm{o}}$	$f(t_1, \ldots, t_k)$, where $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	$f(t_1, \ldots, t_k)$, where $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\mathrm{I}(t)$				
t	$\mathrm{c}^{\mathrm{p}}(b, t')$, $b \in \mathbb{B}^{\mathrm{p}}$	$\mathrm{c}^{\mathrm{o}}(b, t')$, $b \in \mathbb{B}^{\mathrm{o}}$	$\mathrm{c}^{\mathrm{p}}(t_0, t')$, where $t_0 = f(t_1, \ldots, t_k)$ and $f \in \{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}\}$, $k \geq 1$	$\mathrm{c}^{\mathrm{o}}(t_0, t')$, where $t_0 = f(t_1, \ldots, t_k)$ and $f \in \{\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}\}$, $k \geq 1$
$\mathrm{I}(t)$			 f $\mathrm{I}(t_1)$ $\cdots$ $\mathrm{I}(t_k)$ $\mathrm{I}(t')$	 f $\mathrm{I}(t_1)$ $\cdots$ $\mathrm{I}(t_k)$ $\mathrm{I}(t')$
t			$\mathrm{c}^{\mathrm{p}}(t_0, t')$, where $t_0 = \mathrm{c}^{\mathrm{p}}(t_1, t_2)$	$\mathrm{c}^{\mathrm{o}}(t_0, t')$, where $t_0 = \mathrm{c}^{\mathrm{o}}(t_1, t_2)$
$\mathrm{I}(t)$				

Table 2.2: Transformation rules from ADTerms to ADTrees.

see Figure 2.5.

- The symbols denoting the connector type may also be connected to the vertically aligned bars.

- Brackets may be replaced with a red oval (in case of attack nodes) or green rectangles (in case of defense nodes) surrounding the node label.

Figure 2.5 depicts a visually more appealing version of Figure 2.2 that respects the above conventions.

```
(Defeat Guard)
|-(Bribe Guard)
|-(Subdue Guard)
||=(Outnumber Guard)
||=(Use Weapon)
|-(Steal Key)
|#[Install Video Camera]
||#(Sabotage Camera)
```

Figure 2.4: The linear ASCII syntax of Figure 2.2.

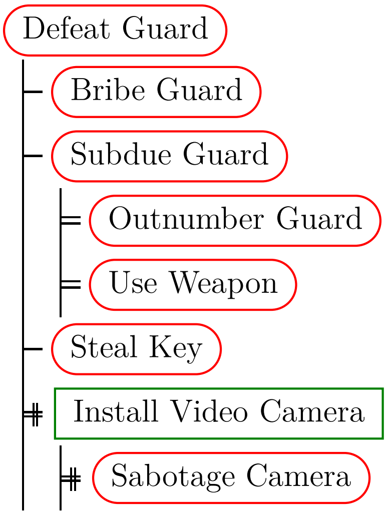


Figure 2.5: A textual ADTree for defeating a guard.

To reverse the procedure and recreate the ADTree from the textual syntax, we draw a node for every line in the syntax. We draw an attack node (○) for every line that ends in “`( )`” and a defense node (□) for every line that ends in “`[ ]`”. The node labels correspond to the description inside the parentheses or brackets. A node A is a child of a node B when the line corresponding to B has exactly one vertical bar “`|`” less than the line corresponding to node A and the line corresponding to node A is the first such line that we encounter after the line corresponding to B when traversing the textual description from bottom to top. The parent-child relationship indicates a countermeasure with a dashed line when the line corresponding to the node contains a “`#`”. Finally, when a line contains a “`=`” then the edges between the refining siblings and the parent are connected by an arc.

Transformations between the textual syntax and ADTerms are obtained by first transforming into ADTrees.

While the ADTerms are context-free (Definition 2.7), the textual syntax is not. To prove this fact, we use the pumping lemma for context-free languages [Gur89]. The proof relies on the “`|`” symbols that do not exist in an ADTerm. They indicate an absolute depth position within the tree which is not compatible with context-freeness. Intuitively speaking, without context, it is, for example, not possible to determine if the line `||=(Outnumber Guard)` can succeed another one. This line may follow `|-(Subdue Guard)`, as seen in Figure 2.4, but it cannot follow the line `(Defeat Guard)`.

Theorem 2.9 *The ASCII representation language of ADTrees is not context-free.*

Proof. We can reduce the ASCII syntax to the following language over two symbols. We write a 0 for nodes and a 1 for each “`|`” in front of the nodes. The other symbols (connectors and brackets) we leave out, which is possible since context-free grammars are closed under homomorphism. So defeating the guard, i.e., Example 2.2, would be modeled as 0 10 10 110 110 10 10 110. The language has the property that every zero which was preceded by n ones is followed by at most $n+1$ ones and never by a zero. Every string in the language terminates with a zero. Using the pumping lemma for context-free languages we can see that the language is not context-free:

Let p be the pumping length. Consider $uvxyz = 0101101110 \cdots 1^{2p}0$ with $|vy| > 0$ and $|vxy| \leq p$ (i.e., an ADTree with $2p$ empty refinements). Then, by the pumping lemma the strings uv^ixy^iz for $i \geq 0$ should be in the language. We make the following case distinction: If v or y contain only zeros or only ones, then choosing i large enough produces a string that is not in the language, since there cannot be any consecutive zeros and the number of ones is bounded. In the remaining case, v or y contains both zeros and ones. Then choosing $i = 0$ produces a string that is not in the language: Either v or y must contain the sequence 01 or 10. (Which means that $p \geq 2$.) Provided u or z are not empty, the number of ones preceding a zero right of v^0 or right of y^0 is at least two larger than to the left of the corresponding symbol, which yields a contradiction. Due to the pumping length u or z cannot both be empty. If u is empty, a similar reasoning as before holds since z starts with two or more continuous ones. If z is empty then, since $|vxy| \leq p$, y contains only ones or uxz ends in a one. This finishes the proof. □

A consequence of this theorem is that adding a line in the textual syntax may require knowledge about the rest of the syntax. A given line may be inserted between some lines, but not between others. Moreover, for algorithmic applications we recommend the use of ADTerms instead of the textual syntax, since ADTerms are constructed from a simple, context-free grammar.

The textual syntax of ADTrees can also be modeled as an Extended Backus–Naur Form (EBNF) grammar. This allows the parametrization of variables with natural numbers and elements of finite sets. We use the following notation to describe the syntax. In detail, we use $::=$ to denote productions, $\|$ to denote alternatives, $[\ldots]$ to denote optional constructs and $(\ldots)^*$ to denote repetition (0 or more times).

Let Id be a finite set of not further specified strings, denoting identifiers. We consider the following set of terminals: “ − ”, “ = ”, “#”, “|”, “(”, “)”, “[”, “]”, Id.

The grammar defines the following basic variables: Bar_n (for $n \in \mathbb{N}$) denoting a sequence of n vertical bars. The symbol Node^τ (for $\tau \in \{\bigcirc, \square\}$) denotes a node of type τ. Finally, Conn^φ (for $\varphi \in \{\mathtt{or}, \mathtt{and}, \#\}$) denotes a φ-connection. This yields:

$$\begin{array}{lll}
\mathrm{Bar}_0 ::= \varepsilon & \mathrm{Conn}^{\mathtt{or}} ::= \text{“ − ”} & \mathrm{Node}^{\bigcirc} ::= \text{“(” Id “)”} \\
\mathrm{Bar}_{n+1} ::= \text{“ | ”}\, \mathrm{Bar}_n & \mathrm{Conn}^{\mathtt{and}} ::= \text{“ = ”} & \mathrm{Node}^{\square} ::= \text{“[” Id “]”} \\
 & \mathrm{Conn}^{\#} ::= \text{“ \# ”}. &
\end{array}$$

The main variable defined by the grammar is $\mathtt{T}$, which denotes an ADTree in textual syntax. The auxiliary variable $\mathtt{T}_n$ represents an ADTree at depth $n \in \mathbb{N}$. The auxiliary variable $\mathtt{T}_n^\tau$ (for $\tau \in \{\bigcirc, \square\}$) represents an ADTree at depth $n \in \mathbb{N}$ of type τ. The auxiliary variable $\mathtt{T}_n^{\tau,\phi}$ (for $\tau \in \{\bigcirc, \square\}$ and $\phi \in \{\mathtt{or}, \mathtt{and}\}$) represents an ADTree at depth $n \in \mathbb{N}$ of type τ with connector ϕ. We use the notation $\overline{\bigcirc} = \square$

and $\overline{\square} = \bigcirc$.

$$
\begin{aligned}
\mathtt{T} &::= \mathtt{T}_0\\
\mathtt{T}_n &::= \mathtt{T}_n^{\bigcirc} \;\parallel\; \mathtt{T}_n^{\square}\\
\mathtt{T}_n^{\tau} &::= \mathtt{T}_n^{\tau,\mathtt{or}} \;\parallel\; \mathtt{T}_n^{\tau,\mathtt{and}}\\
\mathtt{T}_n^{\tau,\phi} &::= \mathrm{Node}^{\tau}\left(\mathrm{Bar}_{n+1}\ \mathrm{Conn}^{\phi}\ (\mathtt{T}_{n+1}^{\tau,\mathtt{or}} \parallel \mathtt{T}_{n+1}^{\tau,\mathtt{and}})\right)^{*}\left[\mathrm{Bar}_{n+1}\ \mathrm{Conn}^{\#}\ (\mathtt{T}_{n+1}^{\overline{\tau}\mathtt{or}} \parallel \mathtt{T}_{n+1}^{\overline{\tau}\mathtt{and}})\right].
\end{aligned}
$$

We derive the textual syntax for the Example 2.2 from the grammar in EBNF in Figure 2.6.

$$
\begin{aligned}
&\mathtt{T} \to \mathtt{T}_0 \to \mathtt{T}_0^{\bigcirc} \to \mathtt{T}_0^{\bigcirc,\mathtt{or}}\\
&\quad\to \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{and}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\\
&\qquad \mathrm{Bar}_1\ \mathrm{Conn}^{\#}\ \mathtt{T}_1^{\square,\mathtt{or}}\\
&\quad\to \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathtt{T}_2^{\bigcirc,\mathtt{or}}\\
&\qquad \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathtt{T}_2^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\#}\ \mathtt{T}_1^{\square,\mathtt{or}}\\
&\quad\to \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathtt{T}_2^{\bigcirc,\mathtt{or}}\\
&\qquad \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathtt{T}_2^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathtt{T}_1^{\bigcirc,\mathtt{or}}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\#}\ \mathrm{Node}^{\square}\\
&\qquad \mathrm{Bar}_2\ \mathrm{Conn}^{\#}\ \mathtt{T}_2^{\bigcirc,\mathtt{or}}\\
&\quad\to \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathrm{Node}^{\bigcirc}\\
&\qquad \mathrm{Bar}_2\ \mathrm{Conn}^{\mathtt{and}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\mathtt{or}}\ \mathrm{Node}^{\bigcirc}\ \mathrm{Bar}_1\ \mathrm{Conn}^{\#}\ \mathrm{Node}^{\square}\\
&\qquad \mathrm{Bar}_2\ \mathrm{Conn}^{\#}\ \mathrm{Node}^{\bigcirc}\\
&\quad\to \mathrm{Node}^{\bigcirc}\ |-\mathrm{Node}^{\bigcirc}\ |-\mathrm{Node}^{\bigcirc}\ ||=\mathrm{Node}^{\bigcirc}\ ||=\mathrm{Node}^{\bigcirc}\ |-\mathrm{Node}^{\bigcirc}\\
&\qquad |\,\#\,\mathrm{Node}^{\square}\ ||\,\#\,\mathrm{Node}^{\bigcirc}\\
&\quad\to (\text{Defeat Guard})\\
&\qquad |-(\text{Bribe Guard})\\
&\qquad |-(\text{Subdue Guard})\\
&\qquad ||=(\text{Outnumber Guard})\\
&\qquad ||=(\text{Use Weapon})\\
&\qquad |-(\text{Steal Key})\\
&\qquad |\,\#[\text{Install Video Camera}]\\
&\qquad ||\,\#(\text{Sabotage Camera}).
\end{aligned}
$$

Figure 2.6: Derivation of the running example in BNF syntax.

2.5 Design Choices

When designing the ADTree formalism, we have deliberately made the following choices in order to keep a balance between usability, complexity and representational impact.

1. **Refinements and countermeasures.** An ADTree node is refined either conjunctively or disjunctively. Refinement operators are unranked. Each ADTree node may only have one child of opposite type. These choices were made in order for ADTrees to reflect as closely as possible a description of an attack–defense scenario in natural language while keeping the number of defining symbols to a minimum.

   These choices do not limit the expressiveness of the formalism. We would obtain an equally expressive formalism by restricting ADTrees to binary refinements, by allowing nodes with multiple countermeasures or by allowing nodes that are conjunctively and disjunctively refined at the same time.

2. **ADTrees versus parse trees of ADTerms.** The ADTree corresponding to an ADTerm of the form $t = \mathrm{c}^{\mathrm{p}}(t_1, t_2)$ differs from the parse tree of t. We depict the root of the tree corresponding to t_2 as a child of the root node of the tree corresponding to t_1. In this manner we illustrate that t_2 represents a countermeasure for the scenario depicted by t_1. Such an illustration helps us to model interactions between the two players involved in an attack–defense scenario in an intuitive and understandable way.

3. **Finite trees.** We consider only finite ADTrees for the sake of simplicity. Infinite ADTrees are conceivable, for instance, to model recursive goals, such as obtaining keys to a locked box which contains the keys. Infinite ADTrees would also be a useful tool to study the limit case of evolving attack–defense scenarios, such as scenarios involving automated attacks and defenses.

4. **Ordered trees.** We define ADTrees to be ordered trees. This choice makes ADTrees suitable for the analysis of scenarios in which the order between actions is relevant. This could, for instance, be the case when temporal relations are taken into account.

5. **Trees versus directed acyclic graphs.** We use trees instead of directed acyclic graph (DAG) for simplicity of the formalism. DAGs are more expressive because they can be used to indicate dependencies between nodes. For instance, the two nodes labeled "Window" in Figure 2.1 could be replaced by a single node in order to express that they concern the same physical window or that all attacks against one window are also suitable attacks against the other window. Since such shared nodes give rise to different possible interpretations and to a more complicated semantical treatment, we leave the extension to DAGs to future research.

3

Semantics

In the previous chapter we have defined the syntax for ADTrees and ADTerms. The syntax allows us to model a scenario. However, the syntax alone does not enable us to interpret or adequately compare different models. Consider, e.g., the "Subdue Guard" subtree from Example 2.2. Recall that to subdue a guard, the guard must be outnumbered and a weapon must be used. Different security analysts may create different ADTrees when asked to create a model using the ADTree methodology. Figure 3.1 depicts two possible resulting ADTrees.

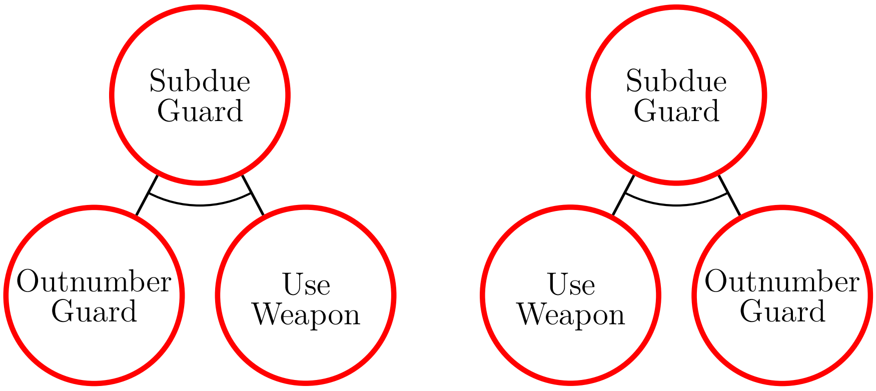


Figure 3.1: Two ADTrees for subduing a guard.

Depending on how ADTrees and ADTerms are interpreted, these two trees may or may not depict the same scenario. On the one hand, it is possible that the two trees are modeling the same situation, in which case they should be equivalent. On the other hand, it is also conceivable to understand them within a temporal setting and interpret the conjunction as having an implied order. Then, the tree in the left of Figure 3.1 could be interpreted as "Outnumber Guard" before "Use Weapon" and the tree on the right could represent a scenario where first a guard has to be threatened with a weapon, before someone else actually subdues the guard. In conclusion, the two ADTrees in Figure 3.1 should be considered as equivalent if time considerations are not factored into the model; they should be considered as not equivalent if the analyst is interested in sequential aspects of his attack.

Already this small example illustrates that the syntax needs to be equipped with one or several meanings. To be able to interpret and compare trees, we introduce semantics for ADTerms (and consequently for ADTrees). We employ the notion of equivalence classes to realize this goal. Every semantics partitions the set of all ADTerms $\mathbb{T}_\Sigma$ into equivalence classes. Then, terms that belong to the same equivalence class represent the same scenario.

Definition 3.1 (Semantics for ADTerms) A semantics for ADTerms is an equivalence relation on $\mathbb{T}_\Sigma$ that preserves types.

Different scenarios can give rise to different applications of the ADTree methodology, we propose three semantics and illustrate their differences. Semantics for ADTrees usually make use of other mathematical concepts, in which an equivalence relation is naturally defined. In Section 3.1, we introduce the propositional semantics that make use of the equivalence of certain propositional formulas. Then, in Section 3.2 we introduce the class of semantics induced by a De Morgan lattice. We show that the propositional semantics are actually an instantiation of the formally more complicated class of the semantics induced by a De Morgan lattice. We define a third semantics, the multiset semantics, in Section 3.3. The equivalence relation that is pulled back to define the multiset semantics, is the equality of sets. After defining these initial semantics in which we pull back a well-defined equivalence relation, we use equational theory to define arbitrary semantics in Section 3.4. Since the equational semantic can express the same concepts as the naturally defined semantics, we show how to define the propositional and the multiset semantics in terms of the equational semantics in Section 3.5.

The choice of an appropriate semantics becomes crucial when a quantitative analysis of an attack–defense scenario is to be performed. We discuss this issue after introducing attributes in Chapter 4.

3.1 Propositional Semantics ($\equiv_\mathcal{P}$)

Attack trees are often seen as representations of AND-OR formulas. Thus, one of the most frequently used semantics for attack trees is the propositional semantics [10KMMS, 12KMRS] and [RSF⁺09, WJ10]. In this section, we extend this particular semantics to ADTerms. When the propositional semantics is used, ADTerms are interpreted as propositional formulas. The satisfiability of the formula interpreting an ADTerm t models the feasibility of the scenario represented by t. The propositional semantics is well-suited to evaluate whether a system is vulnerable to an attack or to answer in how many different ways a system can be successfully attacked. It can also express whether or not special equipment is needed to perform an attack. More generally, it can be used to answer any recursively defined binary question related to the scenario that the ADTree models. We assign a propositional variable x_b to every basic action $b \in \mathbb{B}$. We assume that different basic actions give rise to different propositional variables. In particular, since the sets of basic actions of proponent and of opponent type are disjoint, we have

$$\{x_b \mid b \in \mathbb{B}^\mathrm{p}\} \cap \{x_d \mid d \in \mathbb{B}^\mathrm{o}\} = \emptyset.$$

We define:

Definition 3.2 (Propositional ADTerms) For every ADTerm t (Definition 2.5) a propositional formula $t_\mathcal{P}$, called a propositional ADTerm is defined as follows. Let $t^1, t^2, \ldots, t^k \in \mathbb{T}_\Sigma$, $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k \in \mathbb{N}^+$. Then we define recursively

$$\begin{aligned} b_\mathcal{P} &= x_b, \quad \text{for} \quad b \in \mathbb{B}, & (\vee_k^s(t^1, \ldots, t^k))_\mathcal{P} &= t^1_\mathcal{P} \vee \cdots \vee t^k_\mathcal{P}, \\ (\mathrm{c}^s(t^1, t^2))_\mathcal{P} &= t^1_\mathcal{P} \wedge \neg t^2_\mathcal{P}, & (\wedge_k^s(t^1, \ldots, t^k))_\mathcal{P} &= t^1_\mathcal{P} \wedge \cdots \wedge t^k_\mathcal{P}. \end{aligned}$$

In case t is an attack term (defined after Definition 2.5), the corresponding formula is called a propositional attack term.

Every assignment of Boolean values (`true` and `false`) to the propositional variables x_b, for $b \in \mathbb{B}$, which satisfies a propositional ADTerm $t_{\mathcal{P}}$, describes a way to achieve the proponent's goal represented by the ADTerm t.

Example 3.3 Consider the ADTerm $t = \vee^{\mathrm{p}}(b_1, \mathrm{c}^{\mathrm{p}}(b_2, d))$, where $b_1, b_2 \in \mathbb{B}^{\mathrm{p}}$ and $d \in \mathbb{B}^{\mathrm{o}}$. The corresponding propositional ADTerm $t_{\mathcal{P}}$ is $x_{b_1} \vee (x_{b_2} \wedge \neg x_d)$. The formula $t_{\mathcal{P}}$ is satisfied if the variable x_{b_1} is set to `true` or if the variable x_{b_2} is set to `true` while the variable x_d is set to `false`. This models the fact that, in order to achieve his goal, the proponent needs to execute the action b_1 or he needs to execute b_2 while the action d must not be executed by the opponent.

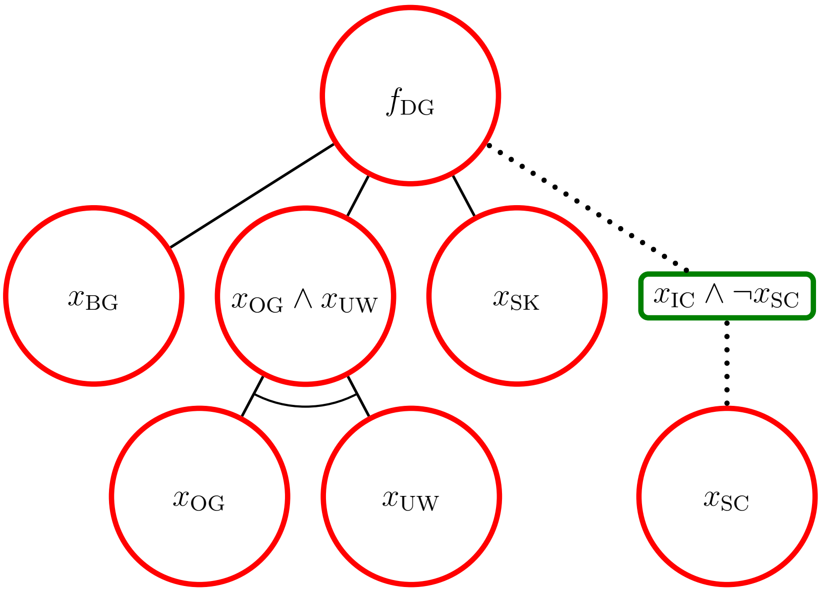


Figure 3.2: Propositional semantics for an ADTree for defeating a guard.

Let us also illustrate the propositional ADTerm with the help of the running Example 2.2. Recall the abbreviations BG, OG, UW, SK, IC and SC for the leaves of the tree "Bribe Guard", "Outnumber Guard", "Use Weapon", "Steal Key", "Install Video Cameras" and "Sabotage Cameras" and the corresponding ADTerm $t = \mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC}))$ from Example 2.6. In order to derive the propositional ADTerm corresponding to the ADTree, we identify the leaves with propositional variables, abbreviated by x_{BG}, x_{OG}, x_{UW}, x_{SK}, x_{IC} and x_{SC}. Then, we deduce the formulas for the remaining nodes by recursively applying Definition 3.2. The propositional ADTerm for the considered tree is given in Figure 3.2. The formula f_{DG} is as follows:

$$f_{\mathrm{DG}} = t_{\mathcal{P}} = (x_{\mathrm{BG}} \vee (x_{\mathrm{OG}} \wedge x_{\mathrm{UW}}) \vee x_{\mathrm{SK}}) \wedge \neg(x_{\mathrm{IC}} \wedge \neg x_{\mathrm{SC}}).$$

The formula f_{DG} expresses how to reach the main goal represented by the root node, i.e., how to defeat a guard. In order to defeat a guard, the propositional formula f_{DG} has to be `true`. To achieve this x_{BG}, x_{SK} or x_{OG} and x_{UW} are to be set to `true` (i.e., an attacker is able to bribe a guard, steal his keys or outnumber him and use a weapon). At the same time the variable x_{SK} has to be `true` or the variable x_{IC} must be `false` (i.e., either cameras are sabotaged or they are not even installed).

The procedure of providing concrete values for all variables occurring in an ADTree or ADTerm is called providing an *assignment*. In the remainder of this chapter, we analyze properties of sets of assignments while we analyze specific assignments (sets of size one) in more detail in Chapter 4.

Recall from propositional logics that two propositional formulas ψ and ψ' are equivalent (denoted by $\psi \approx \psi'$) if and only if, for every assignment ν of Boolean values to the propositional variables, we have $\nu(\psi) = \nu(\psi')$.

Definition 3.4 (Propositional semantics for ADTerms) The propositional semantics for ADTerms is the equivalence relation $\equiv_{\mathcal{P}}$ on $\mathbb{T}_\Sigma$ defined, for all $t, t' \in \mathbb{T}_\Sigma$, by

$$t \equiv_{\mathcal{P}} t' \quad \text{if and only if} \quad t_{\mathcal{P}} \approx t'_{\mathcal{P}}.$$

Note that the propositional semantics refers to equivalence classes. Since we usually specify ADTrees and ADTerms, we often abuse language and say the ADTree or ADTerm in propositional semantics when referring to the representative of the equivalence class and not the equivalence class itself. In other words, we often do not distinguish between the representative and its equivalence class.

The following example illustrates the use of the propositional semantics.

Example 3.5 Consider the ADTerm $t = \vee^{\mathrm{p}}(b_1, \mathrm{c}^{\mathrm{p}}(b_2, d))$, introduced in Example 3.3, and the ADTerm $t' = \vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(b_1, b_1), \mathrm{c}^{\mathrm{p}}(b_2, d))$. Since the propositional conjunction is idempotent, the corresponding propositional ADTerms are equivalent formulas, i.e.,

$$t_{\mathcal{P}} = x_{b_1} \vee (x_{b_2} \wedge \neg x_d) \approx (x_{b_1} \wedge x_{b_1}) \vee (x_{b_2} \wedge \neg x_d) = t'_{\mathcal{P}}.$$

Therefore, we have $t \equiv_{\mathcal{P}} t'$, i.e., the ADTerms are equivalent in the propositional semantics.

Similarly, we can express the propositional ADTerm from Example 3.3 in disjunctive normal form:

$$f_{\mathrm{DG}} = (x_{\mathrm{BG}} \wedge x_{\mathrm{SC}}) \vee (x_{\mathrm{BG}} \wedge \neg x_{\mathrm{IC}}) \vee (x_{\mathrm{SK}} \wedge x_{\mathrm{SC}}) \vee (x_{\mathrm{SK}} \wedge \neg x_{\mathrm{IC}}) \vee \\ (x_{\mathrm{OG}} \wedge x_{\mathrm{UW}} \wedge x_{\mathrm{SC}}) \vee (x_{\mathrm{OG}} \wedge x_{\mathrm{UW}} \wedge \neg x_{\mathrm{IC}})$$

It immediately shows us the six possible ways of attacking described in the example.

With the detour over ADTerms, the previous examples also show how to verify whether two ADTrees are equivalent in the propositional semantics. If they are, we have found two propositionally equivalent representations for our scenario.

We next introduce Boolean functions by closely following [PK11]. Boolean functions are closely related to propositional formulas. We use them to express the propositional semantics in an alternative way. Then, in the next section, we generalize Boolean functions to valuations, which in turn allows us to introduce the De Morgan semantics.

Definition 3.6 (Configuration) Let R be a countable set of variables. A *configuration* with finite domain $D \subseteq R$ is a function $\mathbf{x}\colon D \to \{0,1\}$ that associates a value $\mathbf{x}(x) \in \{0,1\}$ to every variable $x \in D$.

Thus, a configuration $\mathbf{x} \in \{0,1\}^D$ represents an assignment of Boolean values to the variables in D.

Definition 3.7 (Boolean functions) A Boolean function f with domain D is a function $f\colon \{0,1\}^D \to \{0,1\}$ that assigns a value $f(\mathbf{x}) \in \{0,1\}$ to each configuration $\mathbf{x} \in \{0,1\}^D$.

For $D = \{x\}$, we use the function f that satisfies $f(\mathbf{x}) = v$ for $v \in \{0,1\}$ whenever $\mathbf{x}(x) = v$ to denote the Boolean indicator function associated to the propositional variable x.

Remark 3.8 We usually drop the configuration from the assignment of the Boolean function. In other words, we typically write $x_i = v$ or even v to denote the constant configuration evaluating to v. The indicator function f_i with domain $\{0,1\}^{\{x_i\}}$ that satisfies $f_i(\mathbf{x}) = v$ whenever $\mathbf{x}(x_i) = v$ is then defined by the equation $f_i(x_i = v) = v$, or even $f_i(v) = v$.

Observe that in the context of Boolean functions, we denote the Boolean values by 0 and 1, instead of denoting them by `false` and `true`, respectively. The latter notation, we only use to denote values of propositional formula, or values related to attributes as introduced in Definition 4.2.

Given a configuration $\mathbf{x}$ with domain $D \subseteq R$, we denote by $\mathbf{x}^{\downarrow U}$ the *projection* of $\mathbf{x}$ to a subset $U \subseteq D$. This notation allows us to introduce the following two definitions.

Definition 3.9 (Conjunction and disjunction of Boolean functions) Let f and g be two Boolean functions with domains D and U, respectively. The conjunction $(f \wedge g)$ and the disjunction $(f \vee g)$ of f and g are Boolean functions with domain $D \cup U$, defined for every $\mathbf{x} \in \{0,1\}^{D \cup U}$ by

$$(f \vee g)(\mathbf{x}) = \max\{f(\mathbf{x}^{\downarrow D}), g(\mathbf{x}^{\downarrow U})\}, \quad \text{and} \quad (f \wedge g)(\mathbf{x}) = \min\{f(\mathbf{x}^{\downarrow D}), g(\mathbf{x}^{\downarrow U})\}.$$

The negation of f, denoted by $\neg f$, is a Boolean function with domain D, defined for every $\mathbf{x} \in \{0,1\}^D$ by $(\neg f)(\mathbf{x}) = 1 - f(\mathbf{x})$.

Definition 3.10 (Equivalent Boolean functions) Two Boolean functions f and g, with respective finite domains D and U, are said to be equivalent (denoted by $f \equiv g$) if and only if, for every $\mathbf{x} \in \{0,1\}^{D \cup U}$, we have $f(\mathbf{x}^{\downarrow D}) = g(\mathbf{x}^{\downarrow U})$.

Using an equivalence relation with a finite set of variables, as in Definition 3.10, makes the above construction accessible and implementable in a computer program. The significance of the previous definition is illustrated by the following example.

Example 3.11 Let f_1 and f_2 be two Boolean indicator functions with the domains $\{x_1\}$ and $\{x_2\}$, respectively. Suppose we define two new Boolean functions g_1 and g_2 as follows:

$$g_1 = f_1 \qquad g_2 = f_1 \vee (f_2 \vee \neg f_2).$$

Then $g_1 \equiv g_2$ because $g_1(\mathbf{x}^{\downarrow\{x_1\}}) = g_2(\mathbf{x}^{\downarrow\{x_1,x_2\}})$. Indeed,

$$g_1(x_1 = 1) = g_2(x_1 = 1, x_2 = 1) = g_2(x_1 = 1, x_2 = 0) = 1,$$
$$g_1(x_1 = 0) = g_2(x_1 = 0, x_2 = 1) = g_2(x_1 = 0, x_2 = 0) = 0.$$

Remark 3.12 Since logically equivalent propositional formulas represent the same Boolean functions, two ADTerms t and t' are equivalent under the propositional semantics if they represent the same Boolean function.

Using this new notation, we can rewrite Example 3.5 in terms of Definition 3.10. For $i \in \{b_1, b_2, d\}$ let f_i be the indicator function of x_i, i.e., the function with domain $\{0,1\}^{\{x_i\}}$ that satisfies $f_i(\mathbf{x}) = v$ whenever $\mathbf{x}(x_i) = v$. Then,

$$t_{\mathcal{P}} = f_{b_1} \vee (f_{b_2} \wedge \neg f_d) = (f_{b_1} \wedge f_{b_1}) \vee (f_{b_2} \wedge \neg f_d) = t'_{\mathcal{P}}.$$

In other words, the set of propositional ADTerms can be seen as a representation language for certain Boolean functions. We use this representation language to compare the computational complexity of propositional ADTerms and attack terms in Section 6.1.3.

3.2 Semantics Induced by a De Morgan Lattice

In the propositional semantics, ADTerms are interpreted as propositional formulas (or equivalently as Boolean functions). Such an interpretation limits the usefulness of the semantics to those applications which take only Boolean properties into account. Examples of such properties are satisfiability or presence of an attack. These examples imply that the propositional semantics is not well-suited to reason about properties, such as effectiveness or usefulness of the components of an attack, which may have more than two states (or values). In order to overcome this limitation of the propositional semantics, we introduce the (more complicated) semantics induced by De Morgan lattices. They are an extension of the propositional semantics and consequently cover them. In a semantics induced by a De Morgan lattice, ADTerms are interpreted as functions whose range is a De Morgan lattice.

Let $\langle A, +, \times\rangle$ be an algebraic structure defined over a non-empty set A with two binary operations $+$ and $\times$. The structure $\langle A, +, \times\rangle$ is called a *distributive lattice* if the operators $+$ and $\times$ are associative and commutative and if the following laws hold: $a \times (a+b) = a$, $a + (a \times b) = a$ (absorption) and $a \times (b+c) = (a \times b) + (a \times c)$ (distributivity of $\times$ over $+$), for $a, b, c \in A$. It is a basic fact in lattice theory that the last condition is equivalent to its dual, i.e., $a + (b \times c) = (a+b) \times (a+c)$ [DP90]. Furthermore, it is well-known [Grä03] that if $\langle A, +, \times\rangle$ is a lattice it can always be equipped with a canonical partial order $\preceq$, defined for all $a, b \in A$, by

$$a \preceq b \quad \text{if and only if} \quad a + b = b. \tag{3.1}$$

This order is monotone with respect to the operations $+$ and $\times$, see [DP90].

To introduce the notion of a De Morgan lattice, we extend the notion of a distributive lattice $\langle A, +, \times\rangle$ by a unary operation, denoted by $\neg$, satisfying De Morgan's laws and double negation.

Definition 3.13 (De Morgan lattice) An algebraic structure $\langle A, +, \times, \neg \rangle$ is called a De Morgan lattice if the substructure $\langle A, +, \times \rangle$ is a distributive lattice and, for all $a, b \in A$, we have

$$\neg(a + b) = (\neg a) \times (\neg b), \qquad \neg(a \times b) = (\neg a) + (\neg b), \qquad \neg(\neg a) = a.$$

We assume that every De Morgan lattice $\langle A, +, \times, \neg \rangle$ contains the neutral elements $\mathbf{0}$ for $+$ and $\mathbf{1}$ for $\times$. This is not a restriction since, using a result of Pouly [Pou08], it can be easily shown that $\langle A, +, \times, \neg \rangle$ can always be adjoined with such elements.

Example 3.14 The algebraic structure $\langle \{0, 1\}, \vee, \wedge, \neg \rangle$ is an example of a De Morgan lattice. This particular algebra is known as the propositional Boolean algebra.

Not every De Morgan lattice is a Boolean algebra. In fact, De Morgan lattices are more general than Boolean algebras, and are, therefore, well-suited to represent outcomes requiring more than just the two Boolean values. To increase readability, values of De Morgan lattices are typeset in typewriter font.

Example 3.15 The tuple $\langle S = \{\mathtt{I}, \mathtt{P}, \mathtt{C}\}, \max, \min, \neg \rangle$, where $\mathtt{I} \leq \mathtt{P} \leq \mathtt{C}$ and $\neg\mathtt{I} = \mathtt{C}$, $\neg\mathtt{P} = \mathtt{P}$ and $\neg\mathtt{C} = \mathtt{I}$ is a De Morgan lattice[1]. Using the values I, P and C, allows us to distinguish between `Ineffective` actions (I), `Partially` effective (P) and `Completely` effective (C), respectively. Note that I and C are neutral elements for max and min, respectively. However, this De Morgan lattice is not a Boolean algebra with I representing 0 and C representing 1. It does not satisfy the laws of complements, i.e., for all elements in $s \in S$, the maximum of s and its complement is the largest element of S and the minimum of s and its complement is the smallest element. In fact, because $\max\{\mathtt{P}, \neg\mathtt{P}\} = \mathtt{P} \neq \mathtt{C}$ and $\min\{\mathtt{P}, \neg\mathtt{P}\} = \mathtt{P} \neq \mathtt{I}$.

We now introduce De Morgan valuations which are functions represented by ADTerms when a semantics induced by a De Morgan lattice is used. As in the case of the propositional semantics, we assign a propositional variable x_b to every action $b \in \mathbb{B}$.

Definition 3.16 (De Morgan valuation) Let $\langle A, +, \times, \neg \rangle$ be a De Morgan lattice and let $D \subseteq \{x_b \mid b \in \mathbb{B}\}$ be a set of propositional variables. A De Morgan valuation f with domain D is a function $f\colon \{0, 1\}^D \to A$ that assigns a value $f(\mathbf{x}) \in A$ to every configuration $\mathbf{x} \in \{0, 1\}^D$.

Example 3.17 In case the De Morgan lattice in Definition 3.16 is the propositional Boolean algebra $\langle \{0, 1\}, \vee, \wedge, \neg \rangle$, the De Morgan valuations are simply Boolean functions. This means they are functions of the form $f\colon \{0, 1\}^D \to \{0, 1\}$, as defined in Definition 3.7.

Like in the case for the conjunction and disjunction of Boolean functions, we use the projection to define the sum and the product of De Morgan valuations.

[1] It can be verified that the order $\leq$ coincides with the canonical order given by Equivalence (3.1) when $+$ is instantiated as max.

Definition 3.18 (Sum, product and negation of De Morgan valuations) Let the tuple $\langle A, +, \times, \neg\rangle$ be a De Morgan lattice and let f and g be two De Morgan valuations with domains D and U, respectively. The sum of f and g (denoted by $f + g$) and the product of f and g (denoted by $f \times g$) are De Morgan valuations with domain $D \cup U$, defined for every $\mathbf{x} \in \{0, 1\}^{D \cup U}$ by

$$(f + g)(\mathbf{x}) = f(\mathbf{x}^{\downarrow D}) + g(\mathbf{x}^{\downarrow U}) \quad \text{and} \quad (f \times g)(\mathbf{x}) = f(\mathbf{x}^{\downarrow D}) \times g(\mathbf{x}^{\downarrow U}).$$

The negation of the De Morgan valuation f (denoted by $\neg f$) is the De Morgan valuation with domain D, defined for every $\mathbf{x} \in \{0, 1\}^D$ by $(\neg f)(\mathbf{x}) = \neg(f(\mathbf{x}))$.

Example 3.19 Consider the De Morgan lattice $\langle\{\mathtt{I}, \mathtt{P}, \mathtt{C}\}, \max, \min, \neg\rangle$ introduced in Example 3.15. Let $f\colon \{0, 1\}^{\{y\}} \to \{\mathtt{I}, \mathtt{P}, \mathtt{C}\}$ and $g\colon \{0, 1\}^{\{z\}} \to \{\mathtt{I}, \mathtt{P}, \mathtt{C}\}$ be two De Morgan valuations, given by

$$\begin{aligned} f(y = 0) &= \mathtt{I}, & \qquad g(z = 0) &= \mathtt{I}, \\ f(y = 1) &= \mathtt{P}, & \qquad g(z = 1) &= \mathtt{C}. \end{aligned}$$

Negations of f and g are defined as:

$$\begin{aligned} \neg f(y = 0) &= \mathtt{C}, & \qquad \neg g(z = 0) &= \mathtt{C}, \\ \neg f(y = 1) &= \mathtt{P}, & \qquad \neg g(z = 1) &= \mathtt{I}. \end{aligned}$$

Table 3.1 illustrates the sum of f and g as well as their product. Note that, in this case + is equal to max and × is equal to min.

y	z	$f + g = \max\{f, g\}$	$f \times g = \min\{f, g\}$
0	0	I	I
0	1	C	I
1	0	P	I
1	1	C	P

Table 3.1: The sum and product of two De Morgan valuations.

Definition 3.20 (De Morgan ADTerms) Let $\langle A, +, \times, \neg\rangle$ be a De Morgan lattice and let R be a countable set of variables. For every ADTerm t, a De Morgan valuation $t_{\mathcal{DM}} = f_t$, called a De Morgan ADTerm, is defined as follows. First, for every basic action $b \in \mathbb{B}$ we construct a propositional variable $x_b \in R$. We assume that for $b_1, b_2 \in \mathbb{B}$, $b_1 \neq b_2 \implies x_{b_1} \neq x_{b_2}$. Next, a De Morgan valuation is associated with every ADTerm t, as follows. For $t^1, t^2, \ldots, t^k \in \mathbb{T}_\Sigma$, $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k \in \mathbb{N}^+$, we define recursively

$$\begin{aligned} b_{\mathcal{DM}} &= f_b, \quad \text{for} \quad b \in \mathbb{B}, & \qquad (\vee_k^s(t^1, \ldots, t^k))_{\mathcal{DM}} &= t^1_{\mathcal{DM}} + \cdots + t^k_{\mathcal{DM}}, \\ (\mathrm{c}^s(t^1, t^2))_{\mathcal{DM}} &= t^1_{\mathcal{DM}} \times \neg t^2_{\mathcal{DM}}, & \qquad (\wedge_k^s(t^1, \ldots, t^k))_{\mathcal{DM}} &= t^1_{\mathcal{DM}} \times \cdots \times t^k_{\mathcal{DM}}, \end{aligned}$$

where f_b is a De Morgan valuation of the form $f_b\colon \{0, 1\}^{\{x_b\}} \to A$. In case t is a pure attack term, the corresponding formula is called a De Morgan attack term.

If b is a basic action, then the De Morgan valuation with domain $\{x_b\}$ expresses how the value assigned to the action b changes, depending on whether this action is present ($x_b = 1$) or absent ($x_b = 0$). With the help of basic De Morgan valuations f_b, we then associated general De Morgan valuations with composed ADTerms in a recursive way. They are well-defined by associativity of $+$ and $\times$.

A De Morgan ADTerm by itself is not sufficient to uniquely define a semantics. The same De Morgan lattice may induce several semantics. In fact, each semantics induced by a De Morgan lattice is fully determined by a De Morgan lattice $\langle A, +, \times, \neg\rangle$ and a given set of De Morgan valuations $\{f_b \colon \{0,1\}^{\{x_b\}} \to A \mid b \in \mathbb{B}\}$. Modification of a De Morgan valuation f_b results in a different semantics induced by the lattice $\langle A, +, \times, \neg\rangle$. Note that this is not the case in the propositional semantics, where f_b is canonically assumed to be the indicator function.

Recall that the purpose of a semantics for ADTerms is to define which ADTerms are equivalent. This is achieved with the help of equivalent De Morgan valuations. Consider a De Morgan lattice $\langle A, +, \times, \neg\rangle$ and two subsets of propositional variables $D, U \subseteq \{x_b \mid b \in \mathbb{B}\}$. Two De Morgan valuations f and g, with respective domains D and U, are said to be *equivalent* (denoted by $f \equiv g$) if and only if, for every $\mathbf{x} \in \{0,1\}^{D\cup U}$, we have $f(\mathbf{x}^{\downarrow D}) = g(\mathbf{x}^{\downarrow U})$.

Definition 3.21 (Semantics for ADTerms induced by a De Morgan lattice) The semantics for ADTerms induced by a De Morgan lattice $\langle A, +, \times, \neg\rangle$ and a set of De Morgan valuations $\{f_b \colon \{0,1\}^{\{x_b\}} \to A \mid b \in \mathbb{B}\}$ is the equivalence relation $\equiv_{\mathcal{DM}}$ on $\mathbb{T}_\Sigma$ defined, for all $t, t' \in \mathbb{T}_\Sigma$, by

$$t \equiv_{\mathcal{DM}} t' \quad \text{if and only if} \quad t_{\mathcal{DM}} = t'_{\mathcal{DM}}.$$

Note that since every Boolean algebra satisfies the properties of a De Morgan lattice, the propositional semantics introduced in Section 3.1 is also a semantics induced by a De Morgan lattice:

Remark 3.22 The propositional semantics for ADTerms as introduced in Definition 3.4 can be described as the semantics induced by instantiating A as the Boolean algebra $\langle\{0,1\}, \vee, \wedge, \neg\rangle$, where a basic action $b \in \mathbb{B}$ is associated with the Boolean indicator function $f_b \colon \{0,1\}^{\{x_b\}} \to \{0,1\}$ (see Remark 3.8). Note that the lattice addition and multiplication can be written as $\vee$ and $\wedge$ in case of Boolean functions. We apply Definition 3.21 for all $t, t' \in \mathbb{T}_\Sigma$,

$$t \equiv_{\mathcal{P}} t' \quad \text{if and only if} \quad f_t = f_{t'},$$

to obtain an alternative definition for the propositional semantics.

We end this section with a discussion that shows how a semantics induced by a De Morgan lattice that is not the Boolean algebra $\langle\{0,1\}, \vee, \wedge, \neg\rangle$ extends the expressive capabilities of the propositional semantics. To accomplish this, we make use of Example 3.19.

First recall that basic actions in the propositional semantics are Boolean functions of the form $f_b(x_b = v) = v$ for $v \in \{0,1\}$ and $b \in \mathbb{B}$. Such an interpretation does not allow us to differentiate between the execution of an action and its effectiveness. In other words, the propositional semantics assumes that actions which are executed are always completely effective. However, this is rarely the case in a real-life scenario. For instance, the execution of a dictionary attack to guess a password does not guarantee that the password will be found. The following example illustrates how to more accurately model such an attack by using a semantics induced by the De Morgan lattice $\langle\{\mathtt{I},\mathtt{P},\mathtt{C}\},\max,\min,\neg\rangle$.

Example 3.23 Let us consider the De Morgan lattice introduced in Example 3.15 and let $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d_1, d_2))$ be an ADTerm. We use De Morgan valuations to describe efficiency levels of the actions b, d_1 and d_2. We assume that when the actions b, d_1 and d_2 are not executed ($x_i = 0$ for $i \in \{b, d_1, d_2\}$), they are `Ineffective` (`I`). We get

$$f_b(x_b = 0) = \mathtt{I}, \qquad f_{d_1}(x_{d_1} = 0) = \mathtt{I}, \qquad f_{d_2}(x_{d_2} = 0) = \mathtt{I}.$$

Moreover, executing the actions b and d_2 ($x_i = 1$, $i \in \{b, d_2\}$) ensures their `Complete` effectiveness (`C`), but executing the action d_1 guarantees its `Partial` effectiveness only (`P`). We get

$$f_b(x_b = 1) = \mathtt{C}, \qquad f_{d_1}(x_{d_1} = 1) = \mathtt{P}, \qquad f_{d_2}(x_{d_2} = 1) = \mathtt{C}.$$

The De Morgan valuation associated with t is given by

$$f_t(x_b, x_{d_1}, x_{d_2}) = \min\{f_b(x_b), \neg(\min\{f_{d_1}(x_{d_1}), f_{d_2}(x_{d_2})\})\}.$$

This valuation allows us to reason about the effectiveness of the scenario represented by t. We have

$$\begin{array}{llll} f_t(0,0,0) = \mathtt{I}, & f_t(0,1,0) = \mathtt{I}, & f_t(1,0,0) = \mathtt{C}, & f_t(1,1,0) = \mathtt{C}, \\ f_t(0,0,1) = \mathtt{I}, & f_t(0,1,1) = \mathtt{I}, & f_t(1,0,1) = \mathtt{C}, & f_t(1,1,1) = \mathtt{P}. \end{array}$$

Since the first entry of all elements in $f_t^{-1}(\{\mathtt{P},\mathtt{C}\})$ is 1, we deduce that the scenario is at least partially effective for the proponent if the action b is executed, independent of the opponent's actions d_1 and d_2.

3.3 Multiset Semantics ($\equiv_{\mathcal{M}}$)

In the two semantics considered so far, the refining symbols $\vee^s$ and $\wedge^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, have been interpreted with idempotent operators. Therefore, both semantics assume that the multiplicity of a subgoal is irrelevant. This assumption, however, may not be desired in all applications of ADTrees. The number of times that an action needs to be executed, for example, is relevant when considering that every repeated action takes additional time.

Example 3.24 Consider the scenario illustrated in Figure 2.1. In order to deprive the attacker of the possibilities to break in through the back door and to break in through the main door, the defender has to install locks on both doors. Since the two doors are in two physically distinct locations, reusing the same lock is not possible in this case.

Naturally, instead of adapting the semantics of the ADTree, it would also be possible to provide two distinct labels for each lock which is to be installed, e.g., “Install Lock 1” and “Install Lock 2”. However, introducing new node labels has the side-effect that the model is more specific and a possible reuse of the ADTree is less likely.

We, therefore, introduce the multiset semantics. It allows us to distinguish between multiple occurrences of the same action. Thus, it is suitable for analyzing scenarios in which such multiple occurrences of the same subgoal are significant, as in Example 3.24. The multiset semantics has initially been defined for attack trees in [MO05]. Our construction extends this framework to ADTrees.

Given a set H, we use $\mathcal{P}(H)$ to denote the powerset of H and $\mathcal{M}(H)$ to denote the collection of all multisets of elements in H. We use $\{\!|a_1,\ldots,a_n|\!\}$ to denote a multiset composed of (not necessarily distinct) elements $a_1,\ldots,a_n$. The symbol $\uplus$ denotes the multiset union.

In the multiset semantics, the ADTerms are interpreted as a set of pairs of the form $(P,O) \in \mathcal{M}(\mathbb{B}^{\mathrm{p}}) \times \mathcal{M}(\mathbb{B}^{\mathrm{o}})$ which are called *bundles*. A bundle (P,O) encodes how the proponent can achieve his goal: the proponent must perform all actions present in P while the opponent must not perform any of the actions in O. The set of bundles corresponding to an ADTerm t is an element of $\mathcal{P}(\mathcal{M}(\mathbb{B}^{\mathrm{p}}) \times \mathcal{M}(\mathbb{B}^{\mathrm{o}}))$, denoted by $t_{\mathcal{M}}$. It represents alternative possibilities for the proponent to achieve his goal. A basic action b of proponent type is interpreted as a singleton $b_{\mathcal{M}} = \{(\{\!|b|\!\},\emptyset)\}$ because in order to achieve his goal it is sufficient for the proponent to execute action b. A basic action b of opponent type is interpreted as $b_{\mathcal{M}} = \{(\emptyset,\{\!|b|\!\})\}$ because in order for the proponent to be successful, the action b must not be executed by the opponent. In order to obtain the multiset interpretation of composed ADTerms, we use the union of sets of bundles ($\cup$) and the *distributive product* of sets of bundles ($\otimes$). The distributive product of two sets of bundles S and Z is defined as the set of bundles

$$S \otimes Z = \{(P_S \uplus P_Z, O_S \uplus O_Z) \mid (P_S,O_S) \in S \quad \text{and} \quad (P_Z,O_Z) \in Z\}.$$

The distributive product can be extended to any finite number of sets of bundles. The multiset interpretation $t_{\mathcal{M}}$ of a composed ADTerm t is then given by

$$\begin{aligned}
&(\vee^{\mathrm{p}}_k(t^1,\ldots,t^k))_{\mathcal{M}} = t^1_{\mathcal{M}} \cup \cdots \cup t^k_{\mathcal{M}}, && (\vee^{\mathrm{o}}_k(t^1,\ldots,t^k))_{\mathcal{M}} = t^1_{\mathcal{M}} \otimes \cdots \otimes t^k_{\mathcal{M}},\\
&(\wedge^{\mathrm{p}}_k(t^1,\ldots,t^k))_{\mathcal{M}} = t^1_{\mathcal{M}} \otimes \cdots \otimes t^k_{\mathcal{M}}, && (\wedge^{\mathrm{o}}_k(t^1,\ldots,t^k))_{\mathcal{M}} = t^1_{\mathcal{M}} \cup \cdots \cup t^k_{\mathcal{M}},\\
&(\mathrm{c}^{\mathrm{p}}(t^1,t^2))_{\mathcal{M}} = t^1_{\mathcal{M}} \otimes t^2_{\mathcal{M}}, && (\mathrm{c}^{\mathrm{o}}(t^1,t^2))_{\mathcal{M}} = t^1_{\mathcal{M}} \cup t^2_{\mathcal{M}}.
\end{aligned}$$

Let t be an ADTerm and let t' be one of its subterms. Note that the set of bundles $t'_{\mathcal{M}}$ encodes how the proponent of the term t can be successful in the situation described by subterm t', regardless of the type of t'. In particular, in order to achieve a disjunctive goal, the proponent has to achieve *at least one* of the corresponding subgoals. Similarly, in order to successfully prevent a conjunctive countermeasure of the opponent, it is sufficient for the proponent to prevent *at least one* of the corresponding subcountermeasures. An analogous reasoning holds for a goal of the proponent which is conjunctively refined and a disjunctively refined countermeasure of the opponent. This is the reason why the operator used to define the multiset interpretation of a disjunctively refined goal for one player is the same as the operator used to define the multiset interpretation of a conjunctively refined goal for the other player.

Definition 3.25 (Multiset semantics for ADTerms) The multiset semantics for ADTerms is the equivalence relation on $\mathbb{T}_\Sigma$, denoted by $\equiv_\mathcal{M}$ and defined for all $t, t' \in \mathbb{T}_\Sigma$ by

$$t \equiv_\mathcal{M} t' \quad \text{if and only if} \quad t_\mathcal{M} = t'_\mathcal{M}.$$

The following example shows that the multiset semantics takes into account multiple occurrences of the same actions.

Example 3.26 The ADTerms $t = \mathrm{c}^\mathrm{p}(b, \wedge^\mathrm{o}(d_1, d_2))$ and $t' = \mathrm{c}^\mathrm{p}(\wedge^\mathrm{p}(b, b), \wedge^\mathrm{o}(d_1, d_2))$ from Example 3.5 have been shown to be equivalent with respect to the propositional semantics. Their multiset interpretations are $t_\mathcal{M} = \{(\{\!|b|\!\}, \{\!|d_1|\!\}), (\{\!|b|\!\}, \{\!|d_2|\!\})\}$ and $t'_\mathcal{M} = \{(\{\!|b, b|\!\}, \{\!|d_1|\!\}), (\{\!|b, b|\!\}, \{\!|d_2|\!\})\}$. Since $t_\mathcal{M} \neq t'_\mathcal{M}$, the ADTerms t and t' are not equivalent with respect to the multiset semantics.

By comparing Examples 3.5 and 3.26, we deduce that the partition of $\mathbb{T}_\Sigma$ defined by the multiset semantics does not coincide with the partition defined by the propositional semantics. A more detailed comparison of these two semantics is presented in Section 3.5.1.

3.4 Equational Semantics

As discussed in previous sections, the choice of an appropriate semantics depends on the aspects a security analyst wants to analyze. Such aspects can frequently be modeled with the help of mathematical properties. For example, if for a modeled aspect, the order in which the subgoals of conjunctively refined goals are executed is irrelevant, the conjunctive refinement should be modeled using a commutative operator. Similarly, if the examined aspect does not distinguish between executing the same action twice or only once, the corresponding operator should be idempotent. In this section, we show how to construct a semantics for ADTerms which takes a given set of properties into account. The idea is to specify an equivalence relation on ADTerms through a set of equations expressing the desired properties. This approach is a generalization of a concept described by Mauw and Oostdijk in [MO05], which uses a specific set of rewrite rules to encode allowed tree transformations. Our framework is more general since we allow any set of equations to define an equivalence relation on ADTerms.

Let $\mathrm{VAR} = \mathrm{VAR}^\mathrm{p} \cup \mathrm{VAR}^\mathrm{o}$ be a set of typed variables. We use capital letters such as X, X_i, Y and Y_i, to denote elements of VAR. We extend the set $\mathbb{T}_\Sigma$ to the set $\mathbb{T}_\Sigma^\mathrm{VAR}$ of typed ADTerms over the variables in VAR. An *equation* is a pair $(t, t') \in \mathbb{T}_\Sigma^\mathrm{VAR} \times \mathbb{T}_\Sigma^\mathrm{VAR}$, where t and t' have the same type. We denote the equation (t, t') by $t = t'$. An *algebraic specification* for ADTerms is a pair (Σ, E), where Σ is the AD–signature and E is a set of equations. Given an algebraic specification (Σ, E), we define the set of syntactic consequences of E as the smallest subset of $\mathbb{T}_\Sigma^\mathrm{VAR} \times \mathbb{T}_\Sigma^\mathrm{VAR}$ containing E and being closed under reflexivity, symmetry, transitivity, substitutions and contexts. In other words, the equation $t = t'$ is a syntactic consequence of E (denoted by $E \vdash (t = t')$) if it can be derived from E by using the following rules:

- If $(t = t') \in E$, then $E \vdash (t = t')$.
- For every $t \in \mathbb{T}_{\Sigma}^{\mathrm{VAR}}$, $E \vdash (t = t)$.
- If $E \vdash (t = t')$, then $E \vdash (t' = t)$.
- If $E \vdash (t = t')$ and $E \vdash (t' = t'')$, then $E \vdash (t = t'')$.
- If $\rho\colon \mathrm{VAR} \to \mathbb{T}_{\Sigma}^{\mathrm{VAR}}$ is a substitution and $E \vdash (t = t')$, then $E \vdash (\rho(t) = \rho(t'))$.
- If $E \vdash (t = t')$ and $C[\,]$ is a context (i.e., a term with a hole of the same type as t), then $E \vdash (C[t] = C[t'])$.

In the following definition we introduce the notion of equational semantics for ADTerms.

Definition 3.27 (Equational semantics for ADTerms) The equational semantics for ADTerms induced by an algebraic specification (Σ, E) is the equivalence relation $\equiv_E$ on $\mathbb{T}_\Sigma$, defined by

$$t \equiv_E t' \quad \text{if and only if} \quad E \vdash (t = t').$$

With the next example we show how to use the equational semantics.

Example 3.28 Let Sym_k denote the set of all bijections from $\{1, \ldots, k\}$ to itself. Consider the equational semantics induced by an algebraic specification (Σ, E_k), where

$$E_k = \{\vee^{\mathrm{p}}(X_1, \ldots, X_k) = \vee^{\mathrm{p}}(X_{\sigma(1)}, \ldots, X_{\sigma(k)}) \mid \sigma \in \mathrm{Sym}_k\}.$$

The equations in E_k encode the commutativity of the disjunctive operator for the proponent. Thus, for the two ADTerms $t_1 = \vee^{\mathrm{p}}(a, b)$ and $t_2 = \vee^{\mathrm{p}}(b, a)$, we have $t_1 \equiv_{E_k} t_2$ for $k > 1$, i.e., t_1 and t_2 model the same situation when the semantics $\equiv_{E_k}$ is used and $k > 1$. In contrast, $t_1' = \wedge^{\mathrm{p}}(a, b) \not\equiv_{E_k} t_2' = \wedge^{\mathrm{p}}(b, a)$ because none of the equations in E_k allows us to transform the operator $\wedge^{\mathrm{p}}$.

The importance of defining a semantics, given a set of equations, is twofold. First, equations allow us to encode many of the mathematical properties desired for analysis of ADTrees. Second, the equations in E model all possible transformations of ADTerms, which preserve the semantics $\equiv_E$. We perform further analysis of the equational semantics in Chapter 4 where we combine equational semantics with attributes. In the next section, we use the concept of equational semantics to axiomatize the propositional and the multiset semantics for ADTerms.

3.5 Axiomatization of Semantics for ADTerms

In this section, we introduce the notion of a complete set of axioms for semantics for ADTerms. Then we show how we can use complete sets of axioms to compare partitions of the set of all ADTerms corresponding to two different semantics. Finally, we provide complete sets of axioms for the propositional and the multiset semantics and prove soundness and completeness of both sets.

We use two different proof strategies to prove completeness of the axiom sets for the propositional semantics and the multiset semantics. For the multiset semantics, we employ a standard proof strategy by transforming the equations into a rewriting system and showing its strong termination as well as confluence. The proof mainly argues with ADTerms before relating ADTerms to the multiset semantics. The proof is constructive in that it can easily be turned into an algorithm that assigns a unique representative to every equivalence class defined by the multiset semantics. We did not succeed in proving completeness of the set of axioms for the propositional semantics by following the same strategy. Instead, we first relate ADTerms to the propositional semantics and then argue using propositional logic. This proof strategy helps us to explain that the propositional ADTerms correspond to a subset of all propositional formulas. There are two reasons that the latter proof strategy is not favorable in the case of the multiset semantics. First, the multiset semantics has a more intricate syntax. Second, employing the proof strategy we followed to prove the completeness of the set of axioms for the propositional semantics does not provide term rewrite rules. Having these could be advantageous when implementing the formalism in a software tool.

3.5.1 The Notion of a Complete Set of Axioms

We start by defining the notion of a complete set of axioms for a semantics for ADTerms. A complete set of axioms allows us to express which expressions are algebraically derivable. The resulting set of expressions reflects properties of the complete set of axioms.

Definition 3.29 (Complete set of axioms) Let (Σ, E) be an algebraic specification and let $\equiv$ be a semantics for ADTerms. A set E is a complete set of axioms for the semantics $\equiv$ if and only if $\equiv$ is equal to the equational semantics induced by the algebraic specification (Σ, E).

Remark 3.30 It follows directly from Definition 3.29 that E is a complete set of axioms for the equational semantics induced by an algebraic specification (Σ, E).

The importance of having a complete set of axioms for a semantics of ADTerms is manifold. First, having complete sets of axioms unifies the treatment of different semantics for ADTrees. Instead of having to argue within different domains, such as sets of multisets or propositional logics, we can reason with ADTerms over the AD–signature. Second, the equations of a complete set of axioms typically state important properties modeled by a semantics, as shown in Example 3.28. We see in Chapter 4 that this helps us to formally define how to quantitatively analyze attack–defense scenarios using attributes. Third, knowing a complete set of axioms is a crucial step in developing algorithms which assign unique representatives to every equivalence class arising from a semantics. This simplifies the development of a computer tool that supports the ADTree methodology and can facilitate tree comparisons. Finally, we can use complete sets of axioms to facilitate a comparison between different semantics. In the remainder of this section we take a closer look at this issue.

In order to decide whether properties of ADTerms interpreted using one semantics can be exploited to reason about ADTerms within a different semantics, we need to compare the corresponding partitions of the set of ADTerms. To this end, we define the notions of finer and coarser semantics. Intuitively, given two semantics, we say that one is finer than the other if it partitions the set of ADTerms in a finer way.

Definition 3.31 (Finer semantics, coarser semantics) Let $\equiv_1$ and $\equiv_2$ be two semantics for ADTerms. The semantics $\equiv_1$ is finer than the semantics $\equiv_2$ if and only if $\equiv_1 \subseteq \equiv_2$, i.e., for all $t, t' \in \mathbb{T}_\Sigma$, $t \equiv_1 t' \Rightarrow t \equiv_2 t'$. If $\equiv_1$ is finer than $\equiv_2$, we also say that $\equiv_2$ is coarser than $\equiv_1$.

The fact that ADTerms which are equivalent according to a semantics which is finer are also equivalent according to any semantics which is coarser, allows us to exploit properties of a finer semantics into any coarser semantics.

In general, given two semantics for ADTerms, it is not clear how to decide whether they are comparable and if so, which one is finer. However, this task may become as trivial as comparing sets if we are able to appropriately axiomatize both semantics using complete sets of axioms.

Theorem 3.32 *Let $\equiv_1$ and $\equiv_2$ be two semantics for ADTerms with complete sets of axioms E_1 and E_2, respectively. If $E_1 \subseteq E_2$, then $\equiv_1$ is finer than $\equiv_2$.*

Proof. An immediate consequence of $E_1 \subseteq E_2$ is that every equation derivable from E_1 is also derivable from E_2, which proves the theorem. □

In Sections 3.5.2 and 3.5.3, we construct complete sets of axioms for the propositional ($E_\mathcal{P}$) and the multiset semantics ($E_\mathcal{M}$), respectively. These sets help us to compare the two semantics. For instance, the idempotent laws hold in the propositional but not in the multiset semantics. It turns out that, in general, all axioms in the multiset semantics also hold in the propositional semantics.

Theorem 3.33 *The multiset semantics for ADTerms is finer than the propositional semantics for ADTerms.*

Proof. It is sufficient to consider the complete sets of axioms $E_\mathcal{P}$ for the propositional semantics and $E_\mathcal{M}$ for the multiset semantics, which we introduce below in Theorems 3.34 and 3.37, respectively. We observe that $E_\mathcal{M} \subseteq E_\mathcal{P}$, which according to Theorem 3.32 finishes the proof. □

Note that the propositional semantics is not finer than the multiset semantics, as shown by Examples 3.5 and 3.26. Thus, these two semantics are not equal.

As mentioned earlier, the De Morgan semantics are, in fact, a class of semantics. A semantics needs to specify the De Morgan lattice and the De Morgan valuations associated to the involved basic actions. We do not provide a complete set of axioms for other De Morgan semantics than the propositional semantics. However, using the results from the next section, we are, nevertheless, able to relate any other semantics induced by a De Morgan lattice to the propositional semantics, as shown in Theorem 3.36.

3.5.2 A Complete Set of Axioms for $\equiv_{\mathcal{P}}$

In this section, we give a complete set of axioms for the propositional semantics.

Theorem 3.34 *The following set of equations, denoted by $E_{\mathcal{P}}$, is a complete set of axioms for the propositional semantics.*[2]

$$\vee^s (X_1, \dots, X_k) = \vee^s(X_{\sigma(1)}, \dots, X_{\sigma(k)}), \textit{ for all } \sigma \in \mathrm{Sym}_k \tag{E_1^s}$$
$$\wedge^s (X_1, \dots, X_k) = \wedge^s(X_{\sigma(1)}, \dots, X_{\sigma(k)}), \textit{ for all } \sigma \in \mathrm{Sym}_k \tag{E_2^s}$$
$$\vee^s (X_1, \dots, X_k, \vee^s(Y_1, \dots, Y_n)) = \vee^s(X_1, \dots, X_k, Y_1, \dots, Y_n) \tag{E_3^s}$$
$$\wedge^s (X_1, \dots, X_k, \wedge^s(Y_1, \dots, Y_n)) = \wedge^s(X_1, \dots, X_k, Y_1, \dots, Y_n) \tag{E_4^s}$$
$$\vee^s (X) = X \tag{E_5^s}$$
$$\wedge^s (X) = X \tag{E_6^s}$$
$$\vee^s (X, \wedge^s(X, X_1, \dots, X_k)) = X \tag{E_7^s}$$
$$\wedge^s (X, \vee^s(X, X_1, \dots, X_k)) = X \tag{E_8^s}$$
$$\vee^s (X, \wedge^s(X_1, \dots, X_k)) = \wedge^s(\vee^s(X, X_1), \dots, \vee^s(X, X_k)) \tag{E_9^s}$$
$$\wedge^s (X, \vee^s(X_1, \dots, X_k)) = \vee^s(\wedge^s(X, X_1), \dots, \wedge^s(X, X_k)) \tag{E_{10}^s}$$
$$\vee^s (X, X, X_1, \dots, X_k) = \vee^s(X, X_1, \dots, X_k) \tag{E_{11}^s}$$
$$\wedge^s (X, X, X_1, \dots, X_k) = \wedge^s(X, X_1, \dots, X_k) \tag{E_{12}^s}$$
$$\mathrm{c}^s(\vee^s(X_1, \dots, X_k), X) = \vee^s(\mathrm{c}^s(X_1, X), \dots, \mathrm{c}^s(X_k, X)) \tag{E_{13}^s}$$
$$\mathrm{c}^s(\wedge^s(X_1, \dots, X_k), X) = \wedge^s(\mathrm{c}^s(X_1, X), \dots, \mathrm{c}^s(X_k, X)) \tag{E_{14}^s}$$
$$\mathrm{c}^s(X, \vee^{\overline{s}}(X_1, \dots X_k)) = \wedge^s(\mathrm{c}^s(X, X_1), \dots, \mathrm{c}^s(X, X_k)) \tag{E_{15}^s}$$
$$\mathrm{c}^s(X, \wedge^{\overline{s}}(X_1, \dots X_k)) = \vee^s(\mathrm{c}^s(X, X_1), \dots, \mathrm{c}^s(X, X_k)) \tag{E_{16}^s}$$
$$\mathrm{c}^s(\mathrm{c}^s(X, X_1), X_2) = \mathrm{c}^s(X, \vee^{\overline{s}}(X_1, X_2)) \tag{E_{17}^s}$$
$$\mathrm{c}^s(X, \mathrm{c}^{\overline{s}}(X_1, X_2)) = \vee^s(\mathrm{c}^s(X, X_1), \wedge^s(X, X_2)) \tag{E_{18}^s}$$
$$\vee^s (\mathrm{c}^s(X_1, Y), X_2, \dots, X_k) = \mathrm{c}^s(\vee^s(X_1, \dots, X_k), \mathrm{c}^{\overline{s}}(Y, \vee^s(X_2, \dots, X_k))) \tag{E_{19}^s}$$
$$\wedge^s (\mathrm{c}^s(X_1, Y), X_2, \dots, X_k) = \mathrm{c}^s(\wedge^s(X_1, \dots, X_k), Y) \tag{E_{20}^s}$$
$$\vee^s (\mathrm{c}^s(X, Y), X) = X \tag{E_{21}^s}$$
$$\wedge^s (\mathrm{c}^s(X, Y), X) = \mathrm{c}^s(X, Y), \tag{E_{22}^s}$$

where $X, Y, X_i, Y_j \in \mathrm{VAR}$, $i, j, k, n \in \mathbb{N}^+$, $s \in \{\mathrm{p}, \mathrm{o}\}$ *and* Sym_k *denotes the set of all bijections from* $\{1, \dots, k\}$ *to itself.*

Outline of the proof. In order to prove Theorem 3.34, we define the notion of a complete set of axioms *for a set of propositional formulas*. We reduce the problem of finding a complete set of axioms for the propositional semantics to the problem of finding a complete set of axioms for the set of all propositional ADTerms. The outline of the proof is as follows:

[2] We remark that the set of axioms given in Theorem 3.34 is, in fact, an axiom scheme [Pot04]. This is unavoidable because the AD–signature contains infinitely many function symbols modeled using unranked functions.

1. By reformulating equations in $E_{\mathcal{P}}$, we define a complete set G of axioms for the set of propositional ADTerms.

2. We show using axioms in G that every propositional ADTerm can be transformed into a disjunctive form.

3. We transform the obtained disjunctive forms further into minimal disjunctive forms.

4. We prove that these minimal disjunctive forms are unique modulo associativity and commutativity.

The above considerations help us to conclude that two ADTerms are equivalent with respect to the propositional semantics if and only if the minimal disjunctive forms for the corresponding propositional ADTerms are equal modulo associativity and commutativity. This finishes the proof since the axioms in G correspond exactly to the axiom schemes in $E_{\mathcal{P}}$ when each equation is transformed into the propositional semantics.

In the remainder of this section, we give details for Steps 1–4.

Step 1: We first define a grammar that generates all propositional ADTerms, which are defined in Section 3.1. Let $X^G = \{x_b \mid b \in \mathbb{B}^{\mathrm{p}}\}$ and $Y^G = \{y_b \mid b \in \mathbb{B}^{\mathrm{o}}\}$ be two sets of propositional variables that correspond to the basic actions in the propositional semantics. We have $X^G \cap Y^G = \emptyset$. Consider the propositional formulas over $X^G \cup Y^G$ generated by the following two grammars, denoted by $\mathcal{ADT}_P$ and $\mathcal{ADT}_N$. The grammars have the start symbol P and N, respectively:

$$\begin{array}{llllllll} P: & X^G & \mid & P \vee P & \mid & P \wedge P & \mid & P \wedge \neg N \\ N: & Y^G & \mid & N \vee N & \mid & N \wedge N & \mid & N \wedge \neg P. \end{array} \qquad (\mathcal{ADT})$$

We define $\mathcal{ADT}$ as union of $\mathcal{ADT}_P$ and $\mathcal{ADT}_N$ and write $\psi \in \mathcal{ADT}$ if ψ is generated by $\mathcal{ADT}$. Thus, we abuse notation and let $\mathcal{ADT}$ denote both the grammar and the set of formulas generated by the grammar. It is easy to see that $t \in \mathbb{T}_{\Sigma}^{\mathrm{p}}$ (resp. $\mathbb{T}_{\Sigma}^{\mathrm{o}}$) if and only if there exists a formula $\psi \in \mathcal{ADT}_P$ (resp. $\phi \in \mathcal{ADT}_N$), such that $t_{\mathcal{P}} = \psi$ (resp. $t_{\mathcal{P}} = \phi$) modulo associativity.

Let E be a set of equations of the form $\xi = \zeta$, where ξ and ζ are propositional formulas and let $\mathcal{E}$ be a set of propositional formulas. We say that E is a *complete set of axioms for* $\mathcal{E}$ if and only if two propositionally equivalent formulas in $\mathcal{E}$ can be transformed into each other by applying substitutions and context to equations in E. The problem of finding a complete set of axioms for the propositional semantics can be reduced to finding a complete set of axioms for the set of propositional formulas generated by $\mathcal{ADT}$.

Lemma 3.35 *Let X, Y and Z be propositional variables. The following set G, where $\psi \star \phi = \psi \wedge \neg\phi$ for propositional formulas ψ and ϕ over the variables X, Y*

and Z, is a complete set of axioms for $\mathcal{ADT}$.[3]

$$\begin{array}{ll}
X \vee Y = Y \vee X & X \wedge Y = Y \wedge X \\
X \vee (Y \vee Z) = (X \vee Y) \vee Z & X \wedge (Y \wedge Z) = (X \wedge Y) \wedge Z \\
\vee(X) = X & \wedge(X) = X \\
X \vee (X \wedge Y) = X & X \wedge (X \vee Y) = X \\
X \vee (Y \wedge Z) = (X \vee Y) \wedge (X \vee Z) & X \wedge (Y \vee Z) = (X \wedge Y) \vee (X \wedge Z) \\
X \vee X = X & X \wedge X = X \\
(X \vee Y) \star Z = (X \star Z) \vee (Y \star Z) & (X \wedge Y) \star Z = (X \star Z) \wedge (Y \star Z) \\
X \star (Y \vee Z) = (X \star Y) \wedge (X \star Z) & X \star (Y \wedge Z) = (X \star Y) \vee (X \star Z) \\
(X \star Y) \star Z = X \star (Y \vee Z) & X \star (Y \star Z) = (X \star Y) \vee (X \wedge Z) \\
(X \star Y) \vee Z = (X \vee Z) \star (Y \star Z) & (X \star Y) \wedge Z = (X \wedge Z) \star Y \\
(X \star Y) \vee X = X & (X \star Y) \wedge X = (X \star Y).
\end{array}$$

Proof. For every axiom $\xi = \zeta$ in G, we have $\xi \approx \zeta$. Therefore, if a formula ψ' is obtained from a formula ψ by using axioms in G, we have $\psi \approx \psi'$. This proves soundness, i.e., the transformations rules provide only formulas that are valid with respect to its semantics.

Proving completeness is done by showing that, using axioms in G, every formula $\psi \in \mathcal{ADT}$ can be transformed into a minimal disjunctive form, which is denoted by $\mathrm{mdf}(\psi)$. In Steps 2–4 below, we prove that this minimal disjunctive form is unique up to commutativity and associativity of $\vee$ and $\wedge$. We denote this by $=_{AC}$. In other words, we show that, for $\psi, \psi' \in \mathcal{ADT}$,

$$\psi \approx \psi' \quad \text{if and only if} \quad \mathrm{mdf}(\psi) =_{AC} \mathrm{mdf}(\psi') \tag{3.2}$$

holds.

Step 2: Note that for all $\psi \in \mathcal{ADT}_P$ and $\phi \in \mathcal{ADT}_N$, we have that $\psi \not\approx \phi$ because $X^G \cap Y^G = \emptyset$. To define minimal disjunctive forms for the formulas in $\mathcal{ADT}$, we first introduce a grammar $\mathcal{DADT}$ generating propositional formulas in disjunctive form and we show that every formula generated by $\mathcal{ADT}$ can be transformed, using axioms in G, into an equivalent formula in disjunctive form, generated by $\mathcal{DADT}$. We later use these disjunctive forms to obtain minimal forms.

Let the following grammars be denoted by $\mathcal{DADT}_P$ and $\mathcal{DADT}_N$, where the start symbols are B^P and B^N, respectively:

$$\begin{array}{rlllll}
B^P: & K^P & \mid & K^P \star D^N & \mid & B^P \vee B^P \\
K^P: & X^G & \mid & K^P \wedge K^P & & \\
D^N: & Y^G & \mid & D^N \vee D^N & & \\
& & & & & \\
B^N: & K^N & \mid & K^N \star D^P & \mid & B^N \vee B^N \\
K^N: & Y^G & \mid & K^N \wedge K^N & & \\
D^P: & X^G & \mid & D^P \vee D^P. & &
\end{array} \tag{$\mathcal{DADT}$}$$

[3] Note that contrary to the set $E_{\mathcal{P}}$, the set G is finite. The reduction of the number of equations is made possible because the unranked function symbols $\vee^s$ and $\wedge^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, are interpreted with the associative operators $\vee$ and $\wedge$.

Like before, we define the union $\mathcal{DADT} = \mathcal{DADT}_P \cup \mathcal{DADT}_N$. It is clear that every formula generated by $\mathcal{DADT}$ is also generated by $\mathcal{ADT}$. This can be seen since every generation rule with index P can be replicated by the generation rule P (in $\mathcal{ADT}_P$) and every generation rule with index N can be replicated by the generation rule N (in $\mathcal{ADT}_N$). To prove the converse, we show that, for every $\psi \in \mathcal{ADT}_P$ (resp. $\phi \in \mathcal{ADT}_N$), there exists an equivalent disjunctive formula of the form $\psi \in \mathcal{DADT}_P$ (resp. $\phi \in \mathcal{DADT}_N$). The disjunctive formula is obtained from start symbol P (resp. N) by only using axioms in G. This is proven by induction on the structure of P (resp. N). For this it suffices to show the following two statements for all $S \in \{P, N\}$.

- If $\psi_1, \psi_2 \in \mathcal{DADT}_S$, then $\psi_1 \wedge \psi_2$ can be transformed, using axioms in G, into a formula of the form $\psi \in \mathcal{DADT}_S$.
- If $\psi_1, \psi_2 \in \mathcal{DADT}_S$, then $\psi_1 \star \psi_2$ can be transformed, using axioms in G, into a formula of the form $\psi \in \mathcal{DADT}_S$.

The statements follow by structural induction.

Let $I \subseteq \mathbb{N}$ be a non-empty, finite index set. We see that every formula $\psi \in \mathcal{DADT}_P$ is in the following disjunctive form.

$$\psi = \bigvee_{k \in I} \alpha_k \star \beta_k,$$

where $\alpha_k = (x_{k,1} \wedge \cdots \wedge x_{k,u})$ and $\beta_k = (y_{k,1} \vee \cdots \vee y_{k,l})$, for some $x_{k,1}, \ldots, x_{k,u} \in X^G$, $y_{k,1}, \ldots, y_{k,l} \in Y^G$, $u \geq 1$ and $l \geq 0$. (A similar disjunctive form exists for $\phi \in \mathcal{DADT}_N$.)

Step 3: Our goal is now to minimize the obtained disjunctive forms. We show that, using axioms in G, we can transform every $\psi \in \mathcal{ADT}_P$ into an equivalent formula, denoted by $\mathrm{mdf}(P)$, which is of the form

$$\mathrm{mdf}(P) = \bigvee_{k \in I} \alpha_k \star \beta_k,$$

where $\alpha_k = (x_{k,1} \wedge \cdots \wedge x_{k,u})$, $\beta_k = (y_{k,1} \vee \cdots \vee y_{k,l})$ and for all $k, k' \in I$, $k \neq k'$ we have $\alpha_{k'} \star \beta_{k'}$ does not imply $\alpha_k \star \beta_k$ and for $i \neq j$ we have $x_{k,i} \neq x_{k,j}$ and $y_{k,i} \neq y_{k,j}$. We proceed by contraposition. We already know that, by using axioms in G, every $\psi \in \mathcal{ADT}_P$ can be transformed into the formula $\psi \in \mathcal{DADT}_P$. Assume that $\psi = \bigvee_{k \in I} \alpha_k \star \beta_k$ is not minimal. This means that either there exist $k, k' \in I$, such that $\alpha_{k'} \star \beta_{k'}$ implies $\alpha_k \star \beta_k$ or there exists $k \in I$ and $i \neq j$, such that $x_{k,i} = x_{k,j}$ or $y_{k,i} = y_{k,j}$. In the latter case, we minimize the formula with the help of the idempotence laws, i.e., $X \vee X = X$ or $X \wedge X = X$. From [11KPS], we know that every formula ψ represents a monotone Boolean function, hence, the former case may only happen if, for $\alpha_k = (x_{k,1} \wedge \cdots \wedge x_{k,u})$, $\beta_k = (y_{k,1} \vee \cdots \vee y_{k,l})$, $\alpha_{k'} = (x_{bk'_1} \wedge \cdots \wedge x_{bk'_u})$ and $\beta_{k'} = (y_{bk'_1} \vee \cdots \vee y_{bk'_l})$, it holds that $\{x_{k,1}, \ldots, x_{k,u}\} \subseteq \{x_{bk'_1}, \ldots, x_{bk'_u}\}$ and $\{y_{k,1}, \ldots, y_{k,l}\} \subseteq \{y_{bk'_1}, \ldots, y_{bk'_l}\}$. We now sketch how to minimize ψ, using axioms in G, in all possible cases. For ease of notation only exemplify the procedure and assume that $\alpha_k = x_b$, $\alpha_{k'} = x_b \wedge x_{b'}$, $\beta_k = y_b$ and $\beta_{k'} = y_b \vee y_{b'}$, unless otherwise stated.

a) If $\alpha_k \star \beta_k = \alpha_{k'} \star \beta_{k'}$, then $(\alpha_k \star \beta_k) \vee (\alpha_{k'} \star \beta_{k'})$ can be reduced to $\alpha_{k'} \star \beta_{k'}$ by using idempotence of $\vee$.

b) If $\{y_{k,1}, \dots, y_{k,l}\} \neq \emptyset$, the following scheme can be used:

$$\begin{aligned}
&(\alpha_k \star \beta_k) \vee (\alpha_{k'} \star \beta_{k'})\\
&= (x_b \star y_b) \vee ((x_b \wedge x_{b'}) \star (y_b \vee y_{b'}))\\
&= (x_b \star y_b) \vee ((x_b \star (y_b \vee y_{b'})) \wedge (x_{b'} \star (y_b \vee y_{b'})))\\
&= (x_b \star y_b) \vee ((x_b \star y_b) \wedge ((x_b \star y_{b'}) \wedge (x_{b'} \star y_b) \wedge (x_{b'} \star y_{b'})))\\
&= (x_b \star y_b)\\
&= \alpha_k \star \beta_k.
\end{aligned}$$

c) If $\{y_{k,1}, \dots, y_{k,l}\} = \emptyset$ and $\{y_{bk'_1}, \dots, y_{bk'_l}\} \neq \emptyset$, the following scheme can be used:

$$\begin{aligned}
\alpha_k \vee (\alpha_{k'} \star \beta_{k'}) &= x_b \vee ((x_b \wedge x_{b'}) \star y_b) = x_b \vee ((x_b \star y_b) \wedge (x_{b'} \star y_b))\\
&= (x_b \vee (x_b \star y_b)) \wedge (x_b \vee (x_{b'} \star y_b)) = x_b \wedge (x_b \vee (x_{b'} \star y_b))\\
&= x_b = \alpha_k.
\end{aligned}$$

d) If $\{y_{k,1}, \dots, y_{k,l}\} = \emptyset$ and $\{y_{bk'_1}, \dots, y_{bk'_l}\} = \emptyset$, the following scheme can be used:

$$\alpha_k \vee \alpha_{k'} = x_b \vee (x_b \wedge x_{b'}) = x_b = \alpha_k.$$

Step 4: It remains to be shown that two minimal disjunctive forms are propositionally equivalent if and only if they are equal modulo associativity and commutativity. This follows from the fact that formulas generated by the grammar $\mathcal{ADT}$ represent monotone Boolean functions, which have a unique minimal DNF representation modulo associativity and commutativity (see [CH11]). Hence, formulas in minimal disjunctive forms are, in fact, unique modulo associativity and commutativity. This ends the proof of Equivalence (3.2) and, hence, the proof of Lemma 3.35. □

Proof of Theorem 3.34. We recapitulate that Steps 1–4 show soundness and completeness of each axiom in G, defined in Lemma 3.35. Since G corresponds exactly to the axiom schemes in $E_{\mathcal{P}}$ when the axiom schemes are transformed into the propositional semantics, this finishes the proof. □

The complete set of axioms $E_{\mathcal{P}}$ introduced in Theorem 3.34 allows us to compare the propositional semantics (Definition 3.4) with other semantics induced by De Morgan lattices (Definition 3.21).

Theorem 3.36 *Let $\equiv_{\mathcal{P}}$ be the propositional semantics and let $\equiv_{\mathcal{DM}}$ be a semantics induced by a De Morgan lattice. The propositional semantics $\equiv_{\mathcal{P}}$ is finer than $\equiv_{\mathcal{DM}}$.*

Proof. It is sufficient to notice that every equation in the complete set of axioms $E_{\mathcal{P}}$ for the propositional semantics is also valid for $\equiv_{\mathcal{DM}}$. According to Theorem 3.32, this proves that $\equiv_{\mathcal{P}}$ is finer than $\equiv_{\mathcal{DM}}$. □

In other words, Theorem 3.36 states that the propositional semantics is the finest amongst all semantics induced by De Morgan lattices.

3.5.3 A Complete Set of Axioms for $\equiv_{\mathcal{M}}$

We now give a complete set of axioms for the multiset semantics.

Theorem 3.37 *The following set, denoted by $E_{\mathcal{M}}$, is a complete set of axioms for the multiset semantics for ADTerms of proponent type.*

$$\{(E_1^s), (E_2^s), (E_3^s), (E_4^s), (E_5^s), (E_6^s), (E_9^{\mathrm{o}}), (E_{10}^{\mathrm{p}}), (E_{11}^{\mathrm{p}}), (E_{12}^{\mathrm{o}}),\\ (E_{13}^{\mathrm{p}}), (E_{16}^{\mathrm{p}}), (E_{17}^{\mathrm{p}}), (E_{18}^{\mathrm{p}}), (E_{19}^{\mathrm{o}}), (E_{20}^{\mathrm{p}})\},$$

where $X, Y, X_i, Y_j \in \mathrm{VAR}$, $i, j, k, n \in \mathbb{N}^+$, $s \in \{\mathrm{p}, \mathrm{o}\}$ and Sym_k denotes the set of all bijections from $\{1, \ldots, k\}$ to itself.

Note that contrary to the equations in the set $E_{\mathcal{P}}$, some of the equations in $E_{\mathcal{M}}$ only hold for either the proponent, e.g., (E_{10}^{p}), or the opponent, e.g., (E_9^{o}). We remark that by switching the type in the ten axioms containing p and o, we yield a set of axioms for the multiset semantics for the ADTerms of opponent type.

Proof. We make use of class rewriting by setting up an equational rewriting system. Then we show that the system is strongly terminating and class confluent, which guarantees that the system has unique normal forms modulo the given equations, see [Pla93].[4] Finally, we show how the normal forms can be used to prove completeness of the axioms. The outline of the proof is as follows:

1. We transform $E_{\mathcal{M}}$ into an equational term rewriting system R.
2. We provide expressions which describe the normal forms of R.
3. We show strong termination of R.
4. We show confluence of R.
5. We prove that the expressions given in Step 2 describe the normal forms of R.
6. Using the normal forms, we show that the multiset semantics ($\equiv_{\mathcal{M}}$) is equal to the equational semantics induced by $E_{\mathcal{M}}$ ($\equiv_{E_{\mathcal{M}}}$).

Step 1: In order to define an equational term rewriting system (ETRS), see [FJN93], we divide the equations in $E_{\mathcal{M}}$ into two parts. Equations (E_1^s)–(E_6^s) express commutativity and associativity of the operators $\vee^s$ and $\wedge^s$ and serve in our system as equations. We turn the remaining ten equations into rewrite rules by directing them from left to right. By R we denote the ETRS composed of Equations (E_1^s)–(E_6^s) and directed rewrite rules corresponding to the Equations (E_9^{o}), (E_{10}^{p}), (E_{11}^{p}), (E_{12}^{o}), (E_{13}^{p}), (E_{16}^{p}), (E_{17}^{p}), (E_{18}^{p}), (E_{19}^{o}), (E_{20}^{p}).

Step 2: We introduce the operator C^{p} to ease notation. Let $M = \{\!|t_1, \ldots, t_m|\!\}$, $t_i \in \mathbb{T}_\Sigma^{\mathrm{p}}$, for $i \in \{1, \ldots, m\}$ and $m \in \mathbb{N}^+$, be a multiset of ADTerms of proponent type.

[4] The notion of class rewriting and class confluence correspond to the notions of rewriting and confluence when rewiring equivalence classes instead of unique expressions.

Moreover, let $M' = \{\!\{t'_1, \dots, t'_l\}\!\}$, $t'_j \in \mathbb{T}^{\mathrm{o}}_{\Sigma}$, for $j \in \{1, \dots, l\}$ and $l \in \mathbb{N}$, be a multiset of ADTerms of opponent type. The operator C^{p} is defined by

$$\begin{aligned} \mathrm{C}^{\mathrm{p}} \colon \mathcal{M}(\mathbb{T}^{\mathrm{p}}_{\Sigma}) \times \mathcal{M}(\mathbb{T}^{\mathrm{o}}_{\Sigma}) &\to \mathbb{T}^{\mathrm{p}}_{\Sigma}, \\ (M, M') &\mapsto \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(t_1, \dots, t_m), \vee^{\mathrm{o}}(t'_1, \dots, t'_l)). \end{aligned}$$

The operator is well-defined due to associativity and commutativity. This is guaranteed by the undirected Equations (E^s_1)–(E^s_6) and we leave out the precise technical details.

With the help of this operator, we define expressions which serve as normal forms for R. Let $I \subseteq \mathbb{N}$ be a nonempty index set. For every $k \in I$, let B_k be a finite multiset of basic actions of the proponent, such that $|B_k| \geq 1$ and let C_k be a finite multiset of basic actions of the opponent, such that $|C_k| \geq 0$. Then, the following expressions represent ADTerms of the proponent which are in normal form with respect to R

$$\bigvee_{k \in I}^{\mathrm{p}} \mathrm{C}^{\mathrm{p}}(B_k, C_k), \tag{3.3}$$

where $\bigvee^{\mathrm{p}}$ represents the unranked function symbol $(\vee^{\mathrm{p}}_k)_{k \in I}$. Moreover, we require that, for $k \neq k'$, we have $(B_k, C_k) \neq (B_{k'}, C_{k'})$.

Step 3: We prove strong termination of R with the help of the AProVE software tool [RWT13]. AProVE can handle ETRS, but it cannot handle unranked functions. In order to overcome this problem, we use currying (see [Sch24]). We create curried versions of $\vee^s$ and $\wedge^s$, which are unary functions, denoted as $\vee^s_{cu}$ and $\wedge^s_{cu}$, respectively. A specific list of arguments of an unranked function would, for example, be encoded in the following way: for the expression $\vee^s(a, b, c)$, we write $\vee^s_{cu}(v(v(u(a), u(b)), u(c)))$. Consequently, due to currying, we add the following rewrite rules $\vee^s_{cu}(v(x, y)) \to \vee^s(\vee^s_{cu}(x), \vee^s_{cu}(y))$ and $\vee^s_{cu}(u(x)) \to x$ and similar rules for $\wedge^s$.

We input the ETRS in AProVE using the syntax that was developed for the competitions of the Workshop on Termination (see [MRZ05]). Due to input restrictions in AProVE, the syntax, given in Figure 3.3, uses the transformations $\mathtt{a} = \vee^{\mathrm{p}}, \mathtt{b} = \vee^{\mathrm{o}}, \mathtt{c} = \wedge^{\mathrm{p}}, \mathtt{d} = \wedge^{\mathrm{o}}, \mathtt{e} = \mathrm{c}^{\mathrm{p}}, \mathtt{f} = \mathrm{c}^{\mathrm{o}}, \mathtt{g} = \vee^{\mathrm{p}}_{cu}, \mathtt{h} = \vee^{\mathrm{o}}_{cu}, \mathtt{k} = \wedge^{\mathrm{p}}_{cu}, \mathtt{l} = \wedge^{\mathrm{o}}_{cu}, \mathtt{v} = v, \mathtt{u} = u$.

Figure 3.3 details the input to the software tool. It can be understood in the following way: The line THEORY together with the first four lines of rewrite rules correspond to Equations (E^s_1)–(E^s_6) and represent associativity, commutativity and currying of the operators $\vee^s$ and $\wedge^s$. The other rules correspond to the remaining ten equations. Briefly summarized, the tool provides the following output: Strong termination of R is shown by multiple application of polynomial interpretation and removal of redundant rewrite rules.

Step 4: To prove confluence of R, it suffices to show that all critical pairs are joinable. We prove the joinability with the help of the software tool $\mathsf{T_TT_2}$, see [KSZM13]. Unfortunately $\mathsf{T_TT_2}$ cannot handle equational rewriting. We circumvent this problem by adding one rewrite rule for commutativity and two for associativity for each of the binary operators, as shown in Figure 3.4. As output, we obtain that all critical pairs are joinable.

Step 5: We show that all proponent ADTerms represented by Expression (3.3) are irreducible and that all other ADTerms are reducible.

```
(VAR  x y z)
(THEORY  (AC  a b c d))
(RULES

g(v(x,y)) → a(g(x),g(y))              h(v(x,y)) → b(h(x),h(y))
k(v(x,y)) → c(k(x),k(y))              l(v(x,y)) → d(l(x),l(y))
g(u(x)) → x                           h(u(x)) → x
k(u(x)) → x                           l(u(x)) → x

c(x,a(y,z)) → a(c(x,y),c(x,z))        b(x,d(y,z)) → d(b(x,y),b(x,z))
a(x,x) → x                            d(x,x) → x

e(e(x,y),z) → e(x,b(y,z))             c(x,e(y,z)) → e(c(x,y),z)
e(a(x,y),z) → a(e(x,z),e(y,z))        e(x,d(y,z)) → a(e(x,y),e(x,z))
e(x,f(y,z)) → a(e(x,y),c(x,z))        b(x,f(y,z)) → f(b(x,y),e(z,x))
)
```

Figure 3.3: An equational term rewriting system in the syntax of the Workshop on Termination that proves termination of the rewrite rules from Theorem 3.37.

```
a(x,y) → a(y,x)   a(x,a(y,z)) → a(a(x,y),z)    a(a(x,y),z) → a(x,a(y,z))
b(x,y) → b(y,x)   b(x,b(y,z)) → b(b(x,y),z)    b(b(x,y),z) → b(x,b(y,z))
c(x,y) → c(y,x)   c(x,c(y,z)) → c(c(x,y),z)    c(c(x,y),z) → c(x,c(y,z))
d(x,y) → d(y,x)   d(x,d(y,z)) → d(d(x,y),z)    d(d(x,y),z) → d(x,d(y,z))
```

Figure 3.4: Additional rewrite rules that have to be added to the input of AProVE, due to conversion of equations into rewrite rules.

If the symbol $\vee^{\mathrm{p}}$ exists in an ADTerm represented by Expression (3.3), it is always the root symbol. For these ADTerms, it is easily seen that, the only rewrite rule corresponding to Equation (E^{p}_{11}) has $\vee^{\mathrm{p}}$ as the root symbol on the left-hand side. However, an ADTerm can only be rewritten using this rewrite rule if the arguments of $\vee^{\mathrm{p}}$ are not distinct, which is specifically excluded in the ADTerms represented by Expression (3.3).

The ADTerms represented by Expression (3.3) that do not contain $\vee^{\mathrm{p}}$, have either $\wedge^{\mathrm{p}}$ or c^{p} as the root symbol. In the former case, the expressions are irreducible. In the latter case, the rewrite rules corresponding to Equations (E^{p}_{13}), (E^{p}_{16}), (E^{p}_{17}) and (E^{p}_{18}) may be applicable because they contain $\vee^{\mathrm{p}}$. In these rules the other occurring operators are c^{p}, $\vee^{\mathrm{p}}$, $\wedge^{\mathrm{o}}$ and c^{o}, respectively. None of these, however, appear in the ADTerms represented by Expression (3.3). Hence, we conclude that none of the ADTerms represented by Expression (3.3) can be reduced.

Next we show that every proponent ADTerm which is not represented by Expression (3.3) can be rewritten. First, we remark that the ADTerms represented by Expression (3.3) are of proponent type only. Suppose a sub-ADTerm to be rewritten is of opponent type. Then it is either a subterm of an ADTerm of proponent type, or already excluded by the theorem. Since c^{p} is the only operator that takes an ADTerm of opponent type and outputs an ADTerm of proponent type, we know that if there exists a subterm of opponent type, the complete ADTerm must contain c^{p}.

We classify the ADTerms with respect to the number of (not necessarily distinct) constants they contain.

ADTerms with one constant They can be divided into two classes: ADTerms of proponent type and ADTerms of opponent type. The former are in normal form, the latter are not covered by the theorem since they are ADTerms of opponent type.

ADTerms with two constants We also subdivide ADTerms with two constants into ADTerms opponent type and ADTerms of proponent type. The ADTerms in the first subclass again are ADTerms of opponent type and not covered by the theorem. In the second subclass, an ADTerm is either in normal form or it is of the form $\vee^{\mathrm{p}}(b, b)$. Then, the rewrite rule corresponding to Equation (E_{11}^{p}) can be used.

ADTerms with three or more constants They are classified as follows:

a) ADTerms that contain c^{o} or $\wedge^{\mathrm{o}}$. These are either of opponent type (and, therefore, not covered by the theorem) or can be rewritten using the rewrite rules corresponding to Equations (E_{9}^{p}), (E_{12}^{o}), (E_{16}^{p}), (E_{18}^{p}) or (E_{19}^{o}).

b) ADTerms that contain a nested $\vee^{\mathrm{p}}$. These are either of opponent type or can be rewritten using the rewrite rules corresponding to Equations (E_{10}^{p}) or (E_{11}^{p}).

c) ADTerms that contain a c^{p} which is not preceded by a $\vee^{\mathrm{p}}$ operator. These terms are either of opponent type or can be rewritten using the rewrite rules corresponding to Equations (E_{10}^{p}), (E_{13}^{p}), (E_{17}^{p}) or (E_{20}^{p}).

d) All remaining ADTerms. They are in normal form or contain at most one functional symbol $\circ$ in $\{\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}, \vee^{\mathrm{o}}\}$, which may appear several times. In the case were $\circ = \vee^{\mathrm{o}}$ the ADTerm is of opponent type, in case $\circ = \wedge^{\mathrm{p}}$ it is actually already in normal form and in case $\circ = \vee^{\mathrm{p}}$ it can be rewritten if the rewrite rule corresponding to Equation (E_{11}^{p}) can be applied otherwise it is already in normal form.

Step 6: Together Steps 1–5 show that R is a convergent ETRS with the unique normal forms represented by Expression (3.3). It remains to be shown that the equations $E_{\mathcal{M}}$, are sound and complete with respect to the multiset semantics. We can easily verify soundness by proving that every equation in $E_{\mathcal{M}}$ holds in the multiset semantics. In other words, for all $t, t' \in \mathbb{T}_{\Sigma}^{\mathrm{p}}$, it holds that from $t \equiv_{E_{\mathcal{M}}} t'$ it follows that $t_{\mathcal{M}} = t'_{\mathcal{M}}$. This is essentially due to the fact $\langle \mathcal{M}, \cup, \otimes \rangle$ forms a semiring. For example, Equation (E_{20}^{p}) concretely yields:

$$\wedge^{\mathrm{p}}(x, \mathrm{c}^{\mathrm{p}}(y, z))_{\mathcal{M}} = x_{\mathcal{M}} \otimes (y_{\mathcal{M}} \otimes z_{\mathcal{M}}) = (x_{\mathcal{M}} \otimes y_{\mathcal{M}}) \otimes z_{\mathcal{M}} = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(x, y), z)_{\mathcal{M}}.$$

All other cases are similar.

Finally, we prove completeness by showing that, for $t, t' \in \mathbb{T}^{\mathrm{p}}_{\Sigma}$, from $t_{\mathcal{M}} = t'_{\mathcal{M}}$ it follows that $t \equiv_{E_{\mathcal{M}}} t'$. To facilitate reasoning, let $\mathrm{NF}(t)$ and $\mathrm{NF}(t')$ denote the normal forms obtained by R, represented by Expression (3.3), in other words, $\mathrm{NF}(t) \equiv_{E_{\mathcal{M}}} t$ and $\mathrm{NF}(t') \equiv_{E_{\mathcal{M}}} t'$. Since the elements considered for the multiset semantics are sets of pairs of multisets, there exists a one-to-one correspondence between such sets of pairs of multisets and the normal forms given by the ADTerm represented by Expression (3.3): Each pair of multisets is mapped to a pair (B_k, C_k) which corresponds to ADTerms in normal form, as shown in Steps 2–5. The pairs (B_k, C_k) are mutually different for different indices k because C^{p} is injective modulo associativity and commutative i.e., we map different sets to different elements. We conclude that given $t_{\mathcal{M}} = t'_{\mathcal{M}}$, it follows that $\mathrm{NF}(t) = \mathrm{NF}(t')$. Consequently, from $t_{\mathcal{M}} = t'_{\mathcal{M}}$, it follows that $t \equiv_{E_{\mathcal{M}}} \mathrm{NF}(t) = \mathrm{NF}(t') \equiv_{E_{\mathcal{M}}} t'$, which concludes the proof of Theorem 3.37. □

4

Quantitative Analysis

Graphical visualization and interpretation of potential attacks and possible countermeasures constitute a first step towards systematic security analysis. A potential next step is to amend the model with quantitative analysis, meaning to add numerical values to the model. When performing quantitative security analysis, we *ask questions related to specific aspects or properties that influence the security of a system or a company.* These questions may be binary (i.e., yes/no?), e.g., “Is the attack realizable?” or may concern physical or temporal aspects, e.g., “What are the minimal costs of successfully attacking a system?” or “How long does it take to detect the attack?” We answer these questions with the help of *attributes.* Attributes provide a powerful analysis tool for vulnerability scenarios. They express a particular property of an attack–defense scenario.

To quantify an attribute, we associate attribute values with the nodes of an ADTree. Then, these basic values are used to determine a value for the entire scenario. For some questions, such as, “What is the minimal skill level that the attacker needs in order to execute an attack?” these basic values can be provided more easily than for other questions, such as, “How probable is a given attack to succeed?” If known, the basic values are assigned directly, otherwise they are estimated by experts or deduced from historical knowledge on or statistical data about similar attacks or defenses.

In this chapter, we first provide an overview of existing attributes in the literature and elaborate on what needs to be considered when evaluating an attribute value for a scenario in Section 4.1. We then formally introduce attributes on ADTrees in Section 4.2. Section 4.3 examines connections between attributes and semantics. Section 4.4 advances a classification of attributes arising in practice. This classification supports users of the attack–defense tree methodology in finding an adequate formalization for attributes they are interested in. The chapter concludes in Section 4.5 describing different ways to construct new attributes.

4.1 Historical Overview of Attributes

Before we formalize the notion of quantitative analysis with the help of ADTrees, we provide a historical perspective. We argue that rigorous formalization is a prerequisite for accurate numerical analysis. In the literature, quantitative measures are generally called attributes or metrics. Some attributes, for example, **costs** or **probability**, appear frequently in the context of attack trees, others, such as **severity** or **response time** can rarely be found. For readability, we always write names of attributes in **bold font**.

4.1.1 Attributes for Attack Trees

In [SSSW98][5], Salter et al. propose to decorate attack trees with values expressing the component's **costs** in an attempt to deduce the cheapest attack for the modeled scenario. Similarly they provide values for other attributes such as **probability**, **severity**, **consequence** to derive the most probable, the most severe and the most consequential attack. They also suggest using the attributes **skill** level and **impact** to determine the attack with the highest required skill level or impact. According to Schneier [SSSW98, Sch99], all of these attributes can be combined or conditioned in order to, for example, identify the cheapest low-risk attack, attacks that cost less than a given amount of money or cheap, highly successful attacks where only medium skill is needed. Other researchers have extended this catalog of attributes further. For example, Edge et al. [EDRM06] introduce protection **costs**. Byres et al. [BFM04] use **detectability** to illustrate how easily a defender can discover an attack and **difficulty** which specifies the needed skill of the attacker. Henniger et al. [HAF+09] suggest **attack time**, to model the amount of time needed to perform an attack and consider it to be independent of an attacker's **skill**, **costs** or **resources**. They also suggest combining a set of attributes, namely **elapsed time**, **expertise**, **knowledge of system**, **window of opportunity** and **required equipment** in order to deduce the **required attack potential**. Fung et al. [FCW+05] show how the **difficulty** level can be used to estimate the scenario's **survivability**. Buldas et al. [BLP+06] define **costs** of an attack, **probability of success** and expected **penalty** of a successful as well as of an unsuccessful attack to derive the expected **outcome** of the attacker. Later Jürgenson and Willemson [JW08] combine the attributes **costs**, **probability of success** and **penalty** differently to obtain an exact expected **outcome** attribute that also respects the propositional semantics as defined by Mauw and Oostdijk [MO05].

4.1.2 Attributes for Defensive Aspects

Traditionally, attack trees only consider attributes directly related to attacks or attackers. With the advent of graphical formalisms that consider protections, such as protections trees, see Section 7.4.3, defense trees, see Section 7.4.3, and naturally ADTrees, defensive aspects have been included in quantitative analysis. Many attack-related attributes can be considered for attack–defense scenarios as well. For instance, the **costs** attribute can refer to the **costs** of performing an attack or the **costs** of installing a defense [EDRM06]. Similarly, the **probability of occurrence** [RKT12a], the **probability of success** [RKT12a] and the required **skill** level [ERG+07] can be adapted. Other attributes, such as the **number of possible countermeasures** [RKT10b], the **probability of detection** [ERG+07] or the **response time** might only make sense when defenders are taken into account.

4.1.3 Value Domains

To be able to answer, for example, how high the **costs** of an attack are, we need to specify all possible values of the attribute. Attribute values can range over

[5] The idea is often only attributed to Schneier and then cited as [Sch99].

the following diversified types of domains. For attributes that can adopt only two values, a binary value domain is most suitable. For other attributes, values from a nominal scale, e.g., `Low`, `Medium` and `High`, real numbers or even discrete or continuous probability distributions may be more adequate. Instead of a single value, it is also possible to associate sets of values to the nodes. For example, if we know that the attack **costs** are not `High`, we could associate the set $\{\texttt{Low}, \texttt{Medium}\}$ to the corresponding node. To increase readability all attribute values are typeset in `typewriter font`.

4.1.4 Calculation Procedure

As pointed out by Slater et al. [SSSW98], in the case of attack trees, the most intuitive way to determine an attribute value for a scenario is the bottom-up approach. The idea is to only associate attribute values with the basic actions and then deduce the values corresponding to the refined nodes from the values associated with their children. The functions which are used to calculate the value for a parent node depend on the kind of the corresponding refinement. Hence, the input of the calculation procedure is the *values associated to the basic actions* and *functions that declare how to combine the values.*

Example 4.1 We graphically illustrate the bottom-up procedure for attack trees on the attack tree given in Figure 1.1. For illustrative purposes we assume that the values of a generic attribute are given as 4, 3 and 2 for the leaf nodes "Disjunctive Subgoal 1", "Conjunctive Subgoal 1" and "Conjunctive Subgoal 2", respectively. Moreover, we use addition to deduce values for conjunctive nodes and minimization to compute values for disjunctive nodes. Hence we need to evaluate $3 + 2 = 5$ to obtain the value associated to the "Disjunctive Subgoal 1" and $\min\{4, 5\} = 4$ to obtain the value associated to the root node "Main Goal". The generic attribute for the entire scenario is then computed to be 4 as illustrated in Figure 4.1.

The bottom-up approach has several advantages. First of all, we only have to find values for all leaf nodes and no values for any refined node. Second, this approach is suitable for evaluation of a large number of attributes since it can be automated. Third, the algorithm is simple and fast since it is linear in the number of nodes. Finally, assigning a value for basic actions should be feasible in most cases since basic actions can be easily understood when the tree is set up properly. A good understanding of the nodes allows us to easily quantify them. If a node cannot be easily understood, it should be refined further.

As it is the case for all calculation procedures, the quality of the result depends on the accuracy of the input values. Instead of asking one expert to provide values, we can improve the quality of the input values by asking several security experts and encouraging them to agree on a consensus value for each node. Another strategy is to involve several people, such as the system owner, developers and administrators, to provide values for subtrees relevant to their expertise. Alternatively, if past data is available, the values could be estimated. If there are no other options, a user has to use his best guess. Naturally, we face the same obstacles for the bottom-up procedure extended to ADTrees, which we introduce in the next section. We do not further discuss how to obtain the most appropriate values for the basic actions that serve as input values for the bottom-up computation.

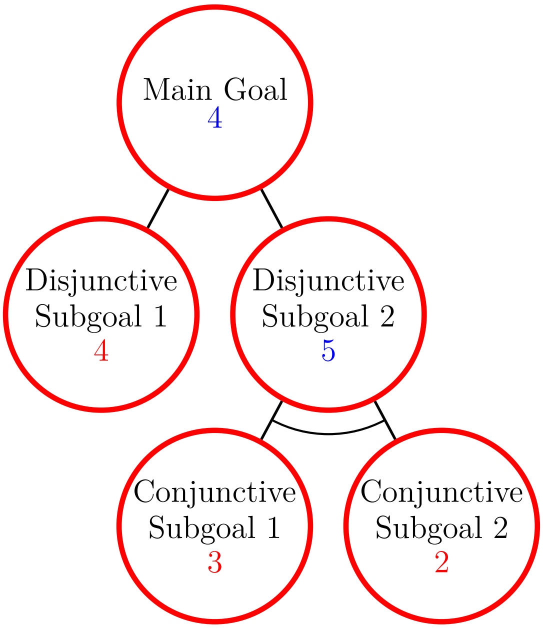


Figure 4.1: Generic attribute evaluation on an attack tree. Values in red represent initially assigned values, values in blue are computed values.

4.2 Formal Definition of Attributes on ADTrees

In order to facilitate and automate the quantitative analysis of attack–defense scenarios we introduce a bottom-up algorithm on ADTrees. Our algorithm extends an intuitive bottom-up procedure often attributed to Schneier [Sch99], which was formalized by Mauw and Oostdijk [MO05]. By an *attribute* we understand any quantifiable aspect or property of a security model of a scenario. Before giving numerous detailed examples, we start by defining the notion of an attribute domain A_α which formally specifies an attribute α.

Definition 4.2 (Attribute domain for ADTerms) An attribute domain for ADTerms A_α is a tuple

$$A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha),$$

where D_α, the *value domain*, is a set of values and, for $s \in \{\mathrm{p}, \mathrm{o}\}$,

- $\vee^s_\alpha$, $\wedge^s_\alpha$ are unranked functions on D_α,
- c^s are binary functions on D_α.

Since attack trees only have nodes of type of the proponent, we can deduce the attribute domain on attack terms (and, therefore, the attribute domain on attack trees) from a corresponding attribute domain on ADTerms. Given the attribute domain for ADTerms $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}})$, the corresponding attribute domain for attack terms is given by $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha)$. With this framework, the bottom-up evaluation of an attribute on attack trees presented in Section 4.1.4 can be described as follows: first, values from D_α are assigned to all leaf nodes of an attack tree. Then, the values for the remaining nodes are deduced by traversing the tree from the leaves to the root, with the help of the operations $\vee^{\mathrm{p}}_\alpha$ and $\wedge^{\mathrm{p}}_\alpha$.

This is demonstrated by Example 4.1, where $\vee_\alpha^{\mathrm{p}}$ is instantiated as min, $\wedge_\alpha^{\mathrm{p}}$ as $+$ and $D_\alpha \supset \{2,3,4,5\}$. We formalize and extend this procedure to ADTrees in the next two definitions.

Definition 4.3 (Basic assignment) Let D_α be a value domain and $\mathbb{B}$ the set of all basic actions. A function $\beta_\alpha \colon \mathbb{B} \to D_\alpha$ that assigns values from D_α to all basic actions $\mathbb{B}$ is called a basic assignment.

The basic assignment allows us to formally define how to compute attribute values corresponding to all subtrees of an ADTree by using the unranked functions $\vee_\alpha^s$, $\wedge_\alpha^s$ and c_α^s for $s \in \{\mathrm{p}, \mathrm{o}\}$ in a bottom-up way.

Definition 4.4 (Bottom-up evaluation for ADTerms) Suppose we are given an attribute domain for ADTerms $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ and a basic assignment $\beta_\alpha \colon \mathbb{B} \to D_\alpha$. Then, the recursively defined function $\alpha \colon \mathbb{T}_\Sigma \to D_\alpha$ assigns to every ADTerm t, the value of the attribute α of t, as follows:

$$\alpha(t) = \begin{cases} \beta_\alpha(t), & \text{if} \quad t \in \mathbb{B}; \\ \vee_\alpha^s(\alpha(t_1), \ldots, \alpha(t_k)), & \text{if} \quad t = \vee^s(t_1, \ldots, t_k); \\ \wedge_\alpha^s(\alpha(t_1), \ldots, \alpha(t_k)), & \text{if} \quad t = \wedge^s(t_1, \ldots, t_k); \\ \mathrm{c}_\alpha^s(\alpha(t_1), \alpha(t_2)), & \text{if} \quad t = \mathrm{c}^s(t_1, t_2), \end{cases} \tag{4.1}$$

where $s \in \{\mathrm{p}, \mathrm{o}\}$ and $k \in \mathbb{N}^+$.

The following sections illustrate the bottom-up attribute evaluation on four different attributes of increasing complexity.

4.2.1 The Satisfiability Attribute

In order to analyze whether an attack–defense scenario is feasible, we introduce the notion of satisfiability of an ADTerm by defining the **satisfiability** attribute sat. It can be used to answer the question “Can the proponent realize the considered scenario?” In other words, it tells us if the proponent can succeed in or win the scenario? The **satisfiability** attribute sat is a function $\mathrm{sat} \colon \mathbb{T}_\Sigma \to D_{\mathrm{sat}} = \{\mathtt{true}, \mathtt{false}\}$. First, we specify the corresponding basic assignment. If the proponent can execute a basic action, we set $\beta_{\mathrm{sat}}(b) = \mathtt{true}$ for $b \in \mathbb{B}^{\mathrm{p}}$. Otherwise, i.e., if the proponent cannot execute an action b or we are unsure whether or not he can execute it, we set $\beta_{\mathrm{sat}}(b) = \mathtt{false}$ for $b \in \mathbb{B}^{\mathrm{p}}$. Similarly, we assign the basic actions of the opponent the value `true` if the opponent can execute them and `false` if he cannot. If we do not have any knowledge about the basic action of the opponent, they are set to `true`. This reflects the worst case for the proponent since it assumes that the opponent can satisfy all his actions. Given the basic assignment, we recursively define:

$$\mathrm{sat}(t) = \begin{cases} \beta_{\mathrm{sat}}(t), & \text{if} \quad t = b \in \mathbb{B}; \\ \vee(\mathrm{sat}(t_1), \ldots, \mathrm{sat}(t_k)), & \text{if} \quad t = \vee^s(t_1, \ldots, t_k); \\ \wedge(\mathrm{sat}(t_1), \ldots, \mathrm{sat}(t_k)), & \text{if} \quad t = \wedge^s(t_1, \ldots, t_k); \\ \mathrm{sat}(t_1) \wedge\neg\, \mathrm{sat}(t_2), & \text{if} \quad t = \mathrm{c}^s(t_1, t_2), \end{cases}$$

where $k \in \mathbb{N}^+$. The underlying attribute domain is given by:

$$A_{\text{sat}} = (\{\texttt{true}, \texttt{false}\}, \vee, \wedge, \vee, \wedge, \star, \star),$$

where $\star$ is defined as $\star(x,y) = x \wedge \neg\, y$ for all $x, y \in D_{\text{sat}}$.

Example 4.5 Suppose we want to know whether the proponent's goal modeled by the ADTree from Example 2.2 is satisfied or not. For the sake of this example, we assume that an attacker can outnumber the guards, use a weapon, steal the keys and sabotage the cameras, but he cannot bribe the guard. We also assume that the defender can install security cameras. We use the attribute domain $A_{\text{sat}} = (\{\texttt{true}, \texttt{false}\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $x \star y = x \wedge \neg y$, for all $x, y \in \{\texttt{true}, \texttt{false}\}$, as well as the basic assignment $\beta_{\text{sat}} : \mathbb{B} \to \{\texttt{true}, \texttt{false}\}$ that assigns the value $\texttt{true}$ to every basic action which is satisfiable and the value $\texttt{false}$ to every basic action which is not satisfiable. Using the recursive evaluation procedure defined by Equation (4.1), we evaluate the **satisfiability** attribute on the ADTerm from Example 2.6. We obtain

$$\begin{aligned}
&\mathrm{sat}(\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC})))\\
=\ &\star\,(\vee(\beta_{\text{sat}}(\mathrm{BG}), \wedge(\beta_{\text{sat}}(\mathrm{OG}), \beta_{\text{sat}}(\mathrm{UW})), \beta_{\text{sat}}(\mathrm{SK})), \star(\beta_{\text{sat}}(\mathrm{IC}), \beta_{\text{sat}}(\mathrm{SC})))\\
=\ &\star\,(\vee(\texttt{false}, \wedge(\texttt{true}, \texttt{true}), \texttt{true}), \star(\texttt{true}, \texttt{true})) = \star(\texttt{true}, \texttt{false}) = \texttt{true}\,.
\end{aligned}$$

The evaluation is also depicted graphically on the ADTree in Figure 4.2. Naturally, the computation of attribute values on trees and corresponding terms always yields the same result. Intuitively, the value is obtained in the following way. Since an attacker can outnumber the guard and can use a weapon, he is capable of subduing the guard. Alternatively, the attacker could steal the keys from the guard. Additionally, the attacker can sabotage the cameras, which could be put in place as defenses. Together these actions allow the attacker to defeat the guard.

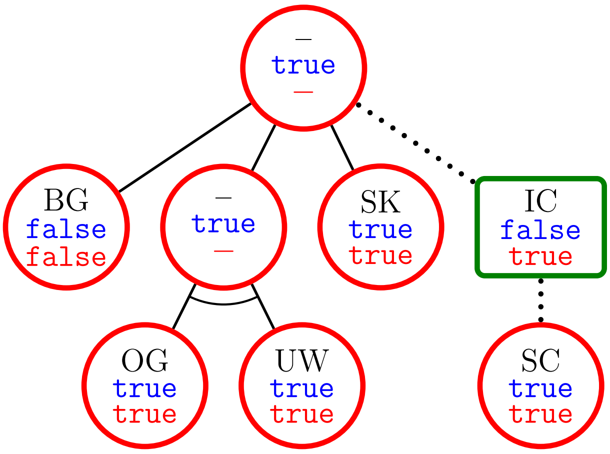


Figure 4.2: The attribute **satisfiability** (sat) and a basic assignment. Red values (on the bottom) depict basic assignment values, blue values (in the middle) depict the computed sat values.

When we slightly reinterpret the **satisfiability** attribute, it also allows us to define the **winner** of the considered attack–defense scenario. If the **satisfiability** value calculated for an ADTerm is equal to `true`, the **winner** of the corresponding scenario is the proponent. If it is `false`, the **winner** is the opponent. In Example 4.5, the value corresponding to the root is `true`, hence, the **winner** is the proponent, which in the scenario is the attacker.

In fact, the attribute domain A_{sat} can be used to answer a recursively defined **binary question**, i.e., a question with only two possible answers which, therefore, can be modeled with Boolean variables. Besides the **satisfiability** attribute, the same attribute domain can be used to model the **presence** or **absence** of a system component and need for **insider knowledge**, **special skill** or **electricity**. Using the same strain of thought, the attribute domain is well-suited to handle *annotating flags*, such as a description whether or not certain nodes are **unmitigable**. It can also be used to simulate hypothetical scenarios. If we wanted to check whether or not a system would be attackable during a power outage, the basic assignment of all components that need **electricity** could be set to `false`. The regular bottom-up evaluation is evaluated on

$$A_\alpha = (\{\texttt{true}, \texttt{false}\}, \vee, \wedge, \vee, \wedge, \star, \star),$$

where α is "sat", "elec" or any other of the previously mentioned attributes. The only difference is that the interpretation of `true` and `false` changes from attribute to attribute.

4.2.2 The Minimal Costs Attribute

Another piece of valuable information about a scenario is a description of the involved costs. More specifically, the question "What are the minimal costs of the proponent, assuming that reusing tools is infeasible?" can be formally modeled using the **costs** attribute. For this, we use the following attribute domain

$$A_{\mathrm{costs}} = (\mathbb{R}^+_\infty, \min, +, +, \min, +, \min), \tag{4.2}$$

where $\mathbb{R}^+_\infty = \mathbb{R}^+ \cup \{\infty\}$. This attribute domain together with Equation (4.1) suffices to determine the costs function that is used to recursively evaluate the **costs** attribute:

$$\mathrm{costs}(t) = \begin{cases} \beta_{\mathrm{costs}}(t), & \text{if} \quad t = b \in \mathbb{B}; \\ \min\{\mathrm{costs}(t_1), \dots, \mathrm{costs}(t_k)\}, & \text{if} \quad t = \vee^{\mathrm{p}}(t_1, \dots, t_k); \\ +(\mathrm{costs}(t_1), \dots, \mathrm{costs}(t_k)), & \text{if} \quad t = \wedge^{\mathrm{p}}(t_1, \dots, t_k); \\ +(\mathrm{costs}(t_1), \dots, \mathrm{costs}(t_k)), & \text{if} \quad t = \vee^{\mathrm{o}}(t_1, \dots, t_k); \\ \min\{\mathrm{costs}(t_1), \dots, \mathrm{costs}(t_k)\}, & \text{if} \quad t = \wedge^{\mathrm{o}}(t_1, \dots, t_k); \\ +(\mathrm{costs}(t_1), \mathrm{costs}(t_2)), & \text{if} \quad t = \mathrm{c}^{\mathrm{p}}(t_1, t_2); \\ \min\{\mathrm{costs}(t_1), \mathrm{costs}(t_2)\}, & \text{if} \quad t = \mathrm{c}^{\mathrm{o}}(t_1, t_2). \end{cases}$$

As basic assignments of the attacker, we assign the real **costs** to the corresponding basic actions. Since the defender's **costs** have no influence on the attacker's **costs**, the values associated with the defender's basic actions express the **costs** from the attacker's point of view rather than the actual **costs** for the defender. To reflect this fact, every basic action of the defender is assigned ∞, which is the absorbing element for the operation $\mathrm{c}^{\mathrm{p}}_{\mathrm{costs}} = +$ and the neutral element for the operation $\mathrm{c}^{\mathrm{o}}_{\mathrm{costs}} = \min$. Therefore, the assignment corresponds to a worst case assumption when no information about the **costs** of an action is available.

Additionally assuming that the proponent is the attacker, we verbally describe the application of Equation (4.1) to the **costs** attribute. To ease presentation, we distinguish the following three cases depending on the structure of the ADTree:

Subtrees rooted in a node which is refined but not countered

- The **costs** calculated for a subtree rooted in a disjunctively refined attack (resp. defense) node are defined as the minimum (resp. addition) of the **costs** calculated for its refining subtrees.

- The **costs** calculated for a subtree rooted in a conjunctively refined attack (resp. defense) node are defined as the addition (resp. minimum) of the **costs** calculated for its refining subtrees.

Subtrees rooted in a node which is not refined but countered The **costs** calculated for a subtree rooted in an attack (resp. defense) node are defined as the addition (resp. minimum) of the initial value for the non-refined root node and the value calculated for the countering subtree.

Subtrees rooted in a node which is refined and countered In this case, first a value corresponding to a refining part of the tree is calculated. This is done in the same way as in the case of a subtree rooted in a refined but not countered node. Then, the functions for a subtree rooted in a non-refined but countered node are used, where the initial value for the root is replaced with the calculated value for the refining part.

We also illustrate the computation of the **costs** attribute on our running example:

Example 4.6 Suppose we want to evaluate the **costs** on our running example defeating a guard. The ADTerm $\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC}))$ was computed in Example 2.6. We know that $\mathrm{BG}, \mathrm{OG}, \mathrm{UW}, \mathrm{SK}, \mathrm{SC} \in \mathbb{B}^{\mathrm{p}}$ and $\mathrm{IC} \in \mathbb{B}^{\mathrm{o}}$. Since the defender's **costs** have no influence on the attacker's **costs**, $\beta_{\mathrm{costs}}(\mathrm{IC})$ is assigned ∞. For concreteness we suppose that the attacker's minimal investment to execute the attacker's basic actions is given by:

$$\begin{aligned} &\beta_{\mathrm{costs}}(\mathrm{BG}) = 1€, \quad \beta_{\mathrm{costs}}(\mathrm{UW}) = 2€, \quad \beta_{\mathrm{costs}}(\mathrm{SC}) = 4€, \\ &\beta_{\mathrm{costs}}(\mathrm{OG}) = 2€, \quad \beta_{\mathrm{costs}}(\mathrm{SK}) = 3€. \end{aligned}$$

Note that the attacker has to potentially perform the action SC in order to counteract the defender's action IC. By using the appropriate operators $\mathrm{c}^{\mathrm{p}}_{\mathrm{costs}} = +$ and $\mathrm{c}^{\mathrm{o}}_{\mathrm{costs}} = \min$, we can compute the actual minimal **costs** for the attacker to succeed in the scenario. The computation is performed on the corresponding ADTree in Figure 4.3 and on the corresponding ADTerm (in prefix notation) in the following:

$$\begin{aligned}
&\mathrm{costs}(\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC})))\\
=& +(\min\{\beta_{\mathrm{costs}}(\mathrm{BG}), +(\beta_{\mathrm{costs}}(\mathrm{OG}), \beta_{\mathrm{costs}}(\mathrm{UW})), \beta_{\mathrm{costs}}(\mathrm{SK})\},\\
&\quad \min\{\beta_{\mathrm{costs}}(\mathrm{IC}), \beta_{\mathrm{costs}}(\mathrm{SC})\})\\
=& +(\min\{1€, +(2€, 2€), 3€\}, \min\{\infty, 4€\})\\
=& +\{1€, 4€\}\\
=& \, 5€.
\end{aligned}$$

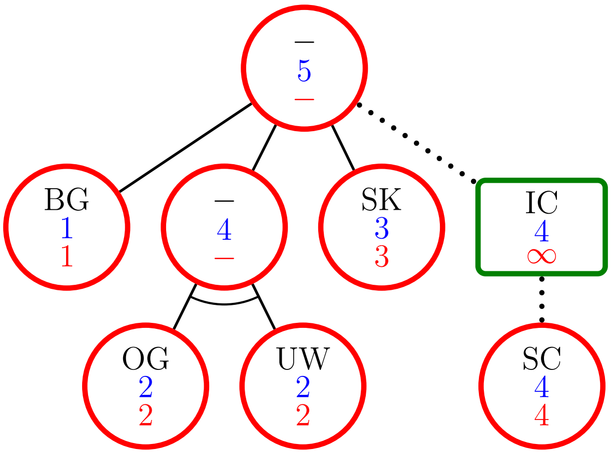


Figure 4.3: The attribute **costs** (costs) and a basic assignment. Red values (on the bottom) depict the basic assignment, blue values (in the middle) depict the computed costs values.

We also illustrate the computation on our second example running.

Example 4.7 The ADTerm $t = \vee^{\mathrm{p}}(b_1, \mathrm{c}^{\mathrm{p}}(b_2, d))$ was used in Example 3.3. We know that $b_1, b_2 \in \mathbb{B}^{\mathrm{p}}$ and $d \in \mathbb{B}^{\mathrm{o}}$. Since the opponent's basic action d is not under control of the proponent, we set $\beta_{\mathrm{costs}}(d) = \infty$. For concreteness we suppose, that the minimal investment to execute the attacker's basic action are $\beta_{\mathrm{costs}}(b_1) = 2€$ and $\beta_{\mathrm{costs}}(b_2) = 3€$ for b_1 and b_2. Then,

$$\mathrm{costs}(t) = \min\{2€, +(3€, \infty)\}) = 2€.$$

The attribute domain given in Equation (4.2) is not the only possible attribute domain that helps us to calculate the minimal **costs** of the attacker. Suppose we are only interested in whether the minimal **costs** are `Low`, `Medium` or `High` and not in the exact monetary amount. Then we can choose a different value domain to yield a different attribute domain. For example, the following attribute domain would then be appropriate.

$$A_{\mathrm{costs}_2} = (\{\texttt{Low}, \texttt{Medium}, \texttt{High}\}, \min, \max, \max, \min, \max, \min),$$

with `Low` < `Medium` < `High`. Hidden in this attribute domain is an implicit assumption that executing several actions of the same cost category does not make the scenario more expensive.

4.2.3 The Cheapest Successful Proponent's Bundle Attribute

Attributes can even be used to quantify more complex questions. Suppose we ask ourselves "What is the cheapest way for the proponent to succeed and which actions does he then have to execute?" To answer this question, we use the **cheapest successful proponent's bundle** attribute, abbreviated as **cspb**.

Selecting the value domain $\mathcal{P}(\mathbb{B}^{\text{p}}) \times \mathbb{R}_\infty$, where $\mathbb{R}_\infty = \mathbb{R} \cup \{\infty\}$, the basic assignment is a pair. More specifically, we assign to a proponent's node b the pair $(\{b\}, c_b)$, where c_b expresses the proponent's costs that are incurred when performing the action b. To the opponent's nodes we associate the pair $(\emptyset, \infty)$. This assignment indicates that no action exists that would allow the proponent to be successful at this node ($\emptyset$) and that he would theoretically have to spent an infinite amount (∞) if he nevertheless wanted to be successful.

The **cspb** attribute is given by the following attribute domain:

$$((\mathcal{P}(\mathbb{B}^{\text{p}}) \times \mathbb{R}_\infty), \text{cspb}_{\vee^{\text{p}}}, \text{cspb}_{\wedge^{\text{p}}}, \text{cspb}_{\vee^{\text{o}}}, \text{cspb}_{\wedge^{\text{o}}}, \text{cspb}_{\text{c}^{\text{p}}}, \text{cspb}_{\text{c}^{\text{o}}}),$$

where

$$\begin{aligned}
\text{cspb}_{\vee^{\text{p}}}((S_1, c_1), \dots (S_k, c_k)) &= (S_i, c_i), \quad \text{such that} \quad c_i = \min\{c_1, \dots, c_k\}.\\
\text{cspb}_{\wedge^{\text{p}}}((S_1, c_1), \dots (S_k, c_k)) &= \begin{cases} (\bigcup_{i=1}^k S_i, \Sigma_{i=1}^k c_i), & \text{if} \quad \Sigma_{i=1}^k c_i < \infty;\\ (\emptyset, \infty), & \text{else.} \end{cases}\\
\text{cspb}_{\vee^{\text{o}}}((S_1, c_1), \dots (S_k, c_k)) &= \text{cspb}_{\wedge^{\text{p}}}((S_1, c_1), \dots (S_k, c_k)).\\
\text{cspb}_{\wedge^{\text{o}}}((S_1, c_1), \dots (S_k, c_k)) &= \text{cspb}_{\vee^{\text{p}}}((S_1, c_1), \dots (S_k, c_k)).\\
\text{cspb}_{\text{c}^{\text{p}}}((S_1, c_1), (S_2, c_2)) &= \begin{cases} (\emptyset, \infty), & \text{if} \quad S_2 = \emptyset;\\ (S_1 \cup S_2, c_1 + c_2), & \text{else.} \end{cases}\\
\text{cspb}_{\text{c}^{\text{o}}}((S_1, c_1), (S_2, c_2)) &= (S_i, c_i), \quad \text{such that} \quad c_i = \min\{c_1, c_2\}.
\end{aligned}$$

If several c_i result in the minimum, any i can be chosen to determine S_i. By default, we choose the smallest i to obtain a unique value for S_i.

Intuitively the operators can be explained from the perspective of the proponent as follows. The $\text{cspb}_{\wedge^{\text{p}}}$ operator describes that the proponent chooses to execute the cheapest action. The operator $\text{cspb}_{\wedge^{\text{p}}}$ expresses that the proponent has to execute all actions in order to be successful. He can, however, only do so if none of the actions are too expensive and, therefore, all are possible. A similar intuition explains $\text{cspb}_{\vee^{\text{p}}}$. If the proponent cannot defend against the opponent's countermeasure, he will never be successful. Otherwise the actions related to both nodes need to be executed and the corresponding **costs** will occur.

The **cspb** attribute allows the following interpretation. Let the pair $\text{cspb}(t) = (S, c)$ be associated with the term t. Then S expresses which basic components have to be executed by the proponent in order to make him successful in the scenario represented by t. The value c expresses the corresponding **costs** for the proponent.

Remark 4.8 If we restrict the operators of the attribute domain A_{cspb} to be binary, it holds that $\text{cspb}_{\vee_2^{\text{p}}}$, $\text{cspb}_{\wedge_2^{\text{o}}}$ and $\text{cspb}_{\text{c}^{\text{o}}}$ as well as $\text{cspb}_{\wedge_2^{\text{p}}}$, $\text{cspb}_{\vee_2^{\text{o}}}$ and $\text{cspb}_{\text{c}^{\text{p}}}$ coincide.

Example 4.9 We illustrate the **cspb** attribute on the ADTerm

$$\text{c}^{\text{p}}(\vee^{\text{p}}(\text{BG}, \wedge^{\text{p}}(\text{OG}, \text{UW}), \text{SK}), \text{c}^{\text{o}}(\text{IC}, \text{SC})),$$

which is already described in Example 2.6. For concreteness, we assume the following **costs** values: 1€ for BG, 2€ for OG, 3€ for UW, 4€ for SK and 1€ for SC. Hence the basic assignment for the nodes of the proponent is given by the tuples $(\{\text{BG}\},1)$, $(\{\text{OG}\},2)$, $(\{\text{UW}\},3)$, $(\{\text{SK}\},4)$ and $(\{\text{SC}\},1)$. To the defense node IC we assign the default value $(\emptyset,\infty)$. To compute the value of the root, we recursively compute the values starting from the leaves. The value of each leaf is simply the given value. The value of the defensive node is computed by evaluating $\text{cspb}_{\text{c}^{\text{o}}}((\emptyset,\infty),(\{\text{SC}\},1))$ as $(\{\text{SC}\},1)$. Similarly, we compute the value of the conjunctive node using $\text{cspb}_{\wedge^{\text{p}}}$ to be $(\{\text{OG},\text{UW}\},5)$. Finally, we use $\text{cspb}_{\vee^{\text{p}}}$ and $\text{cspb}_{\text{c}^{\text{p}}}$ to compute the value of the root as $(\{\text{BG},\text{SC}\},2)$. This value expresses that the proponent needs to execute the nodes BG and SC in order to win the scenario and that the corresponding minimal **costs** are 2. The computation is graphically illustrated on the ADTree in Figure 4.4.

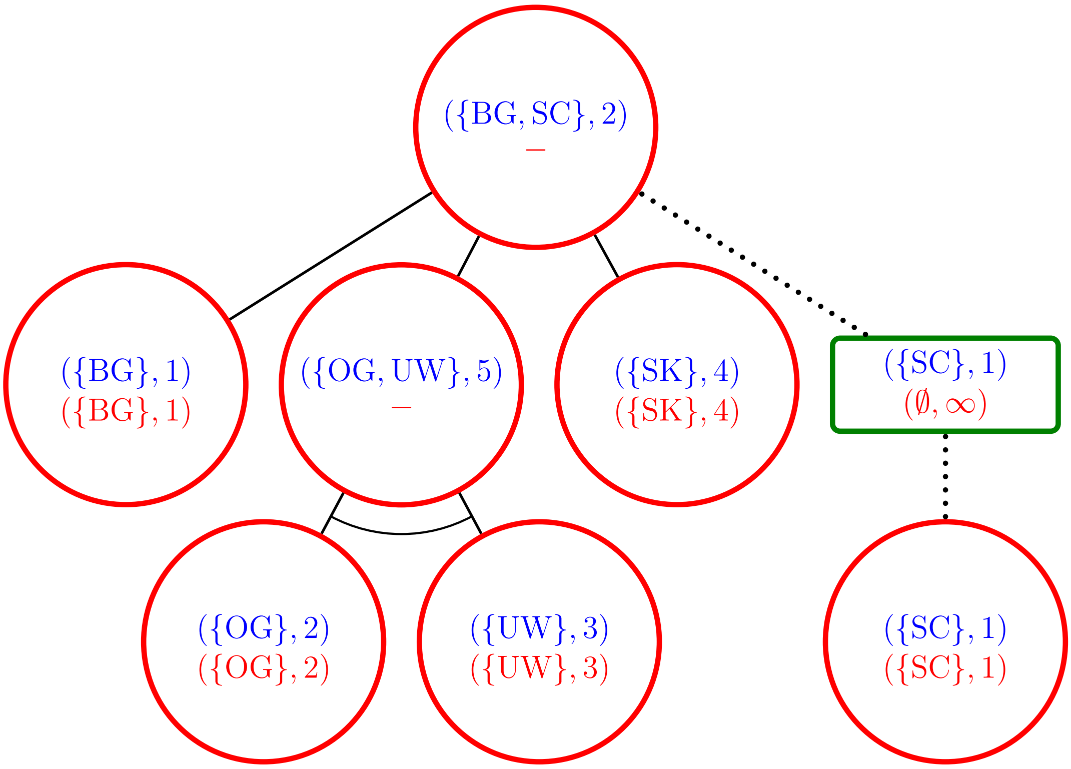


Figure 4.4: The attribute **cspb** (cspb) and a basic assignment. Red values (on the bottom) depict the basic assignment, blue values (on the top) depict the computed cspb values.

4.2.4 The Minimal Costs of the Winner Attribute

Our final example of an attribute answers a question for which we also need to know two values for the basic assignment. We use the attribute **minimal costs of the winner**, abbreviated **mcw**, if we want to know "What are the minimal costs of the winner of the scenario?" This attribute combines information from the first two examples. For the basic assignment we use the values `Low` (`L`), `Medium` (`M`), `High` (`H`), `Extreme` (`X`), `true` and `false` with `Low` < `Medium` < `High` < `Extreme` and `false` < `true` to create the value domain $\{\mathtt{L},\mathtt{M},\mathtt{H},\mathtt{X}\}\times\{\mathtt{true},\mathtt{false}\}$.

The basic assignment is given by $\beta_{\mathrm{mcw}}(b) = (c_b, x_b)$, where c_b and x_b express the actual **costs** and the **satisfiability** of the component b for the player corresponding to b. For instance, $\beta_{\mathrm{mcw}}(b) = (\mathtt{M}, \mathtt{true})$, for $b \in \mathbb{B}^{\mathrm{o}}$, means that the **costs** of the component are `Medium` and that the opponent can satisfy it (`true`). If an action is not satisfiable (`false` for the second component), its **costs** are `Low`.

Note that this default value is neutral for the operator $\mathrm{mcw}_{\vee^{\mathrm{p}}}$ and absorbing for $\mathrm{mcw}_{\wedge^{\mathrm{p}}}$. Also this assignment represents a worst case scenario for the respective player since an assignment of `false` means that the other player succeeds and it is worst if the other player then has to pay as little as possible. This assignment is contrary to the default value for the **costs** attribute (∞), where the value denotes the **costs** of the proponent and not the **costs** of the **winner** of the scenario.

To apply the bottom-up algorithm, we apply the following attribute domain:

$$(\{\mathtt{L}, \mathtt{M}, \mathtt{H}, \mathtt{X}\} \times \{\mathtt{true}, \mathtt{false}\}, \mathrm{mcw}_{\vee^{\mathrm{p}}}, \mathrm{mcw}_{\wedge^{\mathrm{p}}}, \mathrm{mcw}_{\vee^{\mathrm{o}}}, \mathrm{mcw}_{\wedge^{\mathrm{o}}}, \mathrm{mcw}_{\mathrm{c}^{\mathrm{p}}}, \mathrm{mcw}_{\mathrm{c}^{\mathrm{o}}}),$$

where $\mathtt{L} < \mathtt{M} < \mathtt{H} < \mathtt{X}$ and

$$\mathrm{mcw}_{\vee^{\mathrm{p}}}((c_1, x_1), \ldots, (c_k, x_k)) = \begin{cases} (\min\limits_{\{i \mid x_i = \mathtt{true}\}} \{c_i\}, \mathtt{true}), & \text{if } \exists i, x_i = \mathtt{true}; \\ (\max\limits_{1 \le i \le k} \{c_i\}, \mathtt{false}), & \text{else.} \end{cases}$$

$$\mathrm{mcw}_{\wedge^{\mathrm{p}}}((c_1, x_1), \ldots, (c_k, x_k)) = \begin{cases} (\max\limits_{1 \le i \le k} \{c_i\}, \mathtt{true}), & \text{if } \forall i, x_i = \mathtt{true}; \\ (\min\limits_{\{i \mid x_i = \mathtt{false}\}} \{c_i\}, \mathtt{false}), & \text{else.} \end{cases}$$

$$\mathrm{mcw}_{\vee^{\mathrm{o}}}((c_1, x_1), \ldots, (c_k, x_k)) = \mathrm{mcw}_{\vee^{\mathrm{p}}}((c_1, x_1), \ldots, (c_k, x_k)).$$

$$\mathrm{mcw}_{\wedge^{\mathrm{o}}}((c_1, x_1), \ldots, (c_k, x_k)) = \mathrm{mcw}_{\wedge^{\mathrm{p}}}((c_1, x_1), \ldots, (c_k, x_k)).$$

$$\mathrm{mcw}_{\mathrm{c}^{\mathrm{p}}}((c_1, x_1), (c_2, x_2)) = \begin{cases} (c_2, \mathtt{false}), & \text{if } x_1 = \mathtt{true}, x_2 = \mathtt{true}; \\ (c_1, \mathtt{false}), & \text{if } x_1 = \mathtt{false}, x_2 = \mathtt{false}; \\ (\max\{c_1, c_2\}, \mathtt{true}), & \text{if } x_1 = \mathtt{true}, x_2 = \mathtt{false}; \\ (\min\{c_1, c_2\}, \mathtt{false}), & \text{if } x_1 = \mathtt{false}, x_2 = \mathtt{true}\,. \end{cases}$$

$$\mathrm{mcw}_{\mathrm{c}^{\mathrm{o}}}((c_1, x_1), (c_2, x_2)) = \mathrm{mcw}_{\mathrm{c}^{\mathrm{p}}}((c_1, x_1), (c_2, x_2)).$$

The value $\mathrm{mcw}(t) = (c, x)$ expresses the **costs** of the scenario represented by the term t for the winning player. For instance, given an ADTerm $t = \vee^{\mathrm{p}}(t_1, t_2, t_3)$, such that $\mathrm{mcw}(t_1) = (\mathtt{L}, \mathtt{false})$ (i.e., the opponent is successful at t_1 and his **costs** are `Low`), $\mathrm{mcw}(t_2) = (\mathtt{H}, \mathtt{true})$ (i.e., the proponent is successful at t_2 and his **costs** are `High`) and $\mathrm{mcw}(t_3) = (\mathtt{M}, \mathtt{true})$ (i.e., the proponent is successful at t_3 and his **costs** are `Medium`), we have $\mathrm{mcw}(t) = (\mathtt{M}, \mathtt{true})$. Indeed, the proponent can be successful either by proceeding with t_2 and investing a `High` amount or by proceeding with t_3 and investing a `Medium` amount. We chose t_3 because it is less expensive and we are interested in the lowest **costs**.

Example 4.10 We also illustrate the **mcw** attribute on the ADTerm

$$\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{BG}, \wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{SK}), \mathrm{c}^{\mathrm{o}}(\mathrm{IC}, \mathrm{SC})),$$

introduced in Example 2.6. For concreteness, we assume that the attacker only executes the three actions BG, OG and UW. The respective costs are given by `M`, `L` and `H`. The defender executes IC for a `Medium` (`M`) amount of `costs`. The remaining two basic actions SK and SC are not executed and are, therefore, assigned the default value, i.e., the lowest possible costs `L`. Hence the basic assignment for the six nodes BG, OG, UW, SK, IC and SC is given by the six tuples $(\mathtt{M},\mathtt{true})$, $(\mathtt{L},\mathtt{true})$, $(\mathtt{H},\mathtt{true})$, $(\mathtt{L},\mathtt{false})$ $(\mathtt{M},\mathtt{true})$ and $(\mathtt{L},\mathtt{false})$, respectively. To compute the value of the root, we recursively compute the values starting from the leaves. The value of each leaf is simply the given value. The value of the defensive node is computed by evaluating $\mathrm{mcw}_{c^o}((\mathtt{M},\mathtt{true}),(\mathtt{L},\mathtt{false}))$ as $(\mathtt{M},\mathtt{true})$. Similarly, we compute the value of the conjunctive node using $\mathrm{mcw}_{\wedge^p}$ to be $(\mathtt{H},\mathtt{true})$. Finally, we use $\mathrm{mcw}_{\vee^p}$ and mcw_{c^p} to compute the value of the root as $(\mathtt{M},\mathtt{false})$. This value expresses that in this example the opponent wins the attack–dense scenario and in order to win he incurs a `Medium` amount of costs. The computation is graphically illustrated on the ADTree in Figure 4.5.

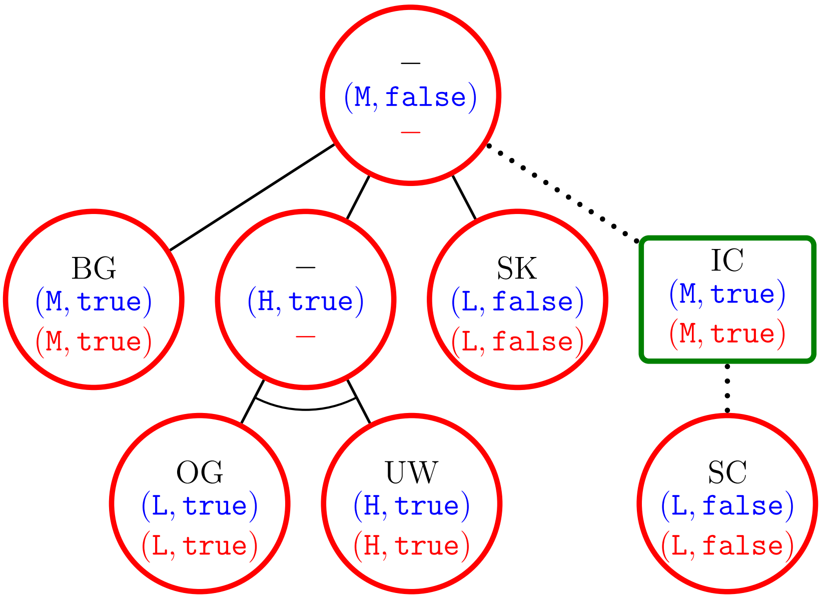


Figure 4.5: The attribute **mcw** (mcw) and a basic assignment. Red values (in the middle) depict the basic assignment, blue values (on the top) depict the computed mcw values.

Remark 4.11 To illustrate that the **mcw** attribute exhibits a structure similar to the structure of the **sat** attribute, we analyze $\mathrm{mcw}_{\wedge^p}$ in its binary form, i.e., $\mathrm{mcw}_{\wedge^p_2}$. Additionally we define a negation function $\neg$ as follows:

$$\neg(c_1, x_1) = (c_1, \neg x_1).$$

Algebraic transformations show that $a_1 \,\mathrm{mcw}_{\wedge^p_2} \neg a_2 = a_1 \,\mathrm{mcw}_{c^p}\, a_2 = a_1 \,\mathrm{mcw}_{c^o}\, a_2$, where $a_i = (c_i, x_i)$ for $i \in \{1, 2\}$.

In Section 4.4.3, we specify the structure of the attribute domains for **sat** and **mcw** and generalize it to a class of attributes.

Remark 4.12 Replacing the value domain with $\mathbb{R}_\infty \times \{\mathtt{true}, \mathtt{false}\}$ and all max-functions with $+$, yields absolute cost values (from $\mathbb{R}$).

In Section 4.4, we come back to an empirical evaluation of attribute domains of attributes that can be found in the literature. We classify the structure and point out pitfalls that may occur when trying to deduce the correct attribute domain of an attribute. In Section 4.5 we provide pointers on how to generalize the theory of attribute composition, illustrated by the last two examples.

4.3 Compatibility of an Attribute with a Semantics

In Chapter 3, we elaborated that we consider equivalent ADTrees to be indistinguishable. Thus, the evaluation of attributes on equivalent ADTerms should be consistent, i.e., should yield the same values. However, as shown in the following example, this is not always the case.

Example 4.13 Consider the two ADTrees given in Figure 4.6. Their ADTerm representation is given by $t = \vee^{\mathrm{p}}(b_1, \mathrm{c}^{\mathrm{p}}(b_2, d))$ and $t' = \vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(b_1, b_1), \mathrm{c}^{\mathrm{p}}(b_2, d))$.

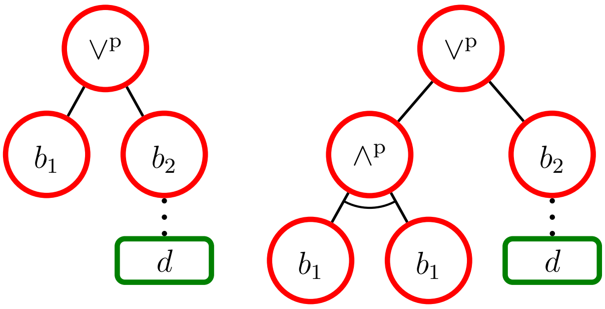


Figure 4.6: Two propositionally equivalent ADTrees (t represented on the left and t' on the right) that do not yield the same values for the attribute **costs**.

In Example 3.5, we have shown that $t \equiv_{\mathcal{P}} t'$. Moreover, in Example 4.7, we have computed that $\mathrm{costs}(t) = 2€$. Using the same basic assignment $\beta_{\mathrm{costs}}(b_1) = 2€$, $\beta_{\mathrm{costs}}(b_2) = 3€$ and $\beta_{\mathrm{costs}}(d) = \infty$, the evaluation of the proponent's minimal **costs** for t' yields $\mathrm{costs}(t') = \min\{+(2€, 2€), +(3€, \infty)\} = 4€$, which is not the same value as for t.

4.3.1 Consistent Bottom-up Evaluation

The problem of consistent bottom-up evaluation of attribute values has already been discussed in the case of attack trees [MO05, JW08]. The authors of [MO05] identify a sufficient condition guaranteeing that, when the multiset semantics is used, the bottom-up evaluation of a specific class of attributes on attack trees is consistent. In this section, we generalize this result to any semantics for ADTerms, by introducing the notion of compatibility of an attribute domain with a semantics for ADTerms. Compatibility constitutes a sufficient condition for consistent bottom-up evaluation of attributes on equivalent ADTrees. Since equivalence of ADTree is determined by the used semantics, both the semantics and the attribute domain have to be specified when checking the compatibility criterion.

Consider an attribute domain

$$A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha),$$

the set $\mathbb{T}_\Sigma^{\mathrm{VAR}}$ of typed ADTerms over the variables in VAR, as introduced in Section 3.4, and an element t of $\mathbb{T}_\Sigma^{\mathrm{VAR}}$. We denote by t_α an expression built from the elements of $\mathbb{B} \cup \mathrm{VAR}$ and the operators $\vee_\alpha^s, \wedge_\alpha^s, \mathrm{c}_\alpha^s$, for $s \in \mathcal{S}$, recursively defined as follows. Let $t^1, \ldots, t^k \in \mathbb{T}_\Sigma^{\mathrm{VAR}}$ and $k \in \mathbb{N}^+$. Then,

$$\begin{aligned} t_\alpha = t, \quad \text{if} \quad t \in \mathbb{B} \cup \mathrm{VAR}, \quad & (\vee^s(t^1, \ldots, t^k))_\alpha = \vee_\alpha^s(t_\alpha^1, \ldots, t_\alpha^k), \\ (\mathrm{c}^s(t^1, t^2))_\alpha = \mathrm{c}_\alpha^s(t_\alpha^1, t_\alpha^2), \qquad & (\wedge^s(t^1, \ldots, t^k))_\alpha = \wedge_\alpha^s(t_\alpha^1, \ldots, t_\alpha^k). \end{aligned} \tag{4.3}$$

Hence $\mathbb{T}_\Sigma \subset \mathbb{T}_\Sigma^{\mathrm{VAR}}$. Moreover, the terms t_α may contain variables from D_α. This means that, in particular, the term $f \vee_\alpha^{\mathrm{p}} g$, where f and g are in VAR, is also contained in $\mathbb{T}_\Sigma^{\mathrm{VAR}}$.

Definition 4.14 (Attribute domains compatible with a semantics) An attribute domain $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ is compatible with a semantics $\equiv$ for ADTerms if and only if, for all $t, t' \in \mathbb{T}_\Sigma$, the semantical equivalence $t \equiv t'$ implies that the equality $t_\alpha = t'_\alpha$ holds in D_α.

In other words, the equivalence check $t_\alpha = t'_\alpha$ is a functional equivalence check since t_α and t'_α may contain variables. Therefore, t_α needs to be equal to t'_α for all substitutions of variables, i.e.,$\forall \sigma(t_\alpha) = \sigma(t'_\alpha)$.

Example 4.15 Consider the following ADTerms $t = \mathrm{c}^{\mathrm{p}}(b, \wedge^{\mathrm{o}}(d_1, d_2))$ and $t' = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(b, b), \wedge^{\mathrm{o}}(d_1, d_2))$ from Example 3.5. We have shown that $t \equiv_{\mathcal{P}} t'$. By using the attribute domain $A_{\mathrm{sat}} = (\{\mathtt{true}, \mathtt{false}\}, \vee, \wedge, \vee, \wedge, \star, \star)$, introduced in Section 4.2.1, and the procedure described by Recursion (4.3), we define the expressions t_{sat} and t'_{sat} as follows:

$$t_{\mathrm{sat}} = b \wedge \neg(d_1 \wedge d_2) \qquad \text{and} \qquad t'_{\mathrm{sat}} = (b \wedge b) \wedge \neg(d_1 \wedge d_2).$$

Due to the idempotence of $\wedge$, we obtain that the equality $t_{\mathrm{sat}} = t'_{\mathrm{sat}}$ holds in $D_{\mathrm{sat}} = \{\mathtt{true}, \mathtt{false}\}$.

From Definitions 3.31 and 4.14 we can easily deduce that if an attribute domain is compatible with a semantics for ADTerms it is also compatible with every semantics which is finer.

In most cases, due to the infinite number of equivalent ADTerms, employing Definition 4.14 is impractical. The next theorem overcomes this drawback.

Theorem 4.16 *Let E be a complete set of axioms for a semantics $\equiv$ for ADTerms. An attribute domain $A_\alpha = (D_\alpha, \vee_\alpha^{\mathrm{p}}, \wedge_\alpha^{\mathrm{p}}, \vee_\alpha^{\mathrm{o}}, \wedge_\alpha^{\mathrm{o}}, \mathrm{c}_\alpha^{\mathrm{p}}, \mathrm{c}_\alpha^{\mathrm{o}})$ is compatible with the semantics $\equiv$ if and only if, for every equation $t = t'$ in E, the equality $t_\alpha = t'_\alpha$ holds in D_α.*

Proof. It follows directly from Recursion (4.3) and Definitions 3.29 and 4.14. □

Theorem 4.16 shows that a complete set of axioms is a powerful tool to ensure the practical usability of semantics for ADTerms. By making use of a complete set of axioms, Theorem 4.16 gives us a simple and efficient procedure for checking compatibility of a given attribute domain with a considered semantics.

Example 4.17 Using Theorem 4.16, we can easily prove that the attribute domain A_{costs}, used in Examples 4.6 and 4.13 to compute the proponent's minimal **costs**, is not compatible with the propositional semantics. Indeed, according to Theorem 3.34, the axiom $\wedge^{\mathrm{p}}(X, X, X_1, \ldots, X_k) = \wedge^{\mathrm{p}}(X, X_1, \ldots, X_k)$ holds for the propositional semantics, but in the **costs** attribute domain with the value domain $\mathbb{R}_\infty$ this does not hold in general. Suppose, $k = 2$ and all variables are equal to 1. Then the left and the right side of the axiom evaluate to

$$(\wedge^{\mathrm{p}}(1,1,1,1))_{\mathrm{costs}} = +(1,1,1,1) = 4 \neq 3 = +(1,1,1) = (\wedge^{\mathrm{p}}(1,1,1))_{\mathrm{costs}}.$$

The fact that $+$ is not idempotent in general, explains why the evaluation of the proponent's minimal **costs** on two equivalent ADTerms in the propositional semantics, presented in Example 4.13, gives two different results.

We now prove that semantically equivalent ADTerms always yield equal attribute values over compatible attribute domains.

Lemma 4.18 *Consider an attribute domain* $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$, *a basic assignment* $\beta_\alpha \colon \mathbb{B} \to D_\alpha$ *and two ADTerms* t *and* t'. *If* $t_\alpha = t'_\alpha$ *holds in* D_α, *then* $\alpha(t) = \alpha(t')$.

Proof. Since $t_\alpha = t'_\alpha$ holds in D_α, we have $\sigma(t_\alpha) = \sigma(t'_\alpha)$, for every substitution $\sigma \colon \mathbb{B} \cup \mathrm{VAR} \to D_\alpha$. Thus, it suffices to show that for every ADTerm t, we have

$$\alpha(t_\alpha) = \alpha(t). \tag{4.4}$$

The proof of the previous equation is by induction on the structure of t. If $t \in \mathbb{B}$, then $t_\alpha = t$, thus $\alpha(t_\alpha) = \beta_\alpha(t_\alpha) = \beta_\alpha(t) = \alpha(t)$. Suppose now that Equation (4.4) holds for all ADTerms composing t and let $t = \vee^{\mathrm{p}}(t^1, \ldots, t^k)$. We have

$$\begin{aligned} \alpha(t_\alpha) &= \alpha(\vee^{\mathrm{p}}_\alpha(t^1_\alpha, \ldots, t^k_\alpha)) = \vee^{\mathrm{p}}_\alpha(\alpha(t^1_\alpha), \ldots, \alpha(t^k_\alpha)) \\ &= \vee^{\mathrm{p}}_\alpha(\alpha(t^1), \ldots, \alpha(t^k)) = \alpha(t). \end{aligned}$$

The proof for the other compositions is similar.

Using Equation (4.4), we obtain that if $t_\alpha = t'_\alpha$ holds in D_α, then $\alpha(t) = \alpha(t_\alpha) = \alpha(t'_\alpha) = \alpha(t')$, which finishes the proof. □

Using the lemma, we can now show that if an attribute domain is compatible with a semantics then the bottom-up algorithm computes the same attribute value on ADTerms that are equivalent.

Theorem 4.19 *Let* $A_\alpha = (D_\alpha, \vee^{\mathrm{p}}_\alpha, \wedge^{\mathrm{p}}_\alpha, \vee^{\mathrm{o}}_\alpha, \wedge^{\mathrm{o}}_\alpha, \mathrm{c}^{\mathrm{p}}_\alpha, \mathrm{c}^{\mathrm{o}}_\alpha)$ *be an attribute domain compatible with a semantics* $\equiv$ *for ADTerms. If* $t \equiv t'$, *then, given any basic assignment* $\beta_\alpha \colon \mathbb{B} \to D_\alpha$, *we have* $\alpha(t) = \alpha(t')$.

Proof. This theorem follows immediately from Equation (4.1), Definition 4.14 and Lemma 4.18. □

4.3.2 Attribute Domains Compatible with the Multiset Semantics

We next turn our attention to attribute domains occurring in the literature. While performing case studies using the ADTree methodology, we have noticed that a large number of useful attribute domains for ADTrees admits the following template.

$$A_\alpha = (D_\alpha, \circ, \bullet, \bullet, \circ, \bullet, \circ),$$

where we use the symbols $\circ$ and $\bullet$ as placeholders for specific operators. Corresponding symbols within an attribute domain indicate that the functions coincide.

One attribute domain that exhibits this structure is

$$A_{\text{costs}} = (\mathbb{R}^+_\infty, \min, +, +, \min, +, \min),$$

which is introduced in Section 4.2.2 to calculate the minimal **costs** for the proponent to achieve his main goal. Another attribute domain following the same structure is given by

$$A_{\text{costs}_2} = (\{\mathtt{L}, \mathtt{M}, \mathtt{H}\}, \min, \max, \max, \min, \max, \min),$$

where $\mathtt{L} < \mathtt{M} < \mathtt{H}$. It is used to calculate the minimal **costs** for the proponent on a 3-level scale, as also introduced in Section 4.2.2. Finally, the attribute domain

$$A_{\text{ltk}} = (\{0, \ldots, k\}, \min, +_k, +_k, \min, +_k, \min),$$

where $+_k(b_1, \ldots, b_n) = \min\{\sum_{i=1}^n b_i, k\}$ for $b_i \in \{0, \ldots, k\}$, also follows the initial template. This attribute domain can be used to model which goals of the proponent are executable in **less than** k time units.

We say that an attribute domain is *constructed from the structure* $\langle D_\alpha, \circ, \bullet\rangle$ if it matches the template A_α as given above. For instance, A_{costs} is constructed from $\langle \mathbb{R}^+_\infty, \min, +\rangle$, the second attribute domain A_{costs} from $\langle\{\mathtt{L}, \mathtt{M}, \mathtt{H}\}, \min, \max\rangle$ and the third A_{costs_2} from $\langle\{0, \ldots, k\}, \min, +_k\rangle$.

All three structures share algebraic properties. They are, in fact, *commutative semirings* or even *idempotent semirings.*

Definition 4.20 (Commutative and idempotent semiring) A structure $\langle D_\alpha, \circ, \bullet\rangle$ is called a commutative semiring if the operations $\circ$ and $\bullet$ are both commutative and associative over the non-empty set D_α. Moreover $\bullet$ has to distribute over $\circ$, i.e., for all $a, b, c \in D_\alpha$ it holds that $a \bullet (b \circ c) = (a \bullet b) \circ (a \bullet c)$ and $(a \circ b) \bullet c = (a \bullet c) \circ (b \bullet c)$.

The structure is called an idempotent semiring if the operator $\circ$ is additionally idempotent, and D_α contains a neutral element with respect to $\circ$ which is absorbing with respect to $\bullet$.

Both structures are well-studied in the literature [PK11]. Typical examples of commutative semirings include the Boolean semiring $\langle\{0, 1\}, \max, \cdot\rangle$, the tropical semiring $\langle\mathbb{N}, \min, +\rangle$, the product t-norm semiring $\langle[0, 1], \max, \cdot\rangle$, the truncation semiring $\langle\{0, \ldots, k\}, \min, +_k\rangle$ and the arithmetic semiring $\langle\mathbb{R}, +, \cdot\rangle$. The Boolean semiring, the tropical semiring, the product t-norm semiring and the truncation semiring are even idempotent semirings.

The fact that idempotent semirings occur frequently as attribute domains can be explained with the realization that they are compatible with the multiset semantics.

Theorem 4.21 *Every attribute domain of the form $A_\alpha = (D_\alpha, \circ, \bullet, \bullet, \circ, \bullet, \circ)$, that is constructed from an idempotent semiring $\langle D_\alpha, \circ, \bullet\rangle$, is compatible with the multiset semantics for ADTerms.*

Proof. Let us consider the complete set of axioms $E_{\mathcal{M}}$ for the multiset semantics, given in Theorem 3.37. According to Proposition 4.16, it is sufficient to show that for every $l = r \in E_{\mathcal{M}}$, the equality $l_\alpha = r_\alpha$ holds in D_α. The equalities corresponding to Equation (E_1^s) and (E_2^s) result from the commutativity of $\circ$ and $\bullet$. The equalities corresponding to Equations (E_3^s), (E_4^s), (E_{17}^p) and (E_{20}^p) hold due to associativity of both operations. Distributivity of $\bullet$ over $\circ$ guarantees that the equalities corresponding to Equations (E_9^o), (E_{10}^p), (E_{13}^p), (E_{16}^p), (E_{18}^p) and (E_{19}^o) are satisfied in D_α. Finally, the equalities corresponding to Equations (E_{11}^p) and (E_{12}^o) result from the idempotence of $\circ$. Note that Equations (E_5^s) and (E_6^s) are concerned with the removal of unary operators only. □

The authors of [MO05] show that, in the case of attack trees, every attribute domain which is a semiring is compatible with the multiset semantics. Theorem 4.21 extends this result from attack trees to ADTrees. Note that the result proven in [MO05] only holds for idempotent semirings. Indeed, Equations (E_1^p), (E_2^p), (E_3^p), (E_4^p), (E_5^p), (E_6^p), (E_{10}^p) and (E_{11}^p) axiomatize the multiset semantics for attack trees. Thus, if the attribute domain forms a semiring which is not idempotent (as for instance in the case of the arithmetic semiring, i.e., $\langle \mathbb{R}, +, \cdot\rangle$), the computation of attribute values on two equivalent attack trees, such as $t = \vee^\mathrm{p}(a, a)$ and $t' = a$, for $a \in \mathbb{B}^\mathrm{p}$ does not yield the same result.

In the next section, we continue our analysis of attribute domains following a more practical approach that provides a classification of attribute domains.

4.4 Practical Use of Attributes

One of the goals of this thesis is to illustrate the benefits of a methodology that supports a graphical as well as a formal component. In the previous sections, attributes have been introduced formally. Using the transformations given in Section 2.3 quantitative analysis with the help of attributes and attribute domains can be performed directly on ADTrees instead of ADTerms. In practice, however, we are not given an attribute domain. Instead we are often asked to analyze a scenario and are supposed to answer certain questions about the scenario. Hence, the user is still faced with another challenge, namely to find a relevant attribute domain for each question.

In the following we describe how to correctly specify a question for an attack–defense scenario in order to construct the corresponding attribute domain. This means that we explain which components a question needs to have in order to avoid underspecified questions. This then allows us to employ the ADTree methodology and deduce a quantitative answer using the bottom-up procedure. We motivate our approach with the following example.

Example 4.22 “What are the costs of the considered scenario?” appears to be a suitable question for the ADTree methodology. However, this question is underspecified, because we do not know whether we should quantify the attacker’s

costs, the defender's **costs** or both. Nor is it clear whether we are interested in the minimal, maximal, average of other **costs**. Clarifying this information is necessary to correctly define the corresponding basic assignment.

In the following we show that the question "What are the minimal costs of the attacker, provided that the defender executes all possible defenses?" is adequately specified for the ADTree methodology.

The example also shows that underspecified questions are not a new phenomenon of ADTrees, but already occur in the case of attack trees.

4.4.1 Classification of Questions

We specify questions by providing a pragmatic taxonomy of quantitative questions that can be asked about ADTrees. The presented classification originates from case studies, e.g., [12BKMS], [EDRM06, TA10], as well as from a detailed literature study concerning quantitative analysis of security. Our study allowed us to identify three main classes of empirical questions, as described below.

Class 1: Questions referring to one player Typically, questions for ADTrees have an explicit or implicit reference to one of the players which we call the *owner* of the question. This is motivated by the fact that the security model is usually analyzed from the point of view of one player only, for instance a defender. This player (the owner of the question) knows his own capabilities but does not have extensive information concerning his adversary. Thus, the owner of the question is only able to quantify his own actions precisely and he assumes the worst case scenario with respect to the actions of the other player. Intuitive examples of questions referring to one player are "What are the minimal costs of the attacker?" or "How much does it at least cost to always protect the system?" Most of the security relevant aspects that are analyzed in the context of attack trees can also be answered on ADTrees. Questions related to attributes such as attacker's or defender's **costs** [Sch99, BLP$^+$06, TA10, BP10, SDP08, MO05, Yag06, ACK10, RKT12a, BFM04, Ame12, WWPP11, EDRM06], attack **detectability** [TA10, BFM04], **attack time** or **defense time** [HAF$^+$09, Sch99, WWPP11], **difficulty** of attack or protection [BFM04, FCW$^+$05, TA10, HAF$^+$09, MO05, ACK10, Amo94, WWPP11], attacker's **special skill** [MO05, ACK10, Sch99], attacker's **profit** [Amo94, JW08, BDP06, RKT12a] and **penalty** [BLP$^+$06, JW08, WWPP11], all belong to Class 1. We analyze questions of this class in Section 4.4.2.

Class 2: Questions where answers for both players can be deduced from each other Exemplary questions belonging to Class 2 are "Is the scenario satisfiable?" or "How probable is it that the scenario will succeed?" We observe that if the scenario is satisfied for the attacker, then it is not satisfied for the defender and vice versa. Similarly, knowing that one player succeeds with probability p, we also know that the other player succeeds with probability $1 - p$. The foremost goal of attack trees and all their extensions is to represent whether attacks are possible. Thus, the **satisfiability** attribute is considered, either explicitly or implicitly, in all works concerning attack trees and their extensions. As for **probability**[6], the attribute has been extensively studied in [Sch99, BLP+06, HAF+09, LLFH09, MTF11, Yag06, ACK10, RKT12a, BFM04, EDRM06, WWPP11]. We perform a detailed analysis of questions of Class 2 in Section 4.4.3.

Class 3: Questions referring to an outside third party Questions belonging to Class 3 relate to a universal property which is influenced by actions of both players. They quantify attack–defense scenarios from the point of view of an outside third party which is neither the attacker nor the defender. For instance, one could ask “How much data traffic is involved in the attack–defense scenario?” In this case, we do not need to distinguish between traffic resulting from the attacker’s and the defender’s actions since both players contribute to the total amount. Another example of a question of Class 3 is “What is the global environmental impact of the scenario?” Examples of environmental **impact** could be CO_2 emissions or water pollution. Attributes corresponding to questions of Class 3 have not been addressed in the attack tree literature since attack trees focus on a single player and do not combine values from two different players. The importance of those questions becomes apparent when actions of two opposite parties are considered. The case study [12BKMS] that we have performed using the attack–defense tree methodology showed that such attributes relate to essential properties which should not be disregarded in the security assessment process. Questions of Class 3 are discussed in Section 4.4.4.

Attack trees form a subset of ADTrees which involve only one player, the attacker. Due to this simplified setting, in the case of attack trees, the three classes of questions coincide.

Next, we set up guidelines that explain how to correctly specify quantitative questions of all three classes. The guidelines’ main purpose is to enable us to find a suitable attribute domain in order to correctly compute an answer using the bottom-up procedure. Figure 4.7 depicts the three classes of questions, as well as general templates for the corresponding attribute domains. The symbols $\circ$, $\bullet$, $\diamond$ and $\bullet\neg$ again serve as placeholders for specific operators. Corresponding symbols within a tuple indicate that the functions coincide. For instance, $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ means that $\vee^{\mathrm{p}}_{\alpha} = \wedge^{\mathrm{o}}_{\alpha} = \mathrm{c}^{\mathrm{o}}_{\alpha}$ and that $\wedge^{\mathrm{p}}_{\alpha} = \vee^{\mathrm{o}}_{\alpha} = \mathrm{c}^{\mathrm{p}}_{\alpha}$. We motivate these equalities and give possible instantiations of $\circ, \bullet, \diamond$ and $\bullet\neg$ in the following three sections.

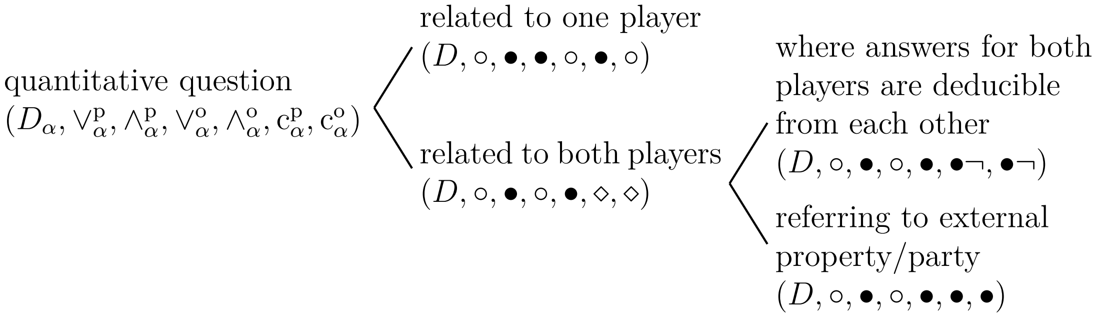

quantitative question
$(D_\alpha, \vee^{\mathrm{p}}_{\alpha}, \wedge^{\mathrm{p}}_{\alpha}, \vee^{\mathrm{o}}_{\alpha}, \wedge^{\mathrm{o}}_{\alpha}, \mathrm{c}^{\mathrm{p}}_{\alpha}, \mathrm{c}^{\mathrm{o}}_{\alpha})$
related to one player
$(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$
related to both players
$(D, \circ, \bullet, \circ, \bullet, \diamond, \diamond)$
where answers for both players are deducible from each other
$(D, \circ, \bullet, \circ, \bullet, \bullet\neg, \bullet\neg)$
referring to external property/party
$(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$

Figure 4.7: Classification of questions and attribute domain templates.

[6] We would like to point out that the **probability** attribute can only be evaluated using the bottom-up procedure given by Equation (4.1) if the ADTree does not contain probabilistically dependent actions.

4.4.2 Questions Referring to One Player

Questions belonging to Class 1 refer to exactly one player, the owner, denoted as own. As we explain below, in the attack–defense tree setting, only two situations occur for a question's owner: either he needs to choose *at least one* option or he needs to execute *all* options. Therefore, two operators suffice to answer questions of Class 1 and the generic attribute domain is of the form $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$. Furthermore, as we elaborate in Section 4.5.4, if we change a question's owner, the attribute domain changes from $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ into $(D, \bullet, \circ, \circ, \bullet, \circ, \bullet)$.

Defining a Formal Model for Questions of Class 1 In this section we illustrate the construction of the formal model for Class 1 using the question "What are the minimal costs of the attacker?", where the owner is the attacker. In questions of Class 1, all values assigned to nodes and subtrees answer the question from the perspective of its owner. In the minimal **costs** example, this means that even subtrees rooted in defense nodes have to be quantified from the attacker's point of view, i.e., a value assigned to the root of a subtree expresses what the minimal amount of money is that the attacker needs to invest in order to be successful in the current subtree.

Subtrees rooted in uncountered attacker's nodes can either be disjunctively or conjunctively refined. In the first case the attacker needs to ensure that he is successful in *at least one* of the refining nodes, in the second case he needs to be successful in *all* refining nodes. The situation for subtrees rooted in uncountered defender's nodes is reversed. If a defender's node is disjunctively refined, the attacker needs to successfully counteract *all* possible defenses to ensure that he is successful at the subtree's root node. If the defender's node is conjunctively refined, successfully counteracting *at least one* of the refining nodes already suffices for the attacker to be successful at the subtree's root node.

This reversal explains that two different operators suffice to quantify all possible uncountered trees: The operator that we use to combine attribute values for disjunctively refined nodes of one player is the same as the operator we use for conjunctively refined nodes of the other player.

Furthermore, the same two operators can also be used to quantify all remaining subtrees. If a subtree is rooted in a countered attacker's node, the attacker needs to ensure that he is successful at the action represented by the root node *and* that he successfully counteracts the existing defensive measure. Dually, for the attacker to be successful in a subtree rooted in a defender's countered node, it is sufficient to successfully overcome the defensive action *or* to successfully perform the attack represented by the counteracting node. This implies that we can use the same operator as for conjunctively refined attacker's nodes in the first case and the same operator as for disjunctively refined attacker's nodes in the second case.

Pruning in general In order to perform the bottom-up algorithm, we do not only need to specify a question or a corresponding attribute domain, but we also need to provide a basic assignment. Since for attributes of Class 1 we are only interested in who the owner of a question is, it is not immediately apparent which values to assign to non-refined nodes of the other player, called the *non-owner*. There are two situations that we need to distinguish. If a non-refined node of the non-owner is countered, its assigned value should not influence the result of the computation. If a non-owner's node is not countered, its value should indicate that the owner does not have a chance to successfully perform this subscenario. Mathematically, it means that the value assigned to the non-refined nodes of the non-owner needs to be neutral with respect to one operator and simultaneously absorbing with respect to the other. Since, in general, such an element may not exist, we need to eliminate one of the described situations. We achieve this by removing all subtrees that do not lead to a successful scenario for the owner, with the help of the *pruning* procedure. This results in elimination of the absorption condition.

We illustrate pruning in the following example.

Example 4.23 Consider a slightly altered version of Example 2.2. Suppose that the defender is unaware of the options to "Install Video Cameras", and instead, he thinks about implementing a fingerprint scanner to have a key-less security system. The altered scenario is illustrated on the ADTree given in the left part of Figure 4.8. We are now interested in calculating the minimal **costs**. In this case, there is no need to consider the subtree rooted in "Steal Keys" because it is countered by the defense "Fingerprint Scanner" (abbreviated FS) and thus does not lead to a successful attack. The subtree rooted in "Steal Keys" should, therefore, be removed. This simultaneously eliminates having to provide values for the non-refined nodes "Steal Keys" and "Fingerprint Scanner". The computation of the minimal **costs** is then executed on the term corresponding to the tree in the right of Figure 4.8.

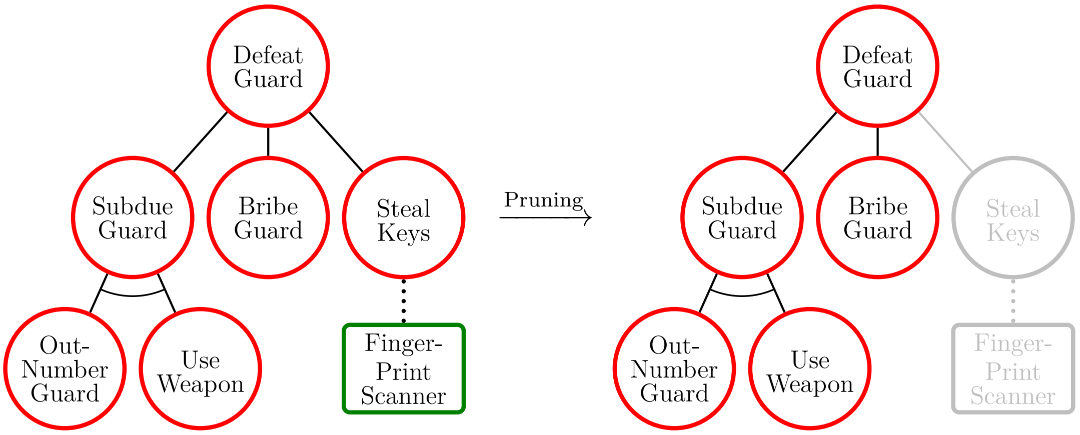


Figure 4.8: Pruning the modified "Defeat Guard" scenario for questions of Class 1 owned by the attacker.

Pruning graphically Let us consider a question of Class 1 and its owner. In order to graphically prune an ADTree, we perform the following procedure.

Definition 4.24 (Graphical pruning procedure) Starting from a leaf of the non-owner, we traverse the tree towards the root until we reach the first node v satisfying one of the following conditions:

- The node v is of the owner and sibling in a proper[7] disjunctive refinement (see Figure 4.9).

[7] A refinement is called *proper* if it contains at least two refining nodes.

- The node v is of the non-owner and sibling in a proper conjunctive refinement (see Figure 4.10).
- The node v is of the owner and counteracts a refined node of the non-owner (see Figure 4.11).
- The node v is the root of the ADTree (see Figure 4.12).

The subtree rooted in node v is removed from the ADTree. Starting from all leaves of the non-owner, this procedure is repeated until no further part of the tree can be eliminated.

We note that the order in which we perform the procedure does not influence the final result since removal of any prunable part does not change the values computed on the rest of the tree.

Also, in some cases the pruning procedure results in the removal of the entire ADTree. This is the case when the owner of the question does not have any way of successfully achieving his goal. In this case the computation of the attribute yields the default value for nodes of the non-owner.

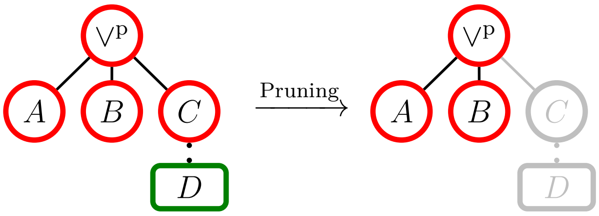


Figure 4.9: Pruning a proper disjunctive refinement.

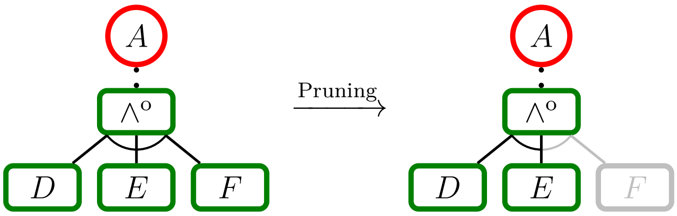


Figure 4.10: Pruning a proper conjunctive refinement.

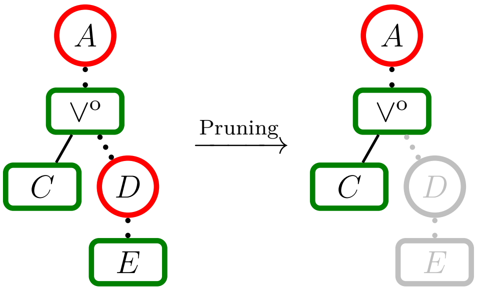


Figure 4.11: Pruning a countermeasure.

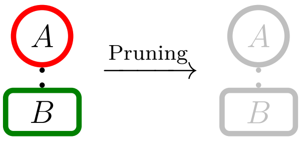


Figure 4.12: Pruning an entire ADTree.

Pruning formally Let Q be a question of Class 1 and let own denote the owner of Q. In order to model the pruning procedure in a mathematical way, we construct a formal model answering the question "Can the owner of Q succeed in a considered attack–defense scenario?" The idea is to assign the Boolean value `true` to all subtrees in which the owner of Q can succeed and the value `false` to the subtrees in which he cannot. Formally, we evaluate an attribute that we denote by $\mathrm{sat}_{\mathrm{own}}$, defined as follows. First, we set the basic assignment

$$\beta_{\mathrm{sat}_{\mathrm{own}}}(b) = \begin{cases} \texttt{true} & \text{if} \quad b \in \mathbb{B}^{\mathrm{own}}; \\ \texttt{false} & \text{if} \quad b \in \mathbb{B}^{\overline{\mathrm{own}}}. \end{cases} \tag{4.5}$$

Then, given an ADTerm t, we use the following attribute domain to derive the values of the attribute $\mathrm{sat}_{\mathrm{own}}$ at all subterms of t:

$$A_{\mathrm{sat}_{\mathrm{own}}} = \begin{cases} (\{\texttt{true}, \texttt{false}\}, \vee, \wedge, \wedge, \vee, \wedge, \vee) & \text{if} \quad \mathrm{own} = \mathrm{p}; \\ (\{\texttt{true}, \texttt{false}\}, \wedge, \vee, \vee, \wedge, \vee, \wedge) & \text{if} \quad \mathrm{own} = \mathrm{o}\,. \end{cases} \tag{4.6}$$

Note that the question "Can the owner of Q succeed in the scenario?" also falls into Class 1, as it is referring to a specific player. This explains why the corresponding attribute domain is constructed using only two different operators and conforms to the template of Class 1.

Remark 4.25 The attribute domain $A_{\mathrm{sat}_{\mathrm{own}}}$ is slightly different from the attribute domain

$$A_{\mathrm{sat}} = (\{\texttt{true}, \texttt{false}\}, \vee, \wedge, \vee, \wedge, \star, \star),$$

that was introduced in Section 4.2.1 to represent the **satisfiability** attribute. The latter attribute domain does not conform with the template of Class 1. By switching the operators and eliminating the negation contained in $\star$, it can, however, be transformed into the attribute domain $A_{\mathrm{sat}_{\mathrm{own}}}$. Such a transformation models changing the point of view from the perspective of the winner of a scenario to the perspective of a single player. In Section 4.5.5, we generalize this observation and argue that every attribute domain of Class 2 can be turned into an attribute domain of Class 1 such that the pruning procedure can be applied.

The following theorem shows that $\mathrm{sat}_{\mathrm{own}}$ models the pruning procedure soundly and correctly.

Theorem 4.26 *Consider a question Q of Class 1, its owner* own*, an ADTree T and the corresponding ADTerm t. Furthermore, let $A_{\mathrm{sat}_{\mathrm{own}}}$ and $\beta_{\mathrm{sat}_{\mathrm{own}}}$ be defined by Equations* (4.5) *and* (4.6)*. The graphical pruning procedure given in Definition 4.24 removes a subtree T' of T if and only if the evaluation of the* $\mathrm{sat}_{\mathrm{own}}$ *attribute on the corresponding subterm t' of t results in the value* `false`*.*

Proof. We first observe that, when a pruned subtree is removed, the attribute values in the rest of the tree do not change. It thus suffices to show that

1. if a subtree is removed by pruning, the evaluation of sat_{own} on the corresponding term results in `false`.
2. if a subtree is not removed by pruning, evaluation of sat_{own} on the corresponding term results in `true`.

Step 1: Let u be a leaf of the non-owner, from which we start the current step of the pruning procedure. We show that if a tree rooted in a node v is removed by pruning, then all subterms corresponding to subtrees rooted in the nodes on the path from u to v (including u and v) evaluate to `false`.

We prove this by contraposition. Assume that there exists a node w on the path between u and v, such that the term corresponding to the tree rooted in w evaluates to `true`. Moreover, let w be the first node with such a property which is encountered when starting from u. Note that $w \neq u$, because the basic assignment $\beta_{\text{sat}_{\text{own}}}$ assigns the value `false` to every non-refined node of the non-owner. This means that there exists a node w_1 which is a child of w lying on the path from u to v. According to our assumptions, the term corresponding to the tree rooted in w evaluates to `true` while the term corresponding to the subtree rooted in w_1 evaluates to `false`. This implies that operator $\vee$ has been used. According to the attribute domain given by Equation (4.6), there are only three situations where the logical disjunction is used:

- either w is a properly, disjunctively refined node of the owner,
- or w is a properly, conjunctively refined node of the non-owner,
- or w is a refined node of the non-owner and w_1 is its countermeasure.

It is now sufficient to realize that in all the three cases, the pruning procedure actually stops at node w_1. This is in contradiction to the fact that pruning stops at w.

Step 2: First, let us remark that the pruning procedure stops at node v if the value of the term corresponding to the tree rooted in the parent node of v is not uniquely determined by the value of the subterm corresponding to the tree rooted in v. This is because, in all three cases where pruning stops at v, the calculation of the sat_{own} attribute for the subterm corresponding to the tree rooted in the parent of v uses the operator $\vee$. It is applied to the value `false` (quantifying the term corresponding to the tree rooted in v) and another value which cannot be deduced from the currently considered path. The tree rooted in the parent of v will either be removed by the pruning procedure starting from another leaf of the non-owner or it will not be removed after all possible steps of the pruning are performed.

Let T be an ADTree and T' be its subtree which is not removed by any step of the pruning procedure. In the remaining part of this proof we show that the evaluation of sat_{own} on a term t' corresponding to T' results in value `true`. The proof is by induction on the structure of T'.

If T' is a leaf of T, then it needs to represent a basic action of the owner. This is because all leaves of the non-owner are removed by pruning. According to the basic assignment $\beta_{\mathrm{sat}_{\mathrm{own}}}$ the term t' is quantified with `true`.

Let us now consider a tree T' which has not been removed by any step of the pruning procedure and which is not a leaf of T. As induction hypothesis, we assume that the evaluation of $\mathrm{sat}_{\mathrm{own}}$ on all subterms corresponding to the subtrees of T' not removed by pruning results in `true`. This implies that the evaluation of $\mathrm{sat}_{\mathrm{own}}$ on t' yields `true` because the only possible ways of combining the values quantifying the subterms of the considered term are $\vee$ or $\wedge$. □

From a question to an attribute domain Next, we analyze how a question of Class 1 should be composed, in order to be able to instantiate the attribute domain template $A = (D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ with a specific value set and operators. To correctly instantiate A, we need a value domain D, two operators (for *all* and *at least one*) and we need to know which of those operators instantiates $\circ$ and which $\bullet$. Thus, a well-specified question of Class 1 contains exactly four parts, as illustrated by the following question:

Modality: What are the minimal
Notion: costs
Owner: of the proponent
Execution: assuming that all actions are executed one after another?

Each of the four parts has a specific purpose in determining the attribute domain.

Notion The notions of Class 1 that we identified during our study are:

- attack potential,
- attack time,
- consequence,
- costs,
- detectability,
- difficulty level,
- elapsed time,
- impact,
- insider required,
- mitigation success,
- outcome,
- penalty,
- profit,
- response time,
- severity,
- resources,
- skill level,
- special equipment needed,
- number of specialists,
- special skill needed,
- survivability.

From the notion we determine the value domain, e.g., $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$, Levels, Booleans, etc. We distinguish two general cases: finite notions, such as Levels, Booleans and infinite notions, such as $\mathbb{N}$, $\mathbb{R}$, $\mathbb{R}_{\geq 0}$. For numerous attributes both categories of notions are conceivable. It is, for example, up to the user to decide whether he wants to utilize cost levels, such as `Low`, `Medium` and `High` or real values.

The choice of the value domain influences the basic assignments, as well as the operators determined by the modality and the style of execution. The selected value domain needs to include all values that we want to use to quantify the owner's actions. It also must contain a neutral element with respect to $\circ$ if $\mathrm{own} = \mathrm{p}$ and with respect to $\bullet$ if $\mathrm{own} = \mathrm{o}$. This neutral element is assigned to all non-refined nodes of the non-owner, as argued in the beginning of this section.

The name of the notion is usually contained in the name of the attribute. In this thesis, if a notion is typeset in **bold**, it refers to the attribute and no longer the notion. Naturally, a full attribute specification needs other components. These may be irrelevant when no concrete attribute quantification is performed, but should never be omitted when actually quantifying values. Unfortunately, throughout the literature there is sometimes no clear distinction between the two notions resulting in confusion.

Modality The modality of a question clarifies how options are treated. Thus, it determines the characteristic of the *at least one* operator. Different notions are accompanied with different modalities. In the case of **costs**, interesting modalities include minimal, maximal and average.

Execution The question also needs to specify a style of execution. It describes the characteristics of the *all* operator, i.e., it determines what happens whenever all actions need to be executed. Exemplary styles of execution are: simultaneously vs. sequentially (for **time**) or with reuse vs. without reuse (for resources).

Owner The owner of a question determines how the modality and the execution are mapped to $\circ$ and $\bullet$. In case the owner of the question is the root player, i.e., the proponent, $\circ$ is instantiated with the *at least one* operator and $\bullet$ with the *all* operator. In case the root player is not the owner, the instantiations are reversed.

Given all four parts, we can construct the appropriate attribute domain. For infinite notions, possible combinations of the modality, the style of execution and the owner have been determined and are exemplified in Table 4.1 using the notion duration. When using finite notions, we need to ensure that the finite set is ordered. Given an ordered set, the functions are similar to the ones for infinite notions and are presented in Table 4.2.

To actually construct the attribute domain of a given attribute, we construct the template $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ from the algebraic structure $\langle D, \circ, \bullet \rangle$ provided by the tables. In case D does not contain the basic assignment for $\overline{\mathrm{own}}$, we extend D with the element $e_{\overline{\mathrm{own}}}$. The tables can also be used in case the notions do not model a duration or a skill level, as shown in the next example.

Example 4.27 In Example 4.6, we have seen that the question "What are the minimal costs of the proponent, assuming that reusing tools is infeasible?" can be answered using the attribute domain $A_{\mathrm{costs}} = (\mathbb{R}_\infty, \min, +, +, \min, +, \min)$. Alternatively, we can apply the newly explained concepts of this section to deduce the attribute domain as follows. The question's notion is *costs*, for which we want to use the value domain $\mathbb{R}$. Replacing the notion of *duration* with the notion of *costs*, we can make use of Table 4.1. From the question, we know that the modality is *minimum*, the owner is the *proponent* and the style of execution is *without reuse*, which corresponds to sequential execution. Hence, we use the semiring $\langle \mathbb{R}, \min, + \rangle$ and the basic assignment ∞ for $\overline{\mathrm{own}}$, as specified in Line 1 of Table 4.1. Executing the attribute domain construction, we see that Table 4.1 indeed can be used to find

	Notion	Modality	Owner	Execution	Structure $\langle D, \circ, \bullet\rangle$	Basic assignment for $\overline{\text{own}}$ ($e_{\overline{\text{own}}}$)
1	duration	min	p	sequential	$\langle \text{R}, \min, +\rangle$	∞
2	duration	avg	p	sequential	$\langle \text{R}, \text{avg}, +\rangle$	e_{avg}
3	duration	max	p	sequential	$\langle \text{R}, \max, +\rangle$	$-\infty$
4	duration	min	o	sequential	$\langle \text{R}, +, \min\rangle$	0
5	duration	avg	o	sequential	$\langle \text{R}, +, \text{avg}\rangle$	0
6	duration	max	o	sequential	$\langle \text{R}, +, \max\rangle$	0
7	duration	min	p	parallel	$\langle \text{R}, \min, \max\rangle$	∞
8	duration	avg	p	parallel	$\langle \text{R}, \text{avg}, \max\rangle$	e_{avg}
9	duration	max	p	parallel	$\langle \text{R}, \max, \max\rangle$	$-\infty$
10	duration	min	o	parallel	$\langle \text{R}, \max, \min\rangle$	$-\infty$
11	duration	avg	o	parallel	$\langle \text{R}, \max, \text{avg}\rangle$	$-\infty$
12	duration	max	o	parallel	$\langle \text{R}, \max, \max\rangle$	$-\infty$

Table 4.1: Determining instantiations of the structure from which to construct attribute domains of questions of Class 1 for infinite notions. In the table, e_{avg} denotes the neutral element with respect to avg. For infinite structures R is $\mathbb{N}$, $\mathbb{R}$, or $\mathbb{R}_{\geq 0}$.

the correct attribute domain. In order to finally answer the question on the ADTree in the left of Figure 4.8, we first prune it, as shown on the right of Figure 4.8. Then, the only basic actions that are left are "Outnumber Guard", "Use Weapon" and "Bribe Guard". Suppose the **costs** are 100€, 200€and 400€, respectively. We use these values as basic assignment β_{costs} and apply the bottom-up procedure introduced in Definition 4.4 to the ADTerm $\vee^{\text{p}}(\wedge^{\text{p}}(\text{OG}, \text{UW}), \text{BG})$:

$$\begin{aligned} & \text{costs}(\vee^{\text{p}}(\wedge^{\text{p}}(\text{OG}, \text{UW}), \text{BG})) \\ = \; & \vee^{\text{p}}_{\text{costs}}\,(\wedge^{\text{p}}_{\text{costs}}(\beta_{\text{costs}}(\text{OG}), \beta_{\text{costs}}(\text{UW}), \beta_{\text{costs}}(\text{BG}))) \\ = \; & \min\{+(100€, 200€), 400€\} \\ = \; & 300€. \end{aligned}$$

Remark 4.28 If the structure $\langle D, \circ, \bullet\rangle$ forms a semiring, it is not necessary to prune the ADTree to correctly answer a question Q of Class 1. This is due to the fact that in a semiring the neutral element[8] for the first operator is at the same time absorbing for the second operator. Such an element can then be assigned to all subtrees which do not yield a successful scenario for the owner of Q, in particular to the uncountered basic actions of the non-owner.

Other attributes in this class include the cheapest successful proponent bundle (**cspb**) attribute introduced in Section 4.2.3, and the attribute that models which goals of the proponent are executable in **less than** k time units, as introduced in Section 4.3.2.

Merging evaluation of attributes of Class 1 that are compatible with the multiset semantics with pruning Finally, we show how to combine the

[8] Such an element is usually called *zero* of the semiring. For instance, ∞ is the zero element of the semiring $\langle \mathbb{R}_{\infty}, \min, +\rangle$.

	Notion	Modality	Owner	Execution	Structure $\langle D, \circ, \bullet \rangle$	Basic assignment for $\overline{\text{own}}$ ($e_{\overline{\text{own}}}$)
1	skill level	min	p	sequential	$\langle \mathrm{K}, \min, +_k \rangle$	k
2	skill level	avg	p	sequential	$\langle \mathrm{K}, \mathrm{avg}, +_k \rangle$	e_{avg}
3	skill level	max	p	sequential	$\langle \mathrm{K}, \max, +_k \rangle$	0
4	skill level	min	o	sequential	$\langle \mathrm{K}, +_k, \min \rangle$	0
5	skill level	avg	o	sequential	$\langle \mathrm{K}, +_k, \mathrm{avg} \rangle$	0
6	skill level	max	o	sequential	$\langle \mathrm{K}, +_k, \max \rangle$	0
7	skill level	min	p	parallel	$\langle \mathrm{K}, \min, \max \rangle$	k
8	skill level	avg	p	parallel	$\langle \mathrm{K}, \mathrm{avg}, \max \rangle$	e_{avg}
9	skill level	max	p	parallel	$\langle \mathrm{K}, \max, \max \rangle$	0
10	skill level	min	o	parallel	$\langle \mathrm{K}, \max, \min \rangle$	0
11	skill level	avg	o	parallel	$\langle \mathrm{K}, \max, \mathrm{avg} \rangle$	0
12	skill level	max	o	parallel	$\langle \mathrm{K}, \max, \max \rangle$	0

Table 4.2: Determining instantiations of the structure from which to construct attribute domains of questions of Class 1 for finite notions. The set K is a finite ordered set of levels with maximum element k or the set $\{\texttt{true}, \texttt{false}\}$ represented as $\{0, 1\}$. The symbol $+_k$ denotes bounded addition over K, i.e., $+_k(b_1, \dots, b_n) = \min\{\sum_{i=1}^{n} b_i, k\}$ for $b_i \in \mathrm{K}$ and e_{avg} denotes the neutral element with respect to avg.

evaluation of an attribute of Class 1 that is compatible with the multiset semantics and the pruning operation in one procedure.

We have argued that, in order to evaluate an attribute α of Class 1 correctly, we first need to prune a considered ADTree with respect to the owner of the corresponding question. In this section, we show how the two procedures of attribute evaluation and pruning can be modeled using an extended attribute domain.

Consider a question Q of Class 1, the corresponding attribute domain $A_\alpha = (D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ and a basic assignment $\beta_\alpha \colon \mathbb{B} \to D$. For ease of presentation, in this section we assume that the owner of Q is the proponent, i.e., that $\circ$ is the *at least one* operator and $\bullet$ is the *all* operator. In order to be able to answer Q without the necessity of first pruning the ADTree, we extend D with an additional Boolean dimension that represents which actions are relevant for our considerations. Therefore, instead of the value domain D, we use the Cartesian product $D \times \{\texttt{true}, \texttt{false}\}$ denoted by $\widehat{D}$. Furthermore, we define $\widehat{\circ}$ and $\widehat{\bullet}$ as two internal operations on $\widehat{D}$, by

$$\widehat{\circ}((d_1, s_1), \dots, (d_k, s_k)) = (\circ(d_1 \otimes s_1, \dots, d_k \otimes s_k), \bigvee_{i=1}^{k} s_i),$$

and

$$\widehat{\bullet}((d_1, s_1), \dots, (d_k, s_k)) = (\bullet(d_1 \otimes s_1, \dots, d_k \otimes s_k), \bigwedge_{i=1}^{k} s_i),$$

where, for all $d \in D$, we set $d \otimes \texttt{true} = d$ and $d \otimes \texttt{false} = e_\circ$. Moreover, $e_\circ$ denotes the neutral element with respect to $\circ$, which is absorbing with respect to $\bullet$. It always exists, since A_α is constructed from an idempotent semiring, see Definition 4.20. In order to define the extended basic assignment $\widehat{\beta_\alpha} \colon \mathbb{B} \to D \times$

$\{\mathtt{true}, \mathtt{false}\}$, we set $\widehat{\beta_\alpha}(b) = (\beta_\alpha(b), \mathtt{true})$, for every basic action b of the owner of Q, and $\widehat{\beta_\alpha}(b) = (\beta_\alpha(b), \mathtt{false})$, for every basic action b of the non-owner of Q. The following theorem shows that the attribute domain defined by

$$\widehat{A_\alpha} = (\widehat{D}, \hat{\circ}, \hat{\bullet}, \hat{\bullet}, \hat{\circ}, \hat{\bullet}, \hat{\circ})$$

constitutes a formal model allowing us to correctly evaluate the attribute α and thus to answer Q, without requiring prior pruning. It can be argued that $\widehat{A_\alpha}$ is also associated to some question of Class 1 since it satisfies the corresponding template for attribute domains, as illustrated in Figure 4.7.

Theorem 4.29 *Let Q, A_α, β_α, $\widehat{A_\alpha}$ and $\widehat{\beta_\alpha}$ be defined as above. For every ADTerm t, we have*

- $\hat{\alpha}(t) = (d, \mathtt{false})$*, for some $d \in D$ if the tree corresponding to t is removed by the pruning procedure related to Q,*
- $\hat{\alpha}(t) = (\alpha(t), \mathtt{true})$ *if the tree corresponding to t is not removed by the pruning procedure related to Q.*

Proof. First, observe that the calculation of the second component of the pair $\hat{\alpha}(t)$ corresponds to the calculation of the attribute $\mathrm{sat}_{\mathrm{own}}$, formalized in Equations (4.5) and (4.6). Theorem 4.26 ensures that the second component of $\hat{\alpha}(t)$ is $\mathtt{false}$ if and only if the corresponding ADTerm is removed by the pruning procedure related to Q. In this case, it is not clear whether the first component of $\hat{\alpha}(t)$ has a conclusive meaning.

Let t be a term corresponding to a subtree which is not removed by pruning related to Q. In order to prove that $\hat{\alpha}(t) = (\alpha(t), \mathtt{true})$, it suffices to show that, according to Theorem 4.26, the evaluation of $\mathrm{sat}_{\mathrm{own}}$ on all subterms of t results in the value $\mathtt{true}$. In other words, that, when calculating $\hat{\alpha}(t)$ we perform operations of the form

$$\begin{aligned} \hat{\diamond}((d_1, \mathtt{true}), \ldots, (d_k, \mathtt{true})) &= (\diamond(d_1 \otimes \mathtt{true}, \ldots, d_k \otimes \mathtt{true}), \mathtt{true}) \\ &= (\diamond(d_1, \ldots, d_k), \mathtt{true}), \end{aligned}$$

where $\diamond \in \{\circ, \bullet\}$. This concludes the proof. □

We illustrate the use of the extended attribute domain introduced in this section on the following example.

Example 4.30 As in Example 4.27, we would like to answer the question "What are the minimal costs of the proponent, assuming that reusing tools is infeasible?", on the tree in the left part of Figure 4.8. From Example 4.27, we know that the corresponding attribute domain is $A_{\mathrm{costs}} = (\mathbb{R}_\infty, \min, +, +, \min, +, \min)$. We extend A_{costs} to the attribute domain $\widehat{A_{\mathrm{costs}}} = (\widehat{\mathbb{R}_\infty}, \widehat{\min}, \hat{+}, \hat{+}, \widehat{\min}, \hat{+}, \widehat{\min})$, as defined in this section. Since ∞ is the neutral element with respect to min on $\mathbb{R}$, the operation $\otimes$ is defined as $x \otimes \mathtt{false} = \infty$, for every $x \in \mathbb{R}$. We evaluate the minimal **costs** attribute on the ADTerm corresponding to the non-pruned ADTree from Figure 4.8, as follows:

$$\begin{aligned}
&\widehat{\mathrm{costs}}(\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{OG},\mathrm{UW}),\mathrm{BG},\mathrm{c}^{\mathrm{p}}(\mathrm{SK},\mathrm{FS})))\\
&=\widehat{\min}(\widehat{+}(\widehat{\beta_{\mathrm{costs}}}(\mathrm{OG}),\widehat{\beta_{\mathrm{costs}}}(\mathrm{UW})),\widehat{\beta_{\mathrm{costs}}}(\mathrm{BG}),\widehat{+}(\widehat{\beta_{\mathrm{costs}}}(\mathrm{SK}),\widehat{\beta_{\mathrm{costs}}}(\mathrm{FS})))\\
&=\widehat{\min}(\widehat{+}((\beta_{\mathrm{costs}}(\mathrm{OG}),\mathtt{true}),(\beta_{\mathrm{costs}}(\mathrm{UW}),\mathtt{true})),(\beta_{\mathrm{costs}}(\mathrm{BG}),\mathtt{true}),\\
&\qquad\widehat{+}((\beta_{\mathrm{costs}}(\mathrm{SK}),\mathtt{true}),(\beta_{\mathrm{costs}}(\mathrm{FS}),\mathtt{false})))\\
&=\widehat{\min}((+(\beta_{\mathrm{costs}}(\mathrm{OG}),\beta_{\mathrm{costs}}(\mathrm{UW})),\mathtt{true}),(\beta_{\mathrm{costs}}(\mathrm{BG}),\mathtt{true}),\\
&\qquad(+(\beta_{\mathrm{costs}}(\mathrm{SK}),\infty),\mathtt{false}))\\
&=\widehat{\min}((+(100\text{€},200\text{€}),\mathtt{true}),(400\text{€},\mathtt{true}),(\infty,\mathtt{false}))\\
&=\widehat{\min}((300\text{€},\mathtt{true}),(400\text{€},\mathtt{true}),(\infty,\mathtt{false}))\\
&=(\min\{300\text{€},400\text{€},\infty\},\mathtt{true})\\
&=(300\text{€},\mathtt{true}).
\end{aligned}$$

This result shows that the scenario is satisfiable for the proponent and that his minimal **costs** are 300€. It is the same result as the one obtained in Example 4.27.

4.4.3 Questions Where Answers for Both Players Can Be Deduced from Each Other

We illustrate the construction of the attribute domain of Class 2 using the question "What is the probability of success of a scenario, assuming that all actions are independent?" In case of questions of Class 2, values assigned to a subtree quantify the considered property from the point of view of the root player of the subtree. This means that, if a subtree that is rooted in an attack node is assigned the value 0.2, the corresponding *attack* is successful with probability 0.2. If a subtree that is rooted in a defense node is assigned the value 0.2, the corresponding *defensive* measure is successful with probability 0.2. Following the same line of reasoning, disjunctive (conjunctive) refinements for the proponent and the opponent have to be treated in the same way when considering questions of Class 2: in both cases, they refer to the *at least one* option for disjunctive refinements (here modeled with $\circ$) and the *all* options for conjunctive refinements (modeled with $\bullet$), of the player whose node is currently considered.

Questions of Class 2 also have the property that, given a value for one player, we can immediately deduce a corresponding value for the other player. For example, if the attacker succeeds with probability 0.2 the defender succeeds with probability 0.8. This property is modeled using a value domain with a predefined unary negation operation $\neg$. This negation allows us to express the operators for both countermeasures. We use the binary *all* operator where the second argument is negated. Hence, formally, $\bullet\neg(x,y) = x \bullet \neg y$. Therefore, the attribute domains of Class 2 follow the template $(D, \circ, \bullet, \circ, \bullet, \bullet\neg, \bullet\neg)$.

Below we discuss three parts that questions of Class 2 need to address.

Notion Questions of Class 2 refer to notions for which the value domain can be equipped with a unary negation operator. Given a value for a node of one player, this unary negation operator allows us to deduce the value corresponding to the other player. E.g., if the scenario is `true` for the attacker it is `false` for the defender. Identified notions for Class 2 are:

- satisfiability,
- probability of success,
- probability of occurrence,
- electricity.

Note that the allocation of the notion is not always unique. For example, in questions of Class 2, *electricity* would refer to the question whether or not electricity is needed. If we ask for the electricity requirements of the proponents, the notion could be placed in Class 1 and if we ask for the electricity requirement of both players, the notion could also be placed in Class 3.

Modality Modality specifies the operator for *at least one* option. For the notions enumerated above, this is either the logical OR ($\vee$) or the probabilistic addition of independent events $\oplus(p_{b_1}, \dots, p_{b_n}) = \prod_{i=1}^{n} p_{b_i}$ for given probabilities p_{b_i} for the events b_i.

Execution Finally, we need to know the style of execution, so that we can specify the operator for *all* options. In the above notions, this is either the logical AND ($\wedge$) or the probabilistic multiplication of independent events: $\otimes(p_{b_1}, \dots, p_{b_n}) = 1 - \prod_{i=1}^{n}(1 - p_{b_i})$.

In the two identified attribute domains the unary negation function is either the logical not ($\neg$) or the complement operator.

Example 4.31 We calculate the **probability of success** of the scenario given in the left of Figure 4.8, assuming that the success of all actions is independent. Let the probability of success of all basic actions be $\beta_{\text{suc}}(b) = p_b = 0.4$. We use the attribute domain

$$A_{\text{suc}} = ([0,1], \oplus, \otimes, \oplus, \otimes, \circledast, \circledast),$$

where $\circledast$ is defined as $\circledast(p_{a_1}, p_{a_2}) = p_{a_1} \cdot (1 - p_{a_2})$ to compute

$$\begin{aligned}
& \text{suc}(\vee^{\text{p}}(\wedge^{\text{p}}(\text{OG}, \text{UW}), \text{BG}, \text{c}^{\text{p}}(\text{SK}, \text{FS}))) \\
={}& \oplus(\otimes(\beta_{\text{suc}}(\text{OG}), \beta_{\text{suc}}(\text{UW})), \beta(\text{BG}), \circledast(\beta_{\text{suc}}(\text{SK}), \beta_{\text{suc}}(\text{FS}))) \\
={}& \oplus(\otimes(\beta_{\text{suc}}(\text{OG}), \beta_{\text{suc}}(\text{UW})), \beta(\text{BG}), \otimes(\beta_{\text{suc}}(\text{SK}), (1 - \beta_{\text{suc}}(\text{FS})))) \\
={}& \oplus(\otimes(0.4 \cdot 0.4), 0.4, (0.4(1 - 0.4))) \\
={}& \oplus(0.16, 0.4, 0.24) \\
={}& 0.61696.
\end{aligned}$$

In Section 4.2.1 we already discussed the **satisfiability** attribute which also follows the attribute template for attribute domains of Class 2. Moreover, in Section 4.2.4 we have seen the **mcw** attribute. From Remark 4.11 we can conclude that also this attribute is in Class 2.

4.4.4 Questions Relating to an Outside Third Party

Suppose an outsider is interested in the overall maximal power consumption of the scenario. As in the previous section, disjunctive (conjunctive) refinements of both players should be interpreted with the same operator. Indeed, for a third party the important information is whether *at least one* of the options or *all* options need to be executed and not who performs the actions. Countermeasures, contrary to countermeasures in Section 4.4.3, lose their opposing aspect. Instead, the arguments of the countermeasures are aggregated in the same way as conjunctive refinements. This is plausible since both the countered and the countering action contribute to the overall power consumption. These observations result in the following template for an attribute domain of Class 3: $(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$.

We specify relevant parts of the questions of Class 3 on the following example.

Modality: What is the maximal
Notion: energy consumption
Execution: when sharing power to execute different basic actions is impossible?

Notion In Class 3, we use notions that express universal properties covering both players. In the literature, we found the following examples:

- social costs,
- global costs,
- third party costs,
- environmental costs,
- information flow,
- environmental damage,
- combined execution time,
- required network traffic,
- energy consumption.

Modality The question should also contain enough information to allow us to specify how to deal with *at least one* option. In general, modalities used in questions of Class 3 are the same as those in Class 1, e.g., minimal, maximal and average.

Execution Finally, we need to know the style of execution, in order to define the correct operator for *all* options. The choices for the style of execution in Class 3 are again the same as in Class 1.

These three parts now straightforwardly define an algebraic structure $\langle D, \circ, \bullet \rangle$ that we use to construct the attribute domain $(D, \circ, \bullet, \circ, \bullet, \bullet, \bullet)$.

Example 4.32 Consider the question "What is the maximal energy consumption for the scenario depicted in the left of Figure 4.8 when sharing power to execute different basic actions is impossible?" The modality of the question is *maximum.* This implicitly assumes we only analyze minimally satisfying scenarios, i.e., that for disjunctive nodes each player chooses exactly one option and we disregard situations where more than once action is performed to satisfy disjunctive nodes. Both the proponent's as well as the opponent's actions may require energy. We assume that "Use Weapon" and "Fingerprint Scan" all consume 2kWh. The "Steal Keys" actions requires 1kWh, whereas "Outnumber Guard" and "Bribe Guard" do not require any energy. These numbers constitute the basic assignment for the considered attribute. From the question we know that when we have a choice, we should consider the option which consumes the most energy. Furthermore, since sharing power is impossible, values for actions which require execution of several subactions should be added. Thus, we use the attribute domain $A_{\mathrm{erg}_{\max}} = (\mathbb{R}, \max, +, \max, +, +, +)$ and compute the maximal possible energy consumption in the scenario as

$$\begin{aligned} &\mathrm{erg}_{\max}(\vee^{\mathrm{p}}(\wedge^{\mathrm{p}}(\mathrm{OG}, \mathrm{UW}), \mathrm{BG}, \mathrm{c}^{\mathrm{p}}(\mathrm{SK}, \mathrm{FS}))) \\ = \ &\max\{+(0\mathrm{kWh}, 2\mathrm{kWh}), 0\mathrm{kWh}, +(1\mathrm{kWh}, 2\mathrm{kWh})\} \\ = \ &3\mathrm{kWh}. \end{aligned}$$

Due to similarities for modality and the style of execution for questions of Class 1 and Class 3, we can use Table 4.1, to find the structure $\langle D, \circ, \bullet\rangle$ that determines an attribute domain for a question of Class 3. The table corresponds to the case where the owner is the proponent.

4.5 Constructing New Attributes and Attribute Domains

We conclude our discussion of attributes for ADTrees by giving an outlook on how to generally construct new attribute domains from previously known attributes. Rather than providing a general theory, we provide pointers for further research. We detail several examples that we hope are useful as a starting point for possible generalizations.

4.5.1 Cartesian Composition

A trivial way of creating a new attribute domain from given ones is to take the Cartesian product of the value domains and extend the operators componentwise. We call this construction the *Cartesian composition* of two attribute domains. It is formally defined as follows. Let

$$A_{\alpha_1} = (D_{\alpha_1}, \vee^{\mathrm{p}}_{\alpha_1}, \wedge^{\mathrm{p}}_{\alpha_1}, \vee^{\mathrm{o}}_{\alpha_1}, \wedge^{\mathrm{o}}_{\alpha_1}, \mathrm{c}^{\mathrm{p}}_{\alpha_1}, \mathrm{c}^{\mathrm{o}}_{\alpha_1})$$

and

$$A_{\alpha_2} = (D_{\alpha_2}, \vee^{\mathrm{p}}_{\alpha_2}, \wedge^{\mathrm{p}}_{\alpha_2}, \vee^{\mathrm{o}}_{\alpha_2}, \wedge^{\mathrm{o}}_{\alpha_2}, \mathrm{c}^{\mathrm{p}}_{\alpha_2}, \mathrm{c}^{\mathrm{o}}_{\alpha_2})$$

be two attribute domains of Class i, where $i \in \{1, 2, 3\}$. Then

$$A_{\alpha} = (D_{\alpha_1} \times D_{\alpha_2}, (\vee^{\mathrm{p}}_{\alpha_1}, \vee^{\mathrm{p}}_{\alpha_2}), (\wedge^{\mathrm{p}}_{\alpha_1}, \wedge^{\mathrm{p}}_{\alpha_2}), (\vee^{\mathrm{o}}_{\alpha_1}, \vee^{\mathrm{o}}_{\alpha_2}), (\wedge^{\mathrm{o}}_{\alpha_1}, \wedge^{\mathrm{o}}_{\alpha_2}), (\mathrm{c}^{\mathrm{p}}_{\alpha_1}, \mathrm{c}^{\mathrm{p}}_{\alpha_2}), (\mathrm{c}^{\mathrm{o}}_{\alpha_1}, \mathrm{c}^{\mathrm{o}}_{\alpha_2}))$$

is also an attribute domain of Class i. Naturally, the Cartesian composition generalizes to a combination of more than two attribute domains. The Cartesian composition is also applicable when combining attribute domains from different classes. However, then the resulting attribute domain cannot always be placed into one of the classes since the structure of the attribute domain is not necessarily retained. In practice, we have not observed such compositions of mixed classes.

In our case studies we have made use of the Cartesian composition when defining *meta-attributes*, see Section 5.3.3. Meta-attributes are properties of attributes. Meta-attributes may be useful when associating attribute values since it is possible to quickly identify information from one attribute instead of having to retrieve it from several places. In general, any meta-attribute can be used in combination with any attribute. We illustrate the use of meta-attributes on the following example.

Example 4.33 Suppose an RFID security analyst and a person working in computer forensics are asked to assess the **probability of occurrence** of a denial of service attack. Both experts think that the likelihood is `Medium` (`M`). However, suppose the security analyst is confident about his guess while the forensics expert is not. Capturing this additional information could help us distinguish between the associated values and, therefore, more precisely represent the scenario. We use the meta-attribute **confidence** to indicate how certain a person is about an attribute value for a given node. Suppose the value domain of the **confidence** is $\{1, 2, 3\}$, where 1 symbolizes no confidence, 2 normal confidence and 3 extreme confidence. Given these values, the basic assignment of the security expert would be $(\mathtt{M}, 3)$ while the basic assignment of the forensics expert would be $(\mathtt{M}, 1)$.

Another meta-attribute is **coverage**, which expresses the number of people who have associated an attribute value with a given node.

4.5.2 Dependent Composition

The Cartesian composition can be generalized by allowing more flexibility in the functional operators of the attribute domains. We create *dependently composed* attributes by taking again the Cartesian product of the value domains and employing new (more involved) functions as functional operators. In Section 4.2, we have already used several dependently composed attribute domains. More specifically, Sections 4.2.3 and 4.2.4 provide two detailed examples while Theorem 4.29 uses a dependently composed attribute to express which nodes are pruned.

As in the case for Cartesian compositions, it is possible to extend the dependent composition to compositions of more than two attributes. For example, it could be beneficial to extend Theorem 4.29 to compose multiple binary attribute domains with a non-binary attribute.

The benefit of a general theory of composition, however, is an open field of investigation. Since most compositions appear to be highly application specific, we conclude this section by highlighting three compositions that we have used in practice.

4.5.3 Derived Attributes

If we lift the restriction that the value domain is the Cartesian product of existing value domains, we can use an arithmetic formula to derive a new attribute by combining several existing attributes. These attributes are called *derived* attributes. Numerous examples where an arithmetic formula is used to derive values on attack trees can be found in the literature. Edge et al. [EDRM06] have defined a variant of a **risk** attribute for attack trees based on the formula **risk** = (**probability**/**costs**) · **impact**. A similar approach has been used in [JW08], where the **costs**, **success probability**, **profit** (there called gain) and **penalty** attributes have been combined in order to define a new attribute called the exact expected **outcome** of the attacker. Henniger et al. [HAF+09] combine the attributes **elapsed time**, **expertise**, **knowledge of system**, **window of opportunity** and **required equipment**, in order to deduce the **required attack potential**. Finally, Fung et al. [FCW+05] show how the **difficulty** level associated with the non-refined nodes can be used to estimate the **survivability** in the root node.

Instead of using an arithmetic formula to derive new values, we can directly incorporate the computation of the values into the standard bottom-up algorithm. We illustrate this general procedure on the following example.

Example 4.34 Suppose we want to compute the risk associated to nodes of an ADTree. Suppose further that the risk is computed using the formula:

$$\textbf{risk} = \frac{\textbf{probability} \cdot \textbf{impact}}{\textbf{costs}}. \tag{4.7}$$

Assume we are given the attribute domains A_{occ}, A_{costs} and A_{imp} as well as the basic assignment β_{occ}, β_{costs} and β_{imp} for a **probability of occurrence** attribute, a **costs** attribute and an **impact** attribute, respectively. Then the attribute domain A_{risk} is a derived attribute domain that includes the **probability**, the **costs** and the **impact** attributes as components. The attribute domain of the **risk** attribute is given by

$$A_{\text{risk}} = (D_{\text{risk}}, f_1, f_2, f_3, f_4, f_5, f_6),$$

where $D_{\text{risk}} = D_{\text{occ}} \times D_{\text{costs}} \times D_{\text{imp}} \times D'_{\text{risk}}$ and D'_{risk} is the value domain of the original risk formula given in Equation (4.7). Moreover, each f_i is a function from D_{risk} to D_{risk}. The first three components of each function f_i correspond to the functions of the original attributes **probability of occurrence**, **costs** and **impact**. The last component is given by the right part of Equation (4.7). However, since the values have to be computed from the values of the children, the risk value is expressed in terms of the functions used in the first three components. We exemplify the construction on f_1:

$$f_1 \colon D_{\text{risk}} \to D_{\text{risk}}$$

$$f_1((x_1, x_2, x_3, x_4)) = (f_{\text{occ}_1}(x_1), f_{\text{costs}_1}(x_2), f_{\text{imp}_1}(x_3), \frac{f_{\text{occ}_1}(x_1) \cdot f_{\text{imp}_1}(x_3)}{f_{\text{costs}_1}(x_2)}),$$

where f_{occ_1}, f_{costs_1} and f_{imp_1} denote the first functions in the attribute domains of the corresponding attributes and x_i for $i \in \{1, \ldots 4\}$ denotes a vector containing the values of the children of a considered node.

Depending on the circumstances, simply applying Equation (4.7) for every node after the other attribute values have been calculated may be more natural when computing the last component of the derived attribute. However, the previous example generalizes to other derived attributes that can be written in form of Equation (4.7). Hence the ADTree formalism unifies other formalisms since it can express numerous other attributes that occur in the literature.

4.5.4 Switching the Owner of the Question

While considering questions of Class 1 in Section 4.4.2, we have seen that it is necessary to specify the owner of a question to unambiguously determine the corresponding attribute domain. We revisit the impact of the owner of a question in the following.

Suppose we are given the attribute domain $A_{\mathrm{p}} = (D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ that allows us to compute the minimal **costs** of the proponent, given all defenses are in place. Recall that the owner of the question is the proponent. Now assume that we are instead interested in computing the minimal **costs** of the opponent. The attribute domain $A_{\mathrm{o}} = (D, \bullet, \circ, \circ, \bullet, \circ, \bullet)$ can be used to answer this question. We call this transformation of questions (and corresponding attribute domains) *switching the owner and the non-owner of the question.*

In general, to deduce the attribute domain corresponding to a question with a switched owner, we merely switch the operators $\bullet$ and $\circ$. The reason behind switching the operators is that the owner (or the non-owner) determines whether to assign $\bullet$ to the modality and $\circ$ to the execution, or vice versa.

Note that the switched attribute domain is, in general, no longer compatible with the multiset semantics: Whereas before we used the algebraic structure $\langle D, \circ, \bullet \rangle$ to construct the attribute domain of Class 1, we now use the algebraic structure $\langle D, \bullet, \circ \rangle$. If the initially considered attribute of Class 1 is compatible with the multiset semantics, we know that $\langle D, \circ, \bullet \rangle$ is an idempotent semiring. After switching, this does not hold in general, i.e., $\langle D, \bullet, \circ \rangle$ is, in general, not a semiring. If $\langle D, \circ, \bullet \rangle$, however, were a De Morgan lattice, then $\langle D, \bullet, \circ \rangle$ would also be a De Morgan lattice and both attribute domains would be compatible with the De Morgan semantics and the multiset semantics.

4.5.5 Switching the Perspectives

Finally, we analyze the relation between attribute domains of Class 1 and Class 2. We exemplify the connection with the help of **satisfiability** attributes.

Example 4.35 One way to ask about the realizability of a scenario is to ask: “Is the attack feasible for the proponent?” Alternatively, we could contemplate “Who is the winner of the scenario?” When computing a value with the bottom-up approach, the former formulation suggests that we already know the type of the root node while the latter allows us to answer the question recursively for every subtree.

We observe that the second formulation of the question can only be generalized to other attributes if the considered attribute can be seen from both the proponent’s as well as the opponent’s side and the proponent’s value and the opponent’s value influence each other. This is true in the case of the **satisfiability** attribute because of the following fact: If the proponent can win the scenario when considering only a specific subtree, it is obvious that the opponent cannot.

For the **costs** attribute, for example, such a relation does not hold since the **costs** of one player do not directly influence the costs of the other player.

The example generalizes to questions which ask for a winner or the current level (Questions corresponding to Class 2) but does not generalize to questions that refer to the type of the root node (Questions corresponding to Class 1). The fact that there are two different formulations of questions of Class 2 indicates that questions of Class 2 can be turned into questions of Class 1 but that the converse does not hold.

Mathematically the reasoning is as follows. From Figure 4.7, we know that the template for attribute domains of Class 2 is given by $(D, \circ, \bullet, \circ, \bullet, \bullet\neg, \bullet\neg)$ and the basic assignment is given as `true` for all nodes that can be realized by the player corresponding to the type of the node and as `false`, otherwise. We also know that the template for attribute domains of Class 1 is given by $(D, \circ, \bullet, \bullet, \circ, \bullet, \circ)$ and the corresponding basic assignment is given as `true` if the owner of the question can realize the basic assignment and as `false`, otherwise. Therefore, to construct the domain of attributes of Class 1, it suffices to know the basic assignment and the algebraic structure $\langle D, \circ, \bullet \rangle$ while for domains of attributes of Class 2 it is also necessary to know the negation function. Hence, formally, given an attribute domain of an attribute of Class 2 it is always possible to construct the corresponding attribute domain of Class 1, but (without additional knowledge) it is impossible the other way around.

5

Practical Applications of ADTrees

Chapters 2, 3 and 4 present the foundations of the ADTree methodology. Throughout the development process of the ADTree formalism, we have conducted numerous case studies. These case studies served a theoretical and a practical purpose. On the one hand, it provided valuable feedback on desired and necessary formal improvements to ensure that the framework is, in fact, theoretically sound. On the other hand, it verified that the approach is intuitive, practically applicable and that *use cases* performed according to the ADTree methodology actually yield meaningful results.

This chapter describes the case studies that were conducted to evaluate and possibly improve the ADTree methodology and is structured as follows. First, Section 5.1 describes which kind of case studies are feasible and useful in the context of the ADTree methodology and introduces the questions we focused on while performing the case studies. The two following sections describe three case studies that we performed. In detail, we describe a case study on an online auction protocol in Section 5.2 and two subsequent case studies on an RFID goods management system in Section 5.3. In Section 5.4, we collect the observations that we made during the case studies and use them to answer the initial questions. Then, in Section 5.5, we summarize our insights from the case studies and present them as guidelines that should be respected when performing use cases that follow the ADTree methodology. Finally, in Section 5.6, the ADTool is introduced. This software tool was developed taking the insights gained during the case studies into account. It enables the user to construct and analyze ADTrees with the help of computer support.

5.1 Selecting Suitable Case Studies and Research Aspects

Before commencing a case study with the aim of evaluating and improving a methodology, it is important to clarify the research strategy behind the experiments. In case studies on the ADTree methodology, we may strive to

- evaluate and improve the entire ADTree methodology,
- evaluate and improve certain aspects of the ADTree methodology,
- evaluate how the ADTree methodology can be applied in certain areas, for example in risk management or penetration testing,
- evaluate how the ADTree methodology can be incorporated in existing methods, like SQUARE [MS05], STRIDE [HLOS06] or other risk management

tools,

- build a library of ADTrees based on scenarios,
- build a library of ADTrees based on known attacks or vulnerabilities templates,
- find and evaluate suitable application domains for ADTrees.

In our case studies, we opted to primarily evaluate *certain aspects* of the approach. More precisely, we first focused on the syntax and the semantics of ADTrees, before we analyzed how to quantitatively assess with the help of the formalism. This advances the methodology and notation in a focused way. Since some of the above goals are interconnected, though, we partially address them as well. For example, by performing numerous case studies with varying scenarios, we simultaneously started a library of ADTrees, which is accessible at http://satoss.uni.lu/projects/atrees/library.php. During the ADTree creation, we made use of known attack pattern or vulnerability libraries such as the Common Attack Pattern Enumeration and Classification (CAPEC) http://capec.mitre.org/, the Common Weakness Enumeration (CWE) http://cwe.mitre.org/, the Common Vulnerability Scoring System (CVSS) http://www.first.org/cvss or the Common Vulnerabilities and Exposures (CVE) http://cve.mitre.org/.

Since we used a wide range of suitable scenarios, we were able to confirm that the ADTree methodology is capable of modeling a multitude of scenarios. For example, we combined the analysis of technical, socio-technical, social as well as physical security on numerous layers of abstraction. Other possible scenarios range from the evaluation of web services and communication technologies, over analyzing OS-level and network attacks, scrutinizing attacks on banks, warehouses or online businesses to purely physical attacks, such as opening a bank vault or locked doors. This wide range of scenarios also allowed us to test the hypothesis that the ADTree methodology is general. In fact, the approach does not distinguish between high-level and low-level attacks or, e.g., OS-level and network-based denial of service (DoS) attacks, so that they can all be combined in the same model.

As we mentioned previously, our case studies targeted the evaluation of specific aspects of the ADTree methodology. In detail, we focused on the following questions relating to the ADTree methodology's syntax, semantics and attributes:

Q1) How can we use ADTrees to find out which defensive measures are present in a system? And which techniques allow us to construct a precise model?

Q2) Are semantics preserving transformations on ADTrees a useful tool? Do they help us to improve our understanding of a scenario?

Q3) Should there be any restrictions when setting up a tree model, for example, should the labels of the nodes have a specific form or should the tree have a predefined depth?

Q4) What has to be considered when providing attribute values for nodes?

Q5) What are suitable methods to turn the values provided for the nodes into a basic assignment?

Q6) What are the strengths and weaknesses of the bottom-up algorithm?

Before we actually answer the above questions in Section 5.4, we present a selection of case studies that we performed during the development of the ADTree methodology. All case studies were performed by following the steps suggested when conducting use cases with the ADTree methodology. At the start of every use case, an ADTree model is set up which possibly requires several rounds of improvements and tweaks. In this phase, the ADTree structure, the attributes and the people who provide attributes should be determined. Since any model rests on these three interconnected pillars, any later adjustment is not advisable because this might invalidate the actual values. After establishing the ADTree, values for attributes can be estimated. In a next step, these values are then turned into a basic assignment, before the bottom-up algorithm is applied in a final step. The work steps to perform a use case according to the ADTree methodology are summarized in Figure 5.1.

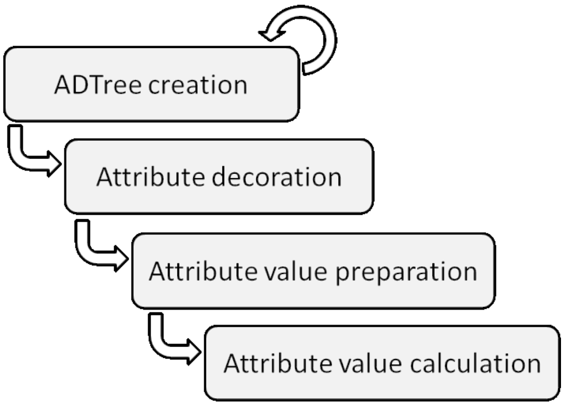


Figure 5.1: High-level overview over the steps in use cases according to the ADTree methodology with attribute evaluation.

The first case study, presented in Section 5.2, was performed on an online auction scenario, mainly targeting the analysis of syntax and semantics, i.e., Questions Q1–Q3. Since this case study was not concerned with attributes, it only illustrates the first step in the ADTree methodology. Two other case studies, presented in Section 5.3, were performed on an RFID goods management system. One case study focused on attribute decoration, while the other one examined the entire methodology. They both targeted quantitative aspects of the formalisms, i.e., Questions Q4–Q6. Conclusions that can be drawn and the implications on future use cases are presented subsequently in Section 5.4.

5.2 Initial Case Study

For our initial case study, we selected to model a specific online auction fraud scenario that had previously been used by the trust in digital life (TDL) research community [TDL13]. The scenario was provided as a textual description and details 10 predefined steps which Otto needs to perform in order to accomplish his goal. We tried to capture the given description as accurately as possible by extracting the most pertinent information to create an ADTree, displayed in Figure 5.2.

Our goal during the case study was to evaluate the ADTree methodology and not to analyze the scenario. Therefore, we focused our attention on the syntax and the semantics of an early version of the ADTree methodology. We present our observations together with observations from subsequent case studies in Section 5.4.

The root of the tree depicts how a fictitious agent called Otto can successfully perform an online auction fraud. In order to do that, he has to anonymously acquire goods that he pays for with someone else's money. Otto achieves this by accomplishing 10 subgoals. To better visualize the scenario, some of these goals are grouped, others refined. For the initial grouping we use the following four subgoals, represented as the four children of the conjunctive root node "Auction Fraud". Otto has to prepare his fraud, purchase the goods from Alice, get them delivered and sell them to Perry. Some of these subgoals can again be refined. As preparation, Otto has to "Ask Vera about Sam", "Shop at Sam's" and "Send Emails" to find victims (conjunctive node "Preparation Phase" with three children). When Otto shops at Sam's, he acquires malware, a list of emails and a stolen credit card number (conjunctive node "Shop at Sam's" with three children). To find victims, he sends two kinds of emails. The first kind contains a "Job Offer", the second kind contains malware (conjunctive node "Send Emails" with two children). Both kinds of emails are sent to the list of email addresses Otto acquired from Sam. Further, Rachel has to read the job advertisement email and accept the job, Tina has to read the email and open the attachment (two conjunctive nodes "Job Offer" and "Malware Distribution" with two children each). Otto also needs to buy goods in an online auction from Alice. For this, he needs to login into the online auction using Tina's credentials, purchase Max's goods and pay these with Chris's credit card (conjunctive node "Purchase from Alice" with three children). Otto organizes the delivery process by telling Max to ship to Rachel, Rachel to forward the goods to Dave and finally by going to Dave, picking them up (conjunctive node "Delivery Process" with three children). This concludes the construction of the part of the ADTree that consists of only attack nodes. It actually represents an attack tree that describes all necessary steps in Otto's fraud scenario.

The scenario was extended to include possible defensive measures that law enforcement could undertake to mitigate online auction fraud attempts. Subverting the verification provided by Vera, can counteract Otto's inquiry about Sam (countermeasure against "Ask Vera about Sam"). Similarly, policing the Internet prevents Otto from shopping at Sam's underground shop (countermeasure against "Shop at Sam's"). Training Internet users like Rachel and Tina counteracts their exploitation as re-shipper or unwilling password supplier (countermeasures against "Rachel Accepts Job" and "Tina Opens Attachment"). To counteract credential theft, an intelligent profiling of user credential usage can be set up by Alice, to mitigate fake credit usage, faster checks for credit card theft can be implemented (countermeasures against "Credentials from Tina" and "Pay with Chris's CC"). Preventing the successful delivery of the goods is also possible. While marking goods helps the police in tracking the origin of the goods, spot-checking drop boxes can prevent Otto from picking up the goods from Dave (countermeasures against "Max to Rachel" and "Otto from Dave"). In addition, either measure can interrupt the delivery process from "Rachel to Dave" (countermeasure against "Rachel to Dave" as disjunctive defense "Stop in Transit" with two children). Finally, the sale of the

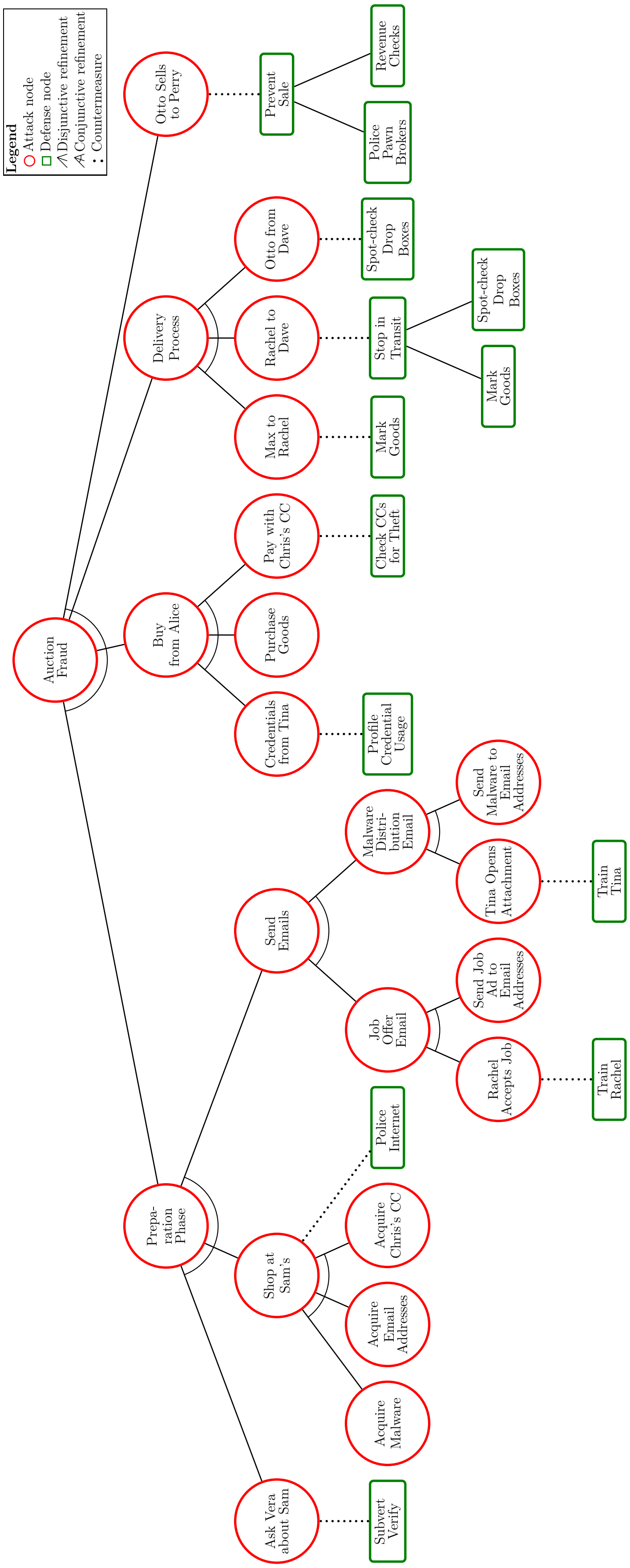


Figure 5.2: An ADTree for an online auction fraud scenario.

goods can be disrupted in two ways, either by policing pawn brokers or increased “Revenue Checks” (countermeasure “Prevent Sale” as disjunctive defense with two children).

5.3 Subsequent Case Studies

To examine how the ADTree formalism handles quantitative aspects related to scenarios, we have performed two case studies. One was an *external case study* which involved SINTEF, a research institution in Trondheim, Norway and TXT e-solutions, an Italy-based consulting company. The second case study was a *classroom experiment*. For both case studies, we used the same ADTree model, describing an RFID-based goods management system. The system had already been used in a threat assessment analysis as part of the FP7 project SHIELDS [SHI10b].

5.3.1 The RFID Goods Management System

In order to perform a case study on quantitative analysis, we selected a real-life scenario of an already deployed and operational system named the Warehouse Information Management System (WIMS) with special focus on one of its components, the *Warehouse Loading Docks Management Application* (WaLDo). This system manages all incoming and outgoing goods to and from a specific warehouse, keeping track of orders, goods location, picking lists, shipping notifications, etc. The warehouse is a highly automated environment where all goods can be electronically identified using RFID tags. Figure 5.3 gives a high-level overview of the system components. Arrows indicate interactions between components.

The WaLDo application controls all goods that cross the loading docks of the warehouse. The physical warehouse is equipped with RFID enabled loading docks. All RFID readers conform to the EPCGlobal specifications and are managed via an *Application Level Event* (ALE) service that provides a web service interface to upper layer applications like WaLDo. Additionally, the warehouse has an information management system able to interact with the company *Enterprise Resource Planning* (ERP) system, the integrated software application that manages the entire company information flow and resources, to process universal business language documents like *Picking Lists*, used to specify which material is to be shipped to whom, and *Advanced Shipping Notification* (ASN) documents, used to specify which goods are expected to be received.

In order to properly analyze potential threats, we also consider the environment in which the system operates. Figure 5.4 depicts the physical premises, the equipment and the workspaces inside the warehouse. The WaLDo application operates in a warehouse where eight employees are working. The size of the warehouse building is 500 m^2. It contains forklifts with an RFID reader, shelves for goods and three loading docks with RFID readers, which can only be opened from the inside. All goods pass in and out through the loading docks and are registered by the RFID readers. The building also has one room for computer servers, one administrative office, one security room containing two *Closed-Circuit Television* (CCTV) monitors and a fuse box, one bathroom, one corridor, a main entrance and an emergency

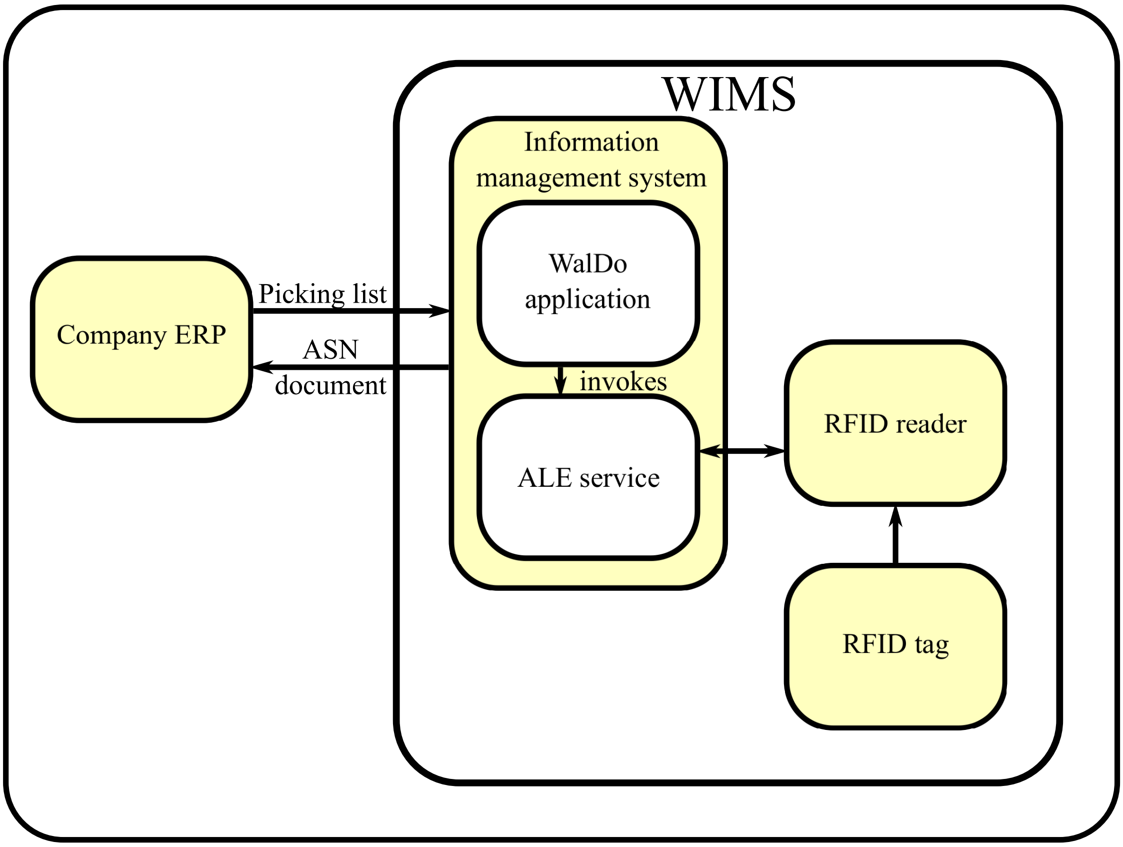


Figure 5.3: The WIMS deployment diagram.

exit. The warehouse is surrounded by a fence that encloses the entire area. The fence has two gates, one for trucks and one for employees, which can only be opened remotely from the security room. The area inside the fence has a parking space where trucks can wait before unloading their goods and one where the employees can park their cars. The warehouse is equipped with a high-speed Internet connection and a wired LAN Ethernet. The Ethernet network connects the servers with the RFID readers of the loading dock. In total, there are seven surveillance cameras that are linked to external security services, monitoring both the inside and the outside of the warehouse. Cameras 1, 5 and 6 monitor the shelves within the WaLDo building, Camera 2 monitors the main entrance gates, Camera 3 monitors the parking spaces, Camera 4 monitors the loading docks and Camera 7 monitors the warehouse's main door. Each day between 10 and 20 trucks deliver goods to or from the warehouse. The drivers load the goods on and off their trucks by accessing the warehouse through the docks. Though we do not specify what kind of goods are stored in the warehouse, we assume they are worth stealing.

5.3.2 The ADTree Model

The information given in the previous section serves as a basis to create an ADTree. Since the warehouse had been part of other unpublished case studies, we benefited from already existing attack trees and misuse case diagrams describing potential threats and countermeasures. We also made use of other relevant attack trees to support the creation of an ADTree, such as Mirowski et al. [MHW09] who

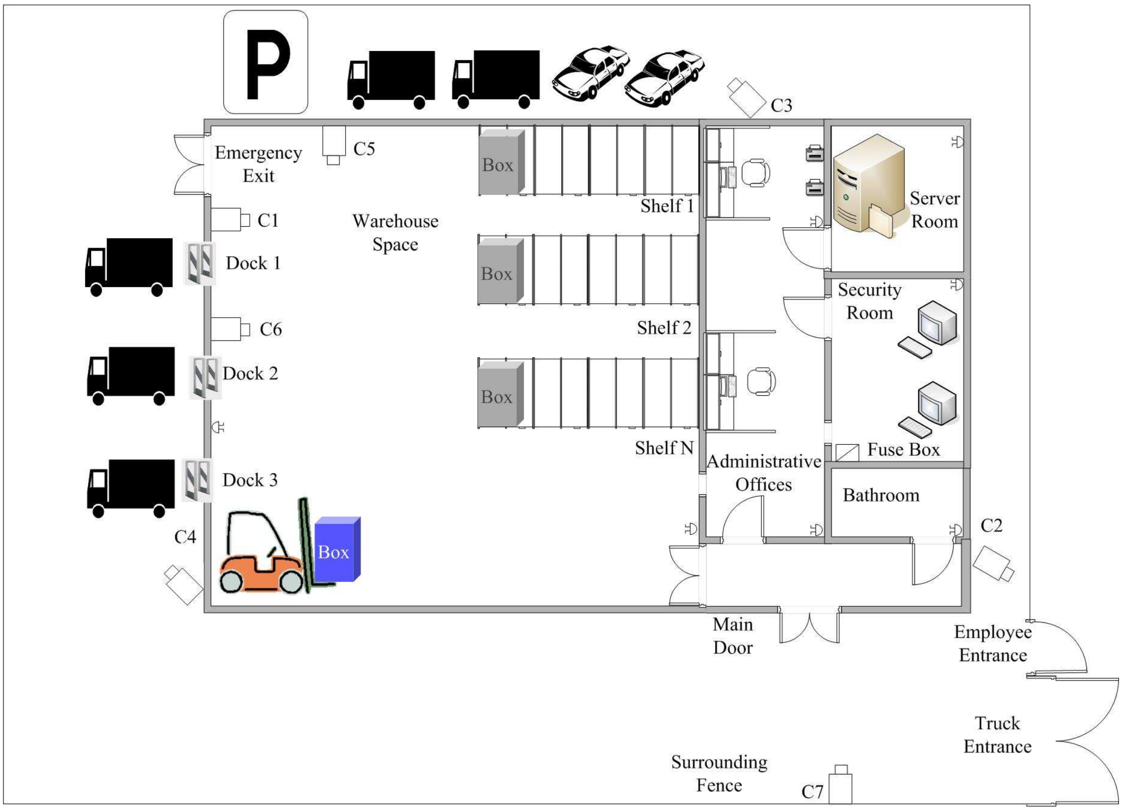


Figure 5.4: The floor plan of the warehouse and its surroundings.

describe generic RFID attacks. In order to extend their work and capture threats related to technical, physical and social engineering, we did a thorough analysis of the conceptual design of the RFID-based goods management system, the physical layout of the warehouse in which the system is deployed, see Figure 5.4, and how people are involved in the work processes.

The top goal node of the high-level tree model that we analyze, as shown in Figure 5.5, is called “RFID DoS Attack”. In order to achieve this goal, we specified six options. The attacker can remove the tag, disable the tag, overload the tag, disable the reader, disable the backend or block the communication between tags and readers. In the case study, we initially refined all children to construct a detailed ADTree model. However, to keep the model tractable, we chose to only focus on the nodes “Remove Tag” and “Block Communication” during the remainder of the case study. Their refinements are depicted in Figures 5.6 and 5.8. We deliberately opted to analyze an incomplete tree to reflect that, in most use cases, the modeling time is limited which inevitably leads to incomplete trees.

To physically remove the RFID tag an attacker can either remove the tag himself or he can convince someone else to remove the tag. In the first case, he either can infiltrate the building or he has to infiltrate the organization and thereby gain legitimate access. Infiltrating the building can be achieved by “Breaking and Entering”, as detailed in Figure 5.7, by posing as a truck driver, by executing a “Postal Trojan Attack” or by staging a “Visitor Attack”. A “Postal Trojan Attack” can be achieved when the attacker hides in a box of 1 SFU and this box is sent to the warehouse. The owner of the warehouse could defend against Trojan mail by employing a “Sniffer dog” that can detect humans in the incoming goods. The

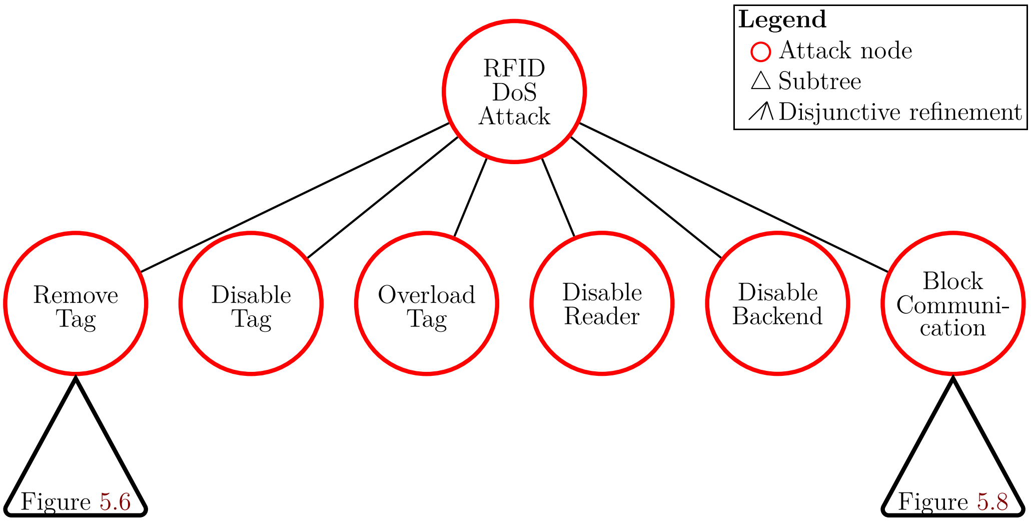


Figure 5.5: An ADTree for RFID DoS attack.

attacker, in turn, could confuse the dog using decoy rats or pepper spray. If the attacker decides to execute a “Visitor attack”, he can “Come as Visitor” during daytime and hide in the bathroom until everyone else has gone home. The defender could anticipate such an attack and “Track Visitors” on the warehouse premises. Tracking the visitors can be accomplished by escorting the visitors, by requiring visitors to register in a visitor’s log, by using an attended visitor’s log or by installing presence detectors on the premises. A visitor could choose to overcome the defense “Register in Visitor’s Log” by faking a log entry. In that case, the warehouse owner should switch to an attended visitor’s log. If the attacker decides that he wants to infiltrate the organization, he can try to “Get Hired as Warehouse Staff”, “Pose as Warehouse Employee”, or simply “Buy the Warehouse” (we deliberately added some extreme actions to the tree to try and provoke some extreme attribute values). The defender could protect himself against any infiltration by performing background checks on everyone he works with.

If the attacker chooses to convince someone else to remove the tag, he can “Bribe”, “Threaten”, “Blackmail” or “Trick” this person. In the first case, he has to identify a corruptible subject and then he has to actually bribe the subject. The warehouse owner could defend against bribery by thwarting the employees from receiving bribes by providing a mandatory “Security Awareness Training” or threatening to fire the employees in case of infringement. Provided the attacker wants to trick another person into removing the tag, he can either send false replacement tags or he can place a “False Management Order” to replace the tags. Fake orders can be initiated by infiltrating the management and ordering replacement tags. A defender can prevent this kind of trickery by mandatory security awareness training courses. Last, a defender could prevent any kind of removal of the RFID tag by using a stronger adhesive, i.e., attaching the tag in a way that it cannot be removed.

If an attacker decides to remove the tag himself by breaking and entering, he must get onto the premises and get into the warehouse undetected by the installed cameras. To get onto the premises, an attacker can climb over the fence or he can enter through the gate for employees undetected by Camera 2. To prevent attackers from climbing over the fence, the defender could install “Barbed Wire”

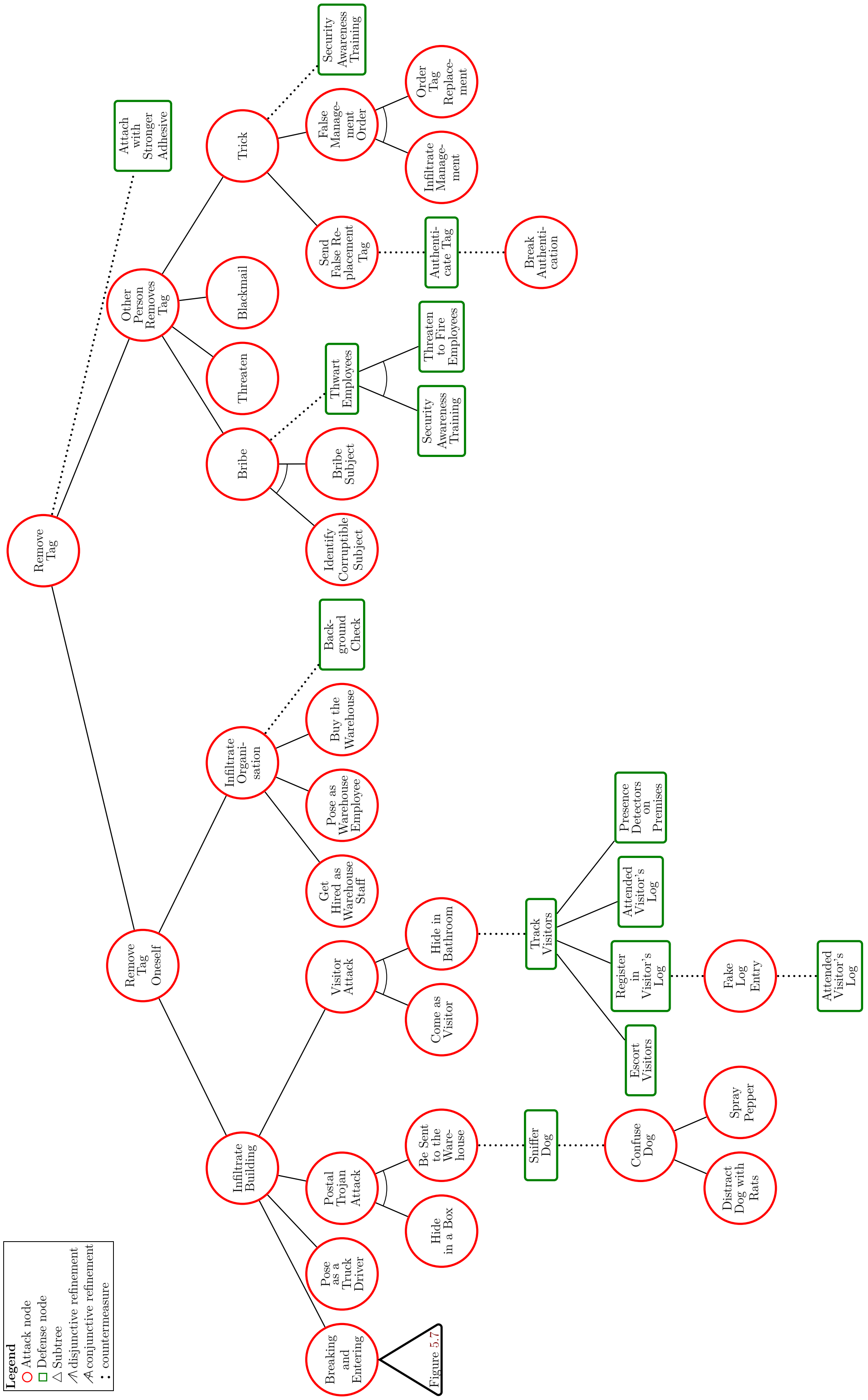


Figure 5.6: The “Remove Tag” subtree.

on the fence. An attacker, in turn, could circumvent the barbs by guarding against them, which he could achieve by either throwing a carpet over the barbs or by wearing protective clothing. The attacker also has to get into the warehouse. He can accomplish that by entering through the door undetected by Camera 7 or entering through the loading dock undetected by Camera 4. The defender could prevent an attacker from entering through the main door by monitoring the door with biometric sensors. Another defensive measure would be to install and monitor the premises with additional security cameras. These new cameras would monitor the parts of the property not yet covered, but could be rendered useless if an attacker disables them. Disabling could be done by shooting a strong laser at the cameras or by video looping the camera feed. Alternatively, guards patrolling the premises could protect against this kind of threat.

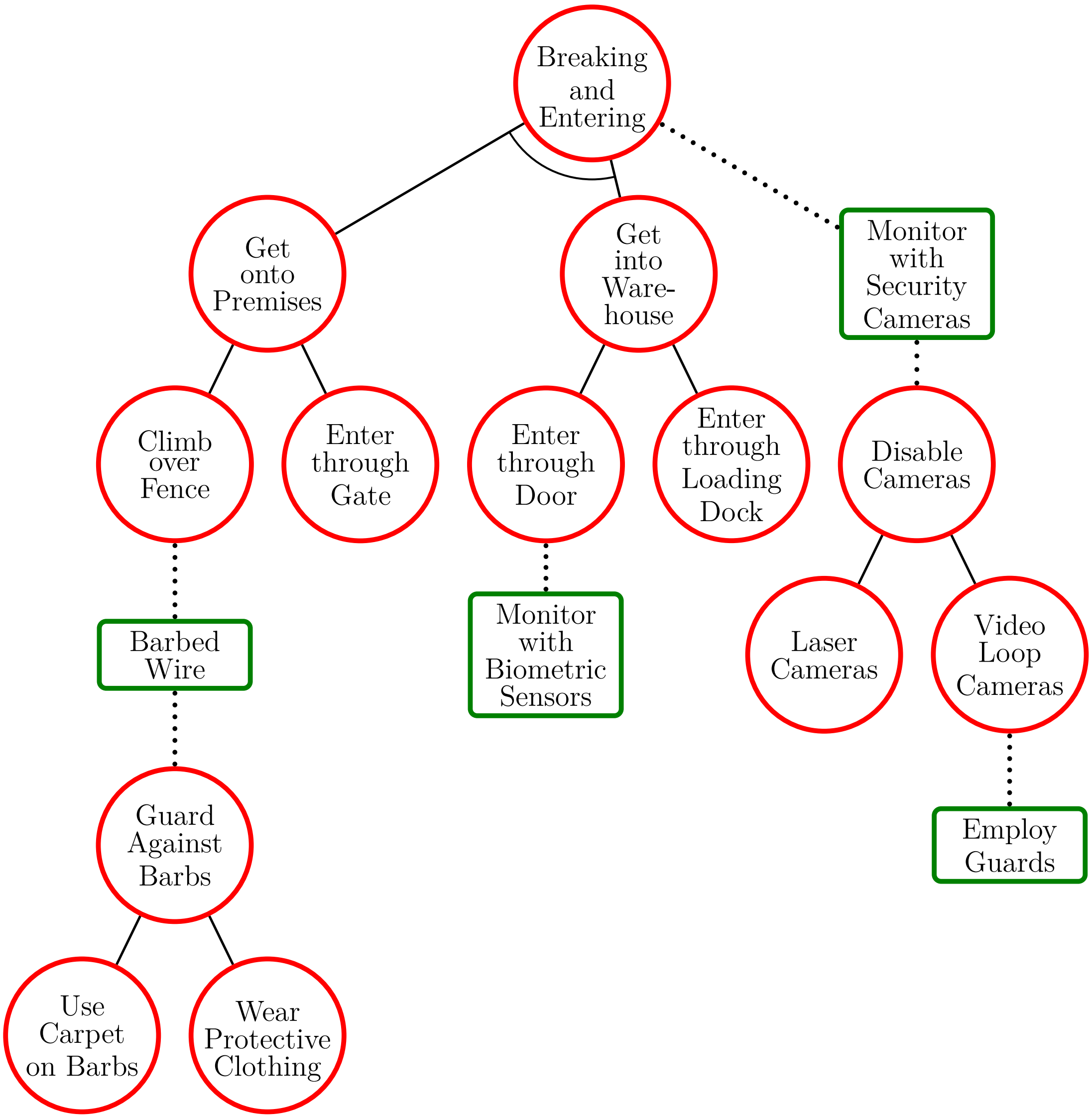


Figure 5.7: The “Breaking and Entering” subtree.

Blocking communication can be done by blocking the communication between the tag and the reader or by blocking communication between the reader and the backend, as depicted in Figure 5.8. To do the former, there exist several options: It is possible to shield the tag, use a malicious reader that constantly requests information from the tag and this way blocks the tag, use a different tag that blocks the reader or jam all signals. Shielding a tag can be achieved by being in the vicinity of the tag and by using a “Faraday Cage”. An obvious defense against attackers being in the vicinity of the tags would be to increase the amount of security personnel guarding the warehouse. A “Faraday Cage” can be installed around the reader or around the tag. To prevent attackers from jamming the signal, the defender could isolate the entire warehouse network, which could be achieved by securing the warehouse or encasing the entire warehouse inside a Faraday cage. If the attacker decides to block the communication between the reader and the backend, he can achieve it by evoking a DoS in the wired network.

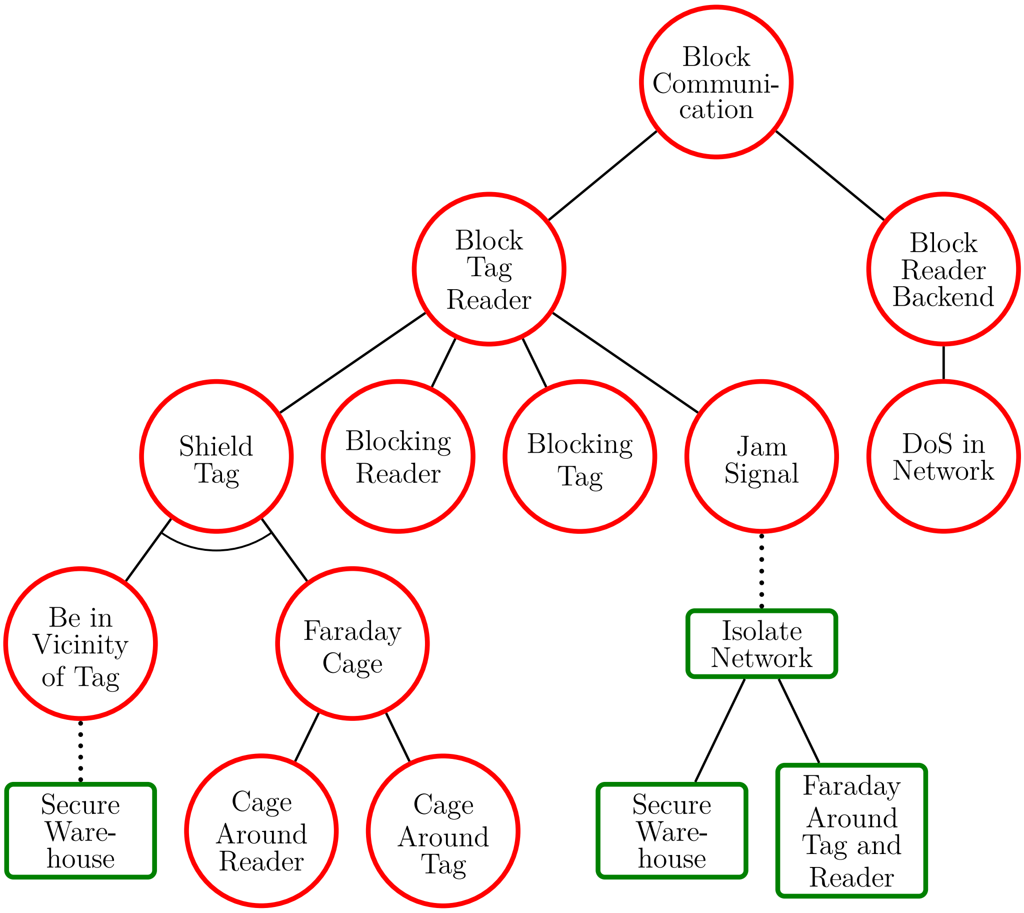


Figure 5.8: The “Block Communication” subtree.

5.3.3 Attribute Selection

For the external case study, we have experimented with a game-based approach for the decoration of the ADTree. We chose a set of nine attributes that we felt were useful. Detailed attribute descriptions, along with references treating these attributes and possible value domains are listed in the following:

Costs [**Sch99, BFM04, MO05, BLP$^+$06, EDRM06, Yag06, SDP08, ACK10, BP10, TA10, WWPP11, Ame12, RKT12a**] The amount of money needed to finance the attack or to defend against it (depending on whether it is the attacker’s or the defender's point of view). This refers to, for example, equipment or software costs, educational expenses, development costs or amount of a bribe.

Values: `Cheap` (`C`): Any attacker or defender can afford this without thinking twice. `Average` (`A`): The **costs** of the attack will fend off most attackers without a steady income. Defenders will typically do a cost-benefit analysis before the expenses can be justified. `Expensive` (`E`): The attacker will need substantial funding in order to perform the attack. It is unlikely a defender with insufficient funds will invest in this. Alternative value domain: Real numbers.

Detectability (Det) [**BFM04**, **TA10**, **Ame12**] The chance that the defender will notice the attack during its execution or the attacker will notice the defense mechanism.

Values: `Easy` (`E`): Any attacker or defender with a clear state of mind will detect that something was out of the ordinary right away. `Possible` (`P`): Some attackers or defenders are able to detect this defense or attack. `Difficult` (`D`): Few attackers or defenders are qualified to notice that something is wrong before the attack occurs.

Difficulty (Diff) [**Amo94**, **Sch99**, **BFM04**, **MO05**, **HAF+09**, **TA10**, **ACK10**, **WWPP11**, **Ame12**] The technical or social skill level needed for the attacker or defender to succeed.

Values: `Trivial` (`T`): Little technical or social skill required. `Moderate` (`M`): Average cyber hacking or defense skills required. `Difficult` (`D`): Demands a high degree of technical expertise, the attacker is a professional con artist. `Unlikely` (`U`): Beyond the known capability of today's best attackers or defenders.

Impact (Imp) [**Amo94**, **Sch99**, **Sch04b**, **MO05**, **EDRM06**, **SDP08**, **HAF+09**, **LLFH09**, **ACK10**, **TA10**, **WWPP11**, **RKT12a**] The severity or consequence from the system owner's point of view. Can refer to loss of money, but also other less tangible resources such as loss of reputation.

Values: `Low` (`L`): The system owner will not care or notice. `Moderate` (`M`): Acceptable but unwanted loss. `High` (`H`): Unacceptable loss, must be avoided. `Extreme` (`E`): Will terminate business. Alternative value domain: Real numbers expressing the expected loss.

Penalty (Pen) [**BLP+06**, **JW08**, **WWPP11**] The consequences for the attacker given that the attack fails, for instance, a fine, jail sentence or being blacklisted. Here, we do not consider any penalty of a successful attack.

Values: `Low` (`L`): The attacker will not care. `Medium` (`M`): The attacker will think twice before performing the attack. `High` (`H`): Few attackers will take the risk. Alternative value domain: Real numbers expressing a fine or years in prison.

Profit (Prof) [**Amo94**, **BDP06**, **JW08**, **RKT12a**] The economic profit or gain the attacker will receive, should the attack succeed. This value does not include **costs** of attack.

Values: `None` (`N`): A successful attack does not lead to any direct income. `Marginal` (`M`): Economic gain is not enough by itself to justify the attack. `Lucrative` (`L`): The attacker can obtain a substantial profit. Alternative value domain: Real numbers.

Probability of Success (Prob) [**Sch99**,**BFM04**,**BLP+06**,**EDRM06**,**Yag06**, **HAF+09**, **LLFH09**, **ACK10**, **MTF11**, **WWPP11**, **RKT12a**] The estimated chance that the attack or defense will succeed. Could be based on heuristics of similar attacks or simply best guesses.

Values: `Unlikely` (`U`): Below 5 %. `Low` (`L`): Between 5 % and 25 %. `Medium` (`M`): Between 25 % and 75 %. `High` (`H`): More than 75 %. `Certain` (`C`): Close to 100 %. Alternative value domain: Specific percentage values.

Special skill or property (Skill) [**Sch99**,**MO05**,**ACK10**] A specified skill or property the attacker or defender will need in order to succeed. This is orthogonal to the **difficulty** attribute. Examples are **insider knowledge** or the need for **electricity**.

Values: `True` (`T`): A special skill or property is required. `False` (`F`): No special skill or property is required.

Time [**Sch99**, **HAF+09**, **WWPP11**] For the attacker this is the time needed to perform the attack, independent of **difficulty** and **costs** of the attack. For the defender this is the time needed until the defense is effective.

Values: `Quick` (`Q`): The attack or defense can be performed in an instance. `Medium` (`M`): The attacker or defender will need to be patient and some time will pass. `Slow` (`S`): The attack or defense takes a long time to complete. Alternative value domain: Real numbers in terms of minutes.

We also decided to make use of the meta-attribute **confidence** (Section 4.5.1), with the value domain $\{1,\ldots,5\}$, where 1 represents total lack of **confidence** and 5 very high **confidence** in the correctness of the provided attribute value. Meta-attributes constituted one of the novelties of the methodology used in the current case study, as they have not yet been mentioned in the context of attack trees nor attack–defense trees in the existing literature.

For the classroom experiment we only selected the three attributes **costs**, **difficulty** and **time** with which to decorate the ADTree. We provided the students with a verbal description similar to the one above and explained the value categories. In addition, we asked for the meta-attribute **confidence** with a reduced value domain of $\{1,\ldots,3\}$. The meta-attribute values were explained as ranging from `Unsure` (1) over `In-between` (2) to `Sure` (3).

5.3.4 Attribute Decoration

For the external case study, we created an empty table with 9 columns for the attributes and 79 rows for all nodes. Our intention for using a table was to prevent illegible values that may have occurred if the estimated values had been handwritten directly on the ADTree printed on a sheet of paper. Everyone who was supposed to estimate values was provided with the warehouse and the system description given in Section 5.3.1, the labeled ADTree given in Figures 5.5–5.8, the attribute descriptions given in Section 5.3.3 and sheets of paper containing an empty table template. The values were estimated independently over a period of one week. We did not require estimation of values for every node and attribute, we rather suggested a best effort strategy be applied.

In the external case study, performed with SINTEF and TXT e-solutions, the entire ADTree consisted of 79 nodes, where 59 were attack nodes and 20 were defense nodes. The case study was conducted as a game between *players*. Two players took the role of an attacker and two players took the role of a defender. Each group only filled out the nodes corresponding to its role. An extract of the resulting estimated values is shown in Table 5.1, where each line for an attack node represents a player who had been given the role of an attacker. Similarly for a defense node, each line represents a player who had been given the role of a defender. A dash in a cell indicates a conscious decision of a player not to estimate a value and an empty field indicates that the player was not considering (or forgot) to provide an estimate. The letters [D] and [B], inserted in front of the node labels, indicate that the node was a defensive node and a basic action, respectively. The first letter of the attribute values has been used to indicate the value, and the number denotes the **confidence** value.

During the external case study, the two players who were given the role of attackers spent approximately 90 min and 120 min, the players with the role of the defender spent 40 min and 90 min to complete Table 5.1. The players did not provide values for all nodes, in fact, for some attributes no node ended up with two values. Across the attributes, the maximal amount of nodes that were given two values was 93 %. For five attributes (**detectability**, **impact**, **penalty**, **profit** and **probability**) either an attacker or a defender did not estimate a value.

For the **special skill** attribute, we discovered that the attackers had not considered the same special skill. One defender estimated values only for basic actions and not for refined nodes. In total 436 values did not get filled in. This roughly amounts to 15.3 %. (Even if the attribute **profit** is not considered since it was the one which had the least filled in values, the percentage of missing entries still amounts to 11.4 %.) Such a large percentage seems to indicate unanticipated problems. We identified three categories of problems: attribute related problems, node related problems and methodological problems, which is discussed in Section 5.4.

The classroom case study was performed by ten master's students in a security modeling class. Due to the classroom setting, we had to adapt the realization of the case study. Any adaptation was carried out with the observations from the external case study in mind. First, the tree was presented and explained (instead of created). Then, definitions of the three attributes **costs**, **difficulty** and **time** were explained and summarized on the blackboard. The students were provided with three sets of papers on which the tree and an attribute were printed. Values for the **costs** attribute were filled in during class directly on the tree and not in a table. During this process the students were allowed to ask questions. Values for the **difficulty** and **time** attributes were given as homework. Unlike in the external case study, the students were asked to provide values for all non-refined nodes, irrespective of the type of the node, but asked not to provide values for refined nodes. A summary of the results for the "Breaking and Entering" subtree is provided in Table 5.2.

The classroom experiment was performed after the external case study. Hence, we tried to circumvent several of the identified problems. A first adjustment was to ask for fewer attribute estimates. We chose the attributes **costs**, **difficulty** and **time** since they already achieved a good performance during the external case study.

Name of the Node	Costs	Det	Diff	Imp	Pen	Prof	Prob	Skill	Time
Breaking and Entering	A,3	P,3	M,3	M,3	M,3	-	-	F,3	Q,3
	-		M,4		-		M,4	F,4	Q,4
Get onto Premises	C,4	P,4	T,4	M,4	M,4	-	-	F,4	Q,4
	C,4		T,4		L,4		H,4	F,4	Q,4
Get into Warehouse	C,4	P,4	T,4	M,4	M,4	-	-	F,4	Q,4
	C,3		M,3		M,3		H,3	F,3	Q,3
[D,B] Monitor with Security Cameras	E,4	P,4	T,4		H,4			F,4	S,4
	E,4	E,4	M,4				M,3	T,4	Q,3
[B] Climb over Fence	C,5	D,5	T,5	L,5	L,5	-	M,5	F,5	Q,5
	C,4		T,4		L,4		H,4	F,4	Q,4
[B] Enter through Gate	C,5	E,5	T,5	L,5	L,5	-	H,5	F,5	Q,5
	C,4		M,4		M,4		M,4	F,4	Q,4
[B] Enter through Door	C,5	E,5	T,5	M,5	M,5	-	M,5	F,5	Q,5
	C,4		M,4		M,4		M,4	F,4	Q,4
[B] Enter through Loading Dock	C,5	E,5	T,5	M,5	M,5	-	L,5	F,5	Q,5
	C,4		M,4		M,4		M,4	F,4	Q,4
Disable Cameras	A,3	E,3	T,3	L,3	M,3	-	M,3	F,3	Q,3
	C,3		M,3		L,3		-	F,3	M,3
[D,B] Barbed Wire	E,4	D,4	D,4		H,4			T,4	S,4
	C,4	E,4	T,4				M,3	T,3	Q,2
[D,B] Monitor with Biometric Sensors	E,4	D,4	D,4		H,4			T,4	S,4
	E,4	E,3	M,3				M,3	T,3	Q,2
[B] Laser Cameras	A,3	E,3	M,3	L,3	M,3	-	L,3	F,3	Q,3
	A,2		M,2		L,2		M,2	F,2	Q,2
[B] Video Loop Cameras	A,2	D,2	D,3	L,2	M,2	-	U,2	T,2	M,2
	A,2		M,2		L,2		L,2	F,2	M,2
Guard Against Barbs	C,4	-	T,4	L,4	L,4	-	H,4	F,4	Q,4
	A,4		T,4		L,4		H,4	F,4	Q,4
[D,B] Employ Guards	A,4	P,4	T,4		H,4			F,4	M,4
	E,4	E,4	T,4				M,3	F,4	M,2
[B] Use Carpet on Barbs	C,5	E,5	T,5	L,5	L,5	-	C,5	F,5	Q,5
	C,4		T,4		L,4		H,4	F,4	Q,4
[B] Wear Protective Clothing	A,5	E,5	T,5	L,5	L,5	-	H,5	F,5	Q,5
	A,4		T,4		L,4		H,4	F,4	Q,4
Nodes with less than two values	11	65	6	79	24	79	47	9	12
Attack nodes with two values	54	0	59	0	55	0	32	56	53
Defense nodes with two values	14	14	14	0	0	0	0	14	14

Table 5.1: Extract of raw data of the external case study. Each row represents the estimates of one player. The first letter of every field represents an attribute value, as abbreviated in Section 5.3.3 and the second represents the **confidence** level on a scale from 1 to 5.

Name of the Node	C,3	C,2	C,1	A,3	A,2	A,1	E,3	E,2	E,1
[D] Monitor with Security Cameras				4		1	2	1	
Climb over Fence	4	2	1		1				
Enter through Gate	5	2			2				
Enter through Door	3	2			1	1			
Enter through Loading Dock	3	2	1		3				
[D] Barbed Wire	1	2		3	1	3			
[D] Monitor with Biometric Sensors				2	1	1	5		1
Laser Cameras				1	2		4	3	
Video Loop Cameras		1		2	3		1	1	
[D] Employ Guards		1		3	2			2	2
Use Carpet on Barbs	5	1		4					
Wear Protective Clothing	5			1	1	1	1	1	

Table 5.2: Extract of table of classroom experiment raw data for the **costs** attribute. If [D] precedes a node name, it indicates a defensive action. The numbers indicate how often a specific result was given.

On first glance these adjustments seemed promising. For the second case study, only 144 out of 1710 values were not filled in. This amounts to roughly 8.4 %, compared to the 15.3 % in the previous case study. On second glance, however, a per attribute comparison looks less promising. Whereas in the external case study 3.5 % of the **costs** attribute were missing, the percentage increased to 7.0 % in the classroom case study. For the **difficulty** attribute it increased from 1.9 % to 11.7 % and for the **time** attribute from 3.8 % to 6.5 %. In conclusion, even though we adapted the case study in several aspects, this still did not ensure that all values were filled in.

5.3.5 Preparation of Attribute Values

The last section is concerned with providing raw data for attribute values of non-refined nodes. In this section, we describe a first step to turn these estimates into a basic assignment (Definition 4.3). We focus on the applicability of the methodology and not the computed results and their meaning.

In the external case study, all attribute values were supposed to be estimated by two players. In practice this means that every node is assigned at most two values, see Table 5.1. However, a basic assignment consists of exactly one value for every basic action. Therefore, we must have exactly one single value for each node and attribute.

We start from the data given in Table 5.1. Optimally, every player has estimated the same value. Then we can immediately use it as the single attribute value. In practice, this idealized scenario occurs seldom. There are several reasons why the values were not identical. First, some players may have opted not to estimate a value at all. Second, the player's understanding of the node may have been different. Third, it may also happen that no player has estimated a value for a node. Finally, it may simply be the case that they have a different opinion with regard to the value. Therefore, in a non-idealized scenario, we need a method to choose one representative value for each node.

Provided all players have estimated a value for a given node, we automate the process of combining the values by using the following procedure. First, we injectively map the attribute values into the natural numbers. In the case studies, for example, the **costs** attribute values `Cheap`, `Average` and `Expensive`, are transformed into 1, 2 and 3, respectively. We let n be the number of independently gathered pairs $(\text{attribute}, \text{confidence})$. In the external case study n was equal to 2, in the classroom case study n was equal to 10. Then we used the following formula:

$$(\text{rnd}\left(\frac{\sum \text{attribute} \cdot \text{confidence}}{\sum \text{confidence}}\right), \left\lfloor \frac{\sum \text{confidence}}{n} \right\rfloor), \tag{5.1}$$

where rnd represents ordinary rounding and $\lfloor \cdot \rfloor$ rounding down to the nearest integer. The choice of regular rounding for the first component allows us to transform it back into the original value domain, using the inverse of the original mapping. The choice of rounding the second component down reflects risk averseness. The results of the application of the formula in the external case study are the non-bold values in Table 5.3.

Name of the Node	**Costs**	**Diff**	**Time**
Breaking and Entering	**A,3**	M,3	Q,3
Get onto Premises	C,4	T,4	Q,4
Get into Warehouse	C,3	T,3	Q,3
[D,B] Monitor with Security Cameras	E,4	M,4	**S,4**
[B] Climb over Fence	C,4	T,4	Q,4
[B] Enter through Gate	C,4	T,4	Q,4
[B] Enter through Door	C,4	T,4	Q,4
[B] Enter through Loading Dock	C,4	T,4	Q,4
Disable Cameras	A,3	M,3	Q,3
[D,B] Barbed Wire	**A,4**	**T,4**	**S,4**
[D,B] Monitor with Biometric Sensors	E,4	D,3	**S,5**
[B] Laser Cameras	A,2	M,2	Q,2
[B] Video Loop Cameras	A,2	D,2	M,2
Guard Against Barbs	A,4	T,4	Q,4
[D,B] Employ Guards	E,4	T,4	M,3
[B] Use Carpet on Barbs	C,4	T,4	Q,4
[B] Wear Protective Clothing	A,4	T,4	Q,4

Table 5.3: Attribute values after the consensus meeting. The first letter of every field represents an attribute value, as abbreviated in Section 5.3.3 and the second represents the **confidence** level on a scale from 1 to 5. The letters preceding the node names denote basic actions [B] and defensive actions [D]. Values obtained during the consensus meeting are depicted in **bold**.

Example 5.1 To illustrate the application of Formula (5.1), we combine the **difficulty** attribute values estimated for the node "Enter through Gate". From Table 5.1 we can see that one player estimated the **difficulty** of "Enter through Gate" as `Trivial` (`T`) and his **confidence** level in this value was 5. The other player found the **difficulty** of the considered action to be `Moderate` (`M`) and his **confidence** level in this value was 4. The value domain for the **difficulty** attribute, as defined in Section 5.3.3, contains four values: `Trivial` (`T`), `Moderate` (`M`), `Difficult` (`D`) and `Unlikely` (`U`). These we transform into 1, 2, 3 and 4, respectively. Thus, we use the pairs $(1, 5)$ and $(2, 4)$ as inputs for Formula (5.1). We obtain:

$$(\mathrm{rnd}\left(\frac{1 \cdot 5 + 2 \cdot 4}{5+4}\right), \left\lfloor \frac{5+4}{2} \right\rfloor) = (1, 4).$$

After combining the values estimated by the players, we can map the resulting pair back to obtain that the **difficulty** of the node “Enter through Gate” is `Trivial` (`T`) (because of the 1), of which we are certain with a **confidence** level of 4.

The **costs**, **difficulty** and **time** values, as well as the corresponding **confidence** levels, obtained in the external case study for the remaining nodes with two estimations each, are depicted as non-bold pairs in Table 5.3. For the classroom case study, all values were computed using Formula (5.1), irrespective of the number of initial values given in Table 5.2. We defer a discussion of other possible methods to combine values where every player has provided a value to Section 5.4.5.

In the external case study, we encountered several exceptional cases where using the formula was problematic. It is, for example, impossible to apply the formula when not even a single value is given; it is highly doubtful to use it when the provided values differ substantially. We classified and discussed such critical cases at a *consensus meeting*.

The main goal of the consensus meeting was to obtain reasonable values that can then be used as input for further calculations on the ADTree. Therefore, the participants of the consensus meeting should have diverse backgrounds. We only conducted a consensus meeting in the external case study. During the meeting all players were present. At the meeting, we first decided to drop several attributes for which we did not have enough data. As a result, we only focused on the **costs**, **difficulty** and **time**.

We first ensured that the nodes’ estimates were correct, i.e., that the players had not mistakenly estimated a wrong value. Whenever mistakes were discovered, they were corrected, and the node was reevaluated. We also uncovered one inconsistency where the tree was not matching the scenario. To repair this mistake, we corrected the tree. (Since the classroom case study was performed after the external case study, the students were immediately given the corrected tree.) To obtain consensus values, the nodes were analyzed in context. More concretely, we looked at the meaning of the surrounding nodes, without however taking any values that were possibly assigned to the neighboring nodes into account.

During the consensus meeting, we identified the following categories and resolved the problems in the following way:

- *Nodes where no one had estimated a value*: We opted to discuss the value and eventually assign a single value. The players who had taken the opposite role commented on plausible values. Then we made a unanimous decision.
- *Nodes where not every player had estimated a value*: We also decided on a single value at the consensus meeting. Concretely, the player who had not

given a value, first explained why he had not done so, then the player who had given a value explained his choice. Then, a consensus was formed.

- *Nodes that had non-neighboring mapped values*: The player with the lower value explained his choice, then the player with the higher value explained his. After that, the involved players agreed on one of the given values or a compromise was chosen. Whenever a compromise was chosen, we lowered the **confidence** value.

- *Nodes where all given values had low* ***confidence*** *levels*: We also planned to discuss these uncertain values. However, due to time constraints, values in this category were skipped and Formula (5.1) was applied instead.

The final result of the consensus meeting is given as pairs of values in **bold font** in Table 5.3. Instead of the allocated hour, we spent two hours discussing the values. Out of the $3 \cdot 79 = 237$ possible values, there were 188 cases for which we applied Formula (5.1), 8 cases without any assigned value, 24 cases to which we had assigned only one value and 17 cases where the values diverged significantly.

Example 5.2 We have illustrated the **costs** attribute values resulting from the consensus meeting (also given in Table 5.3) in Figure 5.9.

5.3.6 Bottom-up Calculation of Attribute Values

As described in the previous two sections, providing a basic assignment for an ADTree is cumbersome. Fortunately, the ADTree methodology allows us to automate the calculation of values on ADTrees with the help of the bottom-up procedure (Definition 4.4). In this section, we calculate the values for the minimal **costs** of the attacker. Then we use the obtained values to analyze the warehouse scenario.

In both case studies, we were interested in calculating the minimal **costs** of a successful attack in the RFID warehouse scenario without reusability of defensive measures. The **costs** attribute is similar to the one introduced in Section 4.2.2 since it includes the meta-attribute **confidence**. We considered the situation where the attacker does not have any precise information on how the defender decides to protect the warehouse. Thus, for this calculation, we assumed that all possible defenses illustrated on the ADTree are in place and that they are fully functional, i.e., a defense attached to an attack node defeats the corresponding attack component, unless the defense itself is rendered useless by a counterattack.

We started by providing a basic assignment for the non-refined nodes of the tree, given the single values obtained in the previous section. In the case of the attacker's non-refined nodes, we used the pairs (costs, confidence) from Table 5.3 as initial values. Since the defender's **costs** do not influence the attacker's **costs**, we did not use the values from Table 5.3 in the case of the defender's non-refined nodes. Instead, we introduced an additional cost value `Infinite`, denoted by `X` and initialized non-refined nodes of the defender with the pair (`X`,5). This indicates that we are fully confident that it is infinitely expensive (and thus impossible) for the attacker to successfully execute a defender's action. These initial values allow us to

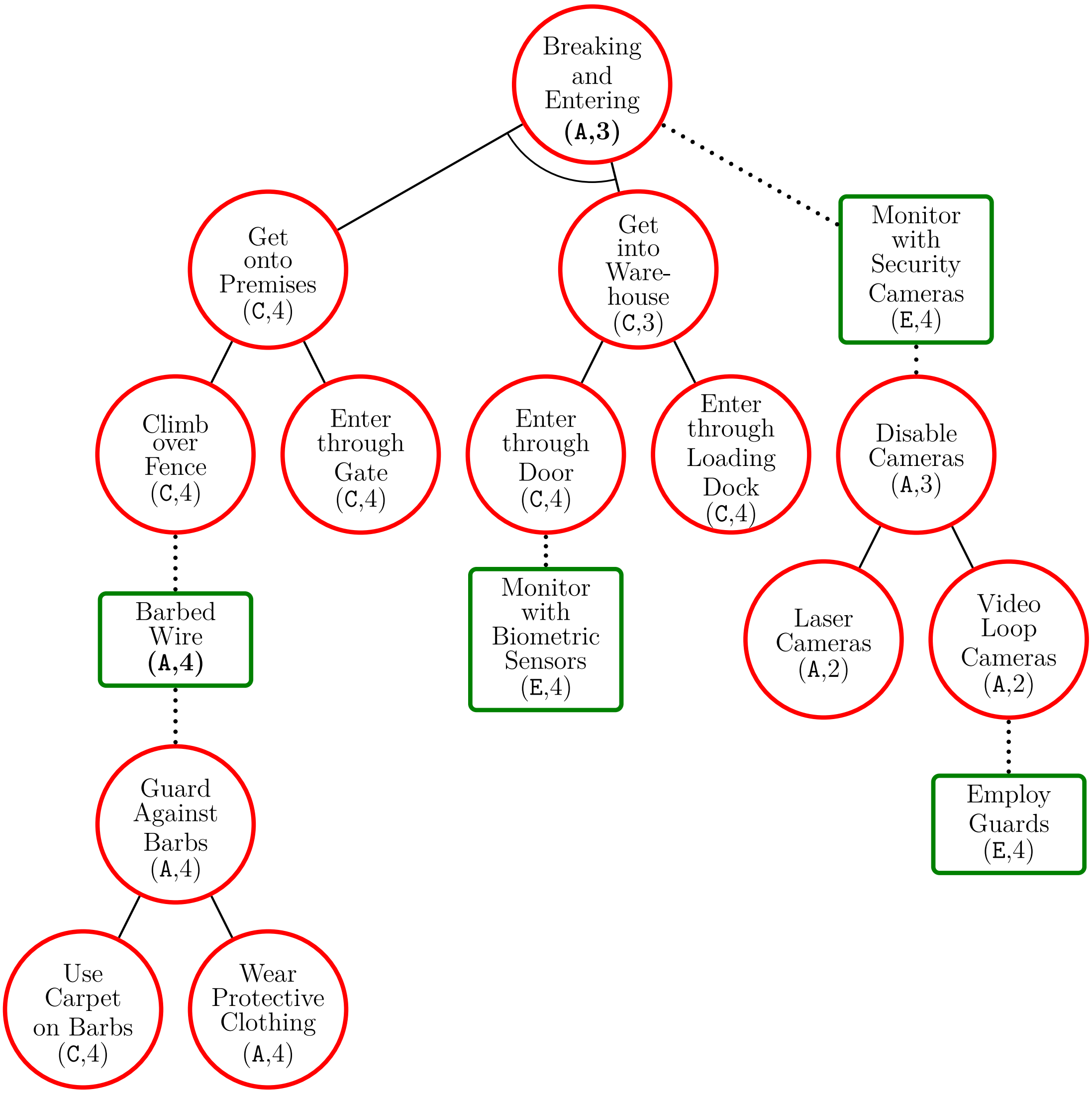


Figure 5.9: The "Breaking and Entering" subtree with the **costs** attribute values after the consensus meeting.

express the **costs** of the considered scenario from the point of view of the attacker, see also Example 4.6.

Since we eventually want to choose between different actions, we have to know how to compare different values. We use the linear order `Cheap` < `Average` < `Expensive` < `Infinite`. Moreover, we assumed that performing several actions belonging to the same **costs** category is not more expensive than performing only one such action. In the case studies, we were interested in the minimal **costs**. We therefore always chose the least expensive action whenever there was a choice between actions. We chose the maximal **costs**, in case several actions had to be executed. Note that the same operator would have been chosen when the reuse of tools would have been allowed. Last, we needed to specify which **confidence** level we chose. Whenever encountering an OR node, the **confidence** level was chosen to be the highest value that appeared among the set of actions with the chosen **costs** value, in the other case, the **confidence** level was chosen to be the lowest value that appeared among the set of actions with the chosen **costs** value.

To express the resulting attribute domain formally, we introduce the following value domains. Let the possible values `Cheap`, `Average` and `Expensive` be represented by Cost $= \{\mathtt{C}, \mathtt{A}, \mathtt{E}\}$, the values `Quick`, `Medium` and `Slow` by Time $= \{\mathtt{Q}, \mathtt{M}, \mathtt{S}\}$ and the values `Trivial`, `Moderate`, `Difficult` and `Unlikely` by Diff $= \{\mathtt{T}, \mathtt{M}, \mathtt{D}, \mathtt{U}\}$. Finally, we denote with Conf $= \{1, 2, 3, 4, 5\}$ (or Conf $= \{1, 2, 3\}$ in the classroom experiment) the set of **confidence** values. The resulting attribute domain of the **costs** attribute is given by

$$A_{\text{costcat}} = ((\text{Cost} \cup \{\mathtt{X}\}) \times \text{Conf}, \widetilde{\min}, \widetilde{\max}, \widetilde{\max}, \widetilde{\min}, \widetilde{\max}, \widetilde{\min}), \tag{5.2}$$

where

$$\begin{aligned} \widetilde{\min}((c_1, x_1), \ldots, (c_k, x_k)) &= (\min_{1 \le i \le k} \{c_i\}, \max_{j | c_j = \min_{1 \le i \le k} \{c_i\}} \{x_j\}), \\ \widetilde{\max}((c_1, x_1), \ldots, (c_k, x_k)) &= (\max_{1 \le i \le k} \{c_i\}, \min_{j | c_j = \max_{1 \le i \le k} \{c_i\}} \{x_j\}). \end{aligned}$$

The attribute domains of the **difficulty** and **time** attributes are obtained by replacing the set Cost with Diff and Time, respectively.

With the assumption that all possible defenses are present and fully functional, the actual minimal **costs** of a successful attack can be lower than the one obtained using our calculation. Indeed, in reality, the defender may decide not to implement some of the defenses and thus the **costs** of the corresponding counterattacks will not be taken into account for the final **costs** of the attacker. However, by taking the described approach we use a safe solution, in the sense that

- the calculated minimal **costs** will not be lower than the actual minimal **costs**, i.e., the minimal **costs** will not be underestimated,
- and the resulting set of attack components that have to be executed in order to achieve the cheapest attack forms a successful attack. In particular, this means that the defender cannot protect himself against this attack.

For the external case study, our calculation of the minimal **costs**, the minimal **difficulty** and the minimal **time** shows that, in order to achieve an attack with minimal **costs**, an attacker needs to spend an `Average` amount of money. We have a **confidence** level of 3 in this value. The corresponding attack consists of "Disable Backend". To execute the attack of minimal **difficulty**, the attacker should also perform the "Disable Backend" action. Its **difficulty** is `Medium` of which we are confident with level 3. The **time** it takes to perform the fastest attack is `Quick` of which we are confident with level 2. To achieve the fastest attack, an attacker should execute the "Disable Tag" action. We observe that for all three attributes, the optimal attack option is an action which we have chosen not to refine, see Figure 5.5. To be able to give a more insightful example, we compute the **costs** attribute for the subtree rooted in the node "Breaking and Entering".

Example 5.3 The values resulting from the bottom-up approach for the **costs** attribute are depicted in Figure 5.10. We deduce that an attacker can break and enter when he spends an `Average` amount of money and we are confident with level 2 about that. To perform the attack, an attacker has two options: either he

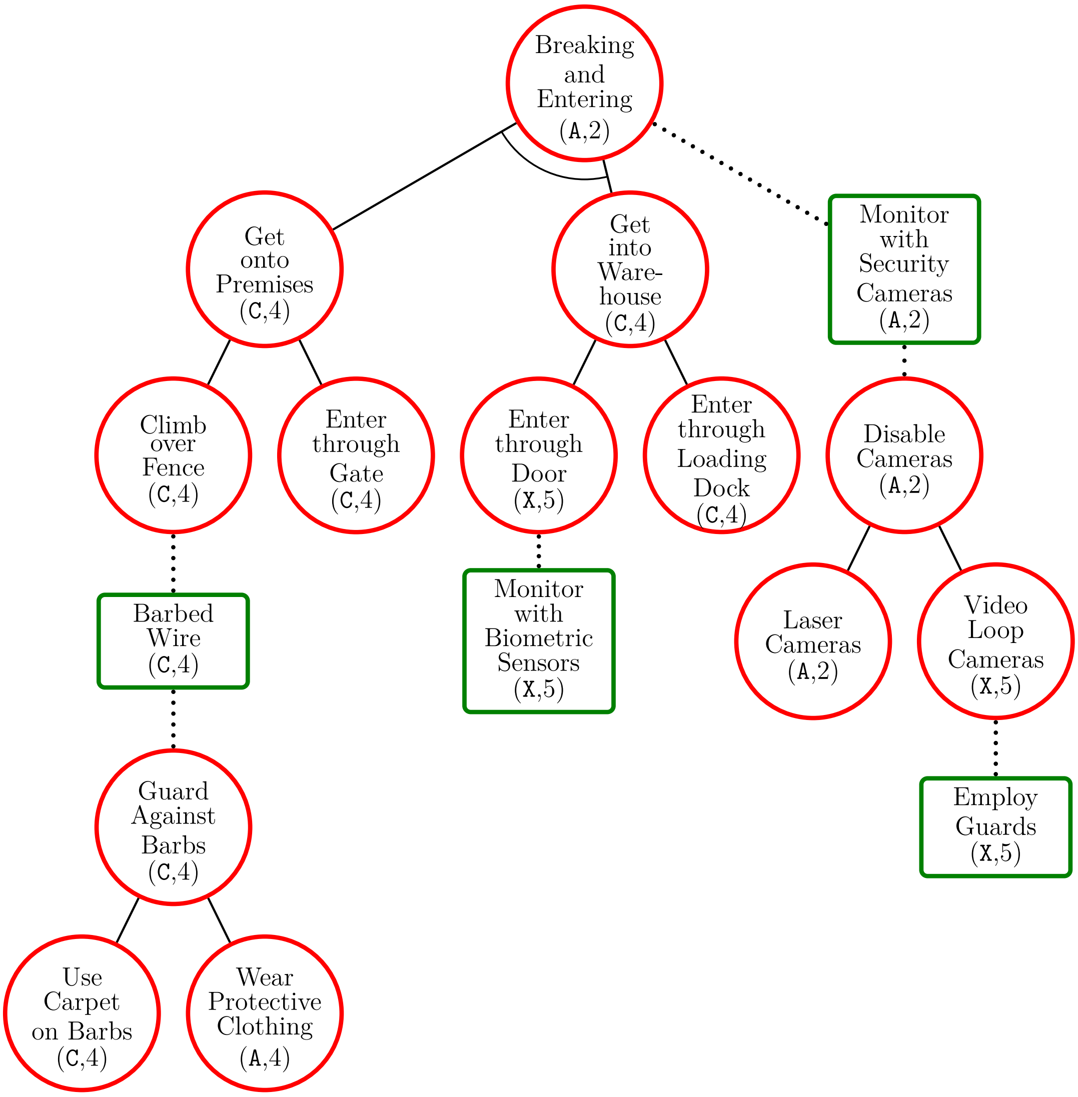

Breaking and Entering (A,2)
Get onto Premises (C,4)
Get into Ware-house (C,4)
Monitor with Security Cameras (A,2)
Climb over Fence (C,4)
Enter through Gate (C,4)
Enter through Door (X,5)
Enter through Loading Dock (C,4)
Disable Cameras (A,2)
Barbed Wire (C,4)
Monitor with Biometric Sensors (X,5)
Laser Cameras (A,2)
Video Loop Cameras (X,5)
Guard Against Barbs (C,4)
Employ Guards (X,5)
Use Carpet on Barbs (C,4)
Wear Protective Clothing (A,4)

Figure 5.10: The “Breaking and Entering” subtree with **costs** calculated by the bottom-up procedure.

has to “Use Carpet on Barbs”, “Climb over Fence”, “Enter trough Loading Dock” and “Laser Cameras” or he has to “Enter through Gate”, “Enter through Loading Dock” and “Laser Cameras”.

Using the attribute domain A_{costcat} defined in Equation (5.2), we computed the minimal **costs** of an attacker for every subtree. This attribute value is depicted in Figure 5.10 for every node. It seems natural to compare these values with the ones gathered during the decoration phase, as presented in Table 5.3. In Section 5.4.6 we elaborate why such an approach may be suitable on attack trees but not on ADTrees.

5.3.7 Evaluation of the RFID Goods Management Case Studies

A comparison between the two case studies seems to indicate that just because the players had a good understanding of an attribute during the external case study, it did not necessarily mean that the students had the same understanding. A further comparison is only partially meaningful since in the external case study, a table was provided and the meta-attribute value domain was different. With hindsight, this was a valuable lesson that indicates that computer tool support (Section 5.6) is indispensable to prevent nodes from not being filled in with values. Tool support may also improve the quality of the values since then more focus can be put on the actual values and not so much focus has to be put on the methodology.

Since the data gathered during the external case study was limited, we did not use it further. Instead, we used the data from the classroom case study to support several hypotheses. We operate under the assumption that students provide good value estimates and are, therefore, helpful to evaluate the methodology but that they are not security experts.

First, we analyzed whether as the **confidence** value rises, the more refined and detailed a tree is. For this we compared the average of the **confidence** values for the "Breaking and Entering" subtree with the average of all 57 **confidence** values. For the **costs** attribute the averages were 2.51 vs. 2.16, for the **difficulty** attribute they were 2.36 vs. 2.30 and for the **time** attribute, they were 2.38 vs. 2.28. This supports our initial hypothesis and indicates that it is (subjectively) easier to provide a value for a more refined tree that represents a more detailed scenario.

Second, we also examined whether the students were more likely to omit to fill in values for non-refined nodes that are not leaves. In the given ADTree, 14 out of 57 nodes were non-refined nodes that were not leaves. This yields a percentage of roughly 25 % non-refined nodes compared to the total number of nodes to be filled in. For the **costs** attribute, there were 40 nodes without values, 22 of which were non-refined nodes that were not leaves. This yields an approximate percentage of 55 %. For the **time** attribute, there were 37 nodes without values, 22 of which were non-refined nodes that were not leaves. This amounts to approximately 59 %. For the **difficulty** attribute, there were 67 nodes without values, 19 of which were non-refined nodes that were not leaves. This equals roughly 28 %. Unfortunately two sheets of papers were lost before we could start the evaluation. Discounting these sheets, the percentages actually increase to 68 % (for the **time** attribute) and to 33 % for the **difficulty** attribute. In summary, these values indicate there is a tendency to forget to fill in non-refined nodes that are not leaves.

Finally, we tried to corroborate the hypothesis that the **confidence** value is high if and only if the variance of the estimated values is low. To compute the variance, we proceeded as follows. We gathered all 10 value estimations for a certain attribute in a vector $X = (x_1, \ldots, x_{10})$. (Naturally, if values were missing, we used a shorter vector X.) Then, for the **costs** attribute, we have $x_i \in \mathrm{Cost} \times \mathrm{Conf}$, for the **difficulty** attribute, we have $x_i \in \mathrm{Diff} \times \mathrm{Conf}$ and for the **time** attribute, we have $x_i \in \mathrm{Time} \times \mathrm{Conf}$. With the help of the expectation operator, denoted by $\mathbb{E}$, we compute the variance as follows:

$$\mathrm{Var}(X) = \frac{\sum_i f_2(x_i)(f_1(x_i) - \mathbb{E}(X))^2}{\sum_i f_2(x_i)},$$

where $f_2(x_i)$ denotes a projection on the **confidence** value. Moreover, $f_1(x_i)$ transforms an attribute value into a natural number in ascending order. For example, for the **costs** attribute we have the following mapping: `Cheap` $\mapsto 1$, `Average` $\mapsto 2$ and `Expensive` $\mapsto 3$. For **costs**, **difficulty** and **time**, we obtain variances for each node that lie between 0 and 0.746, 0.074 and 0.776 as well as 0 and 1.252, respectively. The resulting averages over all nodes are computed to 0.33, 0.40 and 0.46. To be able to compare these values, we also calculated the variance for randomly provided values, which lies around 0.66. Hence, the empirical variance values are lower than the expected variance for random values. Contrarily, the computed total **confidence** gave values between 1.56 and 3, 1.88 and 2.78 as well as 1.78 and 2.70. The resulting averages 2.16, 2.30 and 2.28 indicate a rather high confidence. To conclude, we found some evidence suggesting that a high **confidence** in the estimated values results in a low variance amongst the values. However, to prove or disprove this hypothesis further experiments are necessary.

In conclusion, an analysis of quantification by non-experts showed some promising results. To validate the results, we suggest there is a focus on improving the value estimations of the basic assignments by non-experts.

5.4 Practical Observations

During the course of the three case studies, we observed numerous situations where a user has a design choice of how to proceed when applying the ADTree methodology. We elaborate how different choices affect the methodology and use the discussion to answer Questions Q1–Q6. A summary of this discussion ordered according to the work flow is provided in Table 5.4 in Section 5.5.

5.4.1 Question Q1: Meaning and Visualization of Defenses

We restate Question Q1: How can we use ADTrees to find out which defensive measures are present in a system? And which techniques allow us to construct a precise model?

In an ADTree model, the syntax and the semantics do not specify whether the depicted nodes represent an actual scenario or potential attacks and defenses. We, therefore, need to specify what an ADTree actually depicts. There are two reasonable possibilities. First, we could view the tree that includes all nodes of the root node type that can be reached from the root node via nodes of the same type. For an ADTree rooted in an attack node, this would be an attack tree. This tree would represent how to attack a deployed system. Any further nodes in the tree would then represent hypothetical defenses, i.e., defenses that could be put in place, but are not yet installed. These defenses could, in turn, be countered and re-countered by other hypothetical actions. Second, it is possible to select a (fictional) scenario where no defenses (or attacks) have been implemented. Step by step we add potential defenses (or attacks) and countermeasures until we have a suitably detailed model. Note that for a scenario without countermeasures these possibilities coincide. (Design Choice D1.4)

Both possibilities have advantages as well as disadvantages. When modeling an actually deployed system, we strive to model as accurately as possible. In a deployed system, we do not need to puzzle about which security measures could be put into place, we can simply observe which defensive measures are in place. We represent these defenses indirectly by adding actions that counteract these measures to the model. In the case study on the RFID goods management system, this means that the security cameras from the floor plan, see Figure 5.4, are not explicitly modeled as defense nodes in the ADTree. The cameras mentioned in Figure 5.7 are additional cameras that could be put into place.

However, a specific adaptation to an existing scenario is undesirable when, for example, storing an ADTree model as a pattern in a library. In this case, it is preferable that all defenses, including the already existing ones, are depicted in the ADTree. If we want to convert an ADTree library template into an ADTree model for an actual scenario, we only have to take existing defenses into account and model them indirectly as necessary attack steps. (Design Choice D1.3) Moreover, additional information about the attacker and defender can be included into the model, by adjusting the tree to the considered situation. For example, if the defender only has a limited budget, any attack that is too expensive can be removed, e.g., following a procedure similar to pruning (Section 4.4.2).

In certain scenarios, some actions may only be valid when one or more external conditions are fulfilled. We can generically model this in the tree to improve the accuracy of the tree. To model a requirement or a condition, any node that is not a disjunction gets an additional conjunctively connected child that expresses the requirement or the condition. If a node is a disjunction, a conjunction with two children replaces the disjunction. One child is the subtree rooted in the initial disjunction, the other child depicts the requirement.

During the initial case study, we noticed that the ADTree depicting the scenario (Figure 5.2) does not contain any disjunctively refined attack nodes. This is due to the fact that in the original textual draft exactly one way of attacking online auctions was described. While modeling an attack path is certainly possible with the ADTree methodology, it loses some of its potential. In fact, any of the depicted defenses would suffice to foil the attack. Naturally, if the tree were extended to include other attack options, there would be disjunctively refined attack nodes making an analysis of necessary defensive actions more interesting.

To answer Question Q1: It is possible to model defenses explicitly as children of the attack action they counteract. In a tree adapted to a scenario, however, it may be preferable to only model them implicitly. Such implicit modeling as well as modeling requirements as conjunctions improves the expressive power of the ADTree model.

5.4.2 Question Q2: Usefulness of Transformations

We restate Question Q2: Are semantics preserving transformations on ADTrees a useful tool? Do they help us to improve our understanding of a scenario?

Recall that the root node of the online auction fraud case study has four children, see Figure 5.2. Each child represents one of the four phases: prepare, buy, ship and sell. This choice and, therefore, the refinement with four nodes was arbitrary. Instead, we could have chosen to refine the root node according to the involved people, i.e., Rachel, Perry and Otto. Such a choice would lead to a different ADTree and consequently would group different nodes together as parts of subtrees.

The grouping, however, is subjective since it is not specified by the scenario. We have previously come across a similar phenomenon when modeling how to subdue a guard, see Figure 3.1. There, the order of the children of a node was not given by the scenario. Such an impreciseness is one of the reasons for the introduction of different semantics. The semantics allow us to specify whether or not different ADTree models that arise from different groupings represent the same scenario. In the particular case of the online auction fraud scenario, different groupings of the attack nodes represent the same scenario in every semantics where conjunctive and disjunctive refinements are associative.

As indicated in Section 3.5, semantics can also be seen as transformations of ADTree models. With the help of these transformations, it may be possible to push countermeasures towards the root or to combine several conjunctive refinements into one. Both transformations are likely to flatten the tree at the expense of widening it.

Besides reshaping the model, transformations help us to gain insight about the structure of defenses. Suppose that a defense fully protects against attack A and we later discover that the same defense actually also fully protects against attack B. If A and B are the only children of a disjunctively connected node C, transformations help us to see that the defense actually protects against C. Similarly, they could help us discover missing or redundant defenses. For example if C is conjunctively refined instead of disjunctively refined, we only need to implement the defense once.

A suitable choice of groupings and transformations may depend on individual preferences and usually involves taste. It is, therefore, infeasible to specify universally accepted choices. Nevertheless, it is possible to determine certain transformations to create normal forms, as, for example, done in Section 3.5. These normal forms, in turn, could help us to quickly understand a model through a fast visual inspection. They may also help us to easily discover missing or redundant defenses. Suppose the normal form of an ADTree is a tree that does not contain any attack node leaves. Then, there exists a (potential) defense for every attack. In the De Morgan and the multiset semantics such normal forms that do not contain any attack leaf nodes exist.

While experimenting with different grouping strategies and normal forms, we realized that modeling dependencies between different nodes is challenging and that transformations do not help to overcome this problem. For a given tree, it is, for example, not always possible to find an equivalent model such that dependent nodes are always children of a common parent. For example, in the online auction fraud ADTree the node “Otto from Dave” in subtree “Delivery Process” is not a sibling of the node “Otto Sells to Perry” even though it must be the same Otto that first acquires the goods, before he can sell them. Alternative groupings suffer from similar dependency problems. In general, it is not possible to ensure that dependent nodes are siblings of each other. These problems are inherent to tree structures. To be able to display all possible dependencies and, therefore, also be able to quantify probabilistically dependent actions, we combine ADTree with Bayesian networks in Section 6.2.

Another kind of dependency arises when we want to specify that the same actor needs to execute several actions. On the one hand we want the model to be as precise as possible. We could achieve this by specifying which particular person executes the actions in a particular case. On the other hand we want that the models can serve as templates. This would only require us to indicate that the same person needs to execute the action and we would not have to specify which one. Also this second kind of dependency can not be overcome with the help of transformations. To facilitate reusing models and being precise at the same time, a role-based approach could be applied. Templates with role-based placeholders could be stored in libraries. These placeholders should then be instantiated when an ADTree template is adapted to a specific scenario. (Design Choice D1.3)

Even though transformation rules are defined on ADTerms, we have, so far, only discussed their impact on ADTree representations. A main difference between ADTrees and ADTerms is that refined nodes in ADTrees are equipped with a label. This label is not present in an ADTerm. The fact that ADTerms do not store information about the refined nodes complicates a visual comparison, but it simplifies an automated comparison with the help of a unique normal form. Being able to say whether two terms have the same normal form is important because it is a first step to be able to compare semantically non-equivalent trees.

To answer Question Q2: Semantics preserving transformations allow us to change the shape of a specific scenario representation. They are, therefore, well-suited to compare different representations of the same scenario. They also allow us to gain insights into the structure of the defenses. They do not, however, help us to solve problems related to dependent actions.

5.4.3 Question Q3: Practical Model Restrictions

We restate Question Q3: Should there be any restrictions when setting up a tree model, for example, should the labels of the nodes have a specific form or should the tree have a predefined depth?

A general modeling question is always how detailed a model should be compared to the actual system or scenario. In ADTrees, this question translates into two aspects. First, what are suitable node labels and second, what is a suitable level of refinement.

On the one hand, node labels are important because they help the users to understand the scenario without reading the scenario description in detail. Node labels that are too short may lead to confusion. On the other hand, the labels should be concise because if they are too long and detailed they are difficult to display and reduce the chance of reusability. To make the tree as self contained as possible, it is (often) beneficial that the node label consists of a noun and a verb. In rare cases a single noun or a single verb may also be sufficient. (Design Choice D1.1)

For modeling purposes, the level of refinement is the more crucial aspect. In practice, a model is often set up under limited time or costs. Limited time generally leads to less elaborated trees, i.e., a less detailed model. The level of refinement, may also be influenced by the creator's knowledge of the scenario in particular and of security in general. For example, in the external case study, the players created the tree. They had in-depth knowledge of both security in general and the scenario in particular. If instead the creation of the tree is delegated to independent security experts, the subtree related to the security expert's field of expertise (e.g., social security) may be detailed while other subtrees (e.g., RFID security) may be less refined. (Design Choice D1.2) Last, but not least, the level of refinement is also affected by the availability and use of templates or other models from security repositories. In general, we advocate the use of templates since they speed up and simplify the modeling process. However, there might be drawbacks to purely relying on such available information. New attacks or defenses, for example, are less likely to be found. (Design Choice D1.1)

There is no absolute advice on the best level of refinements an ADTree should have. In the case studies on the RFID goods management system, however, we have seen that the resulting attribute values were primarily determined through less refined branches. If any of these values had been incorrect or missing, it would have had a disproportionately large impact on the result. This, in turn, indicates that the relative level of node refinement is crucial. To avoid biased results, the level of refinement should roughly be the same for all branches. Generally speaking, this is achieved by providing the same intuitive level of understanding across different branches in the tree. Note, however, that this does not mean that all non-refined nodes are at roughly the same heights. A heuristic way of achieving this is to limit the number of nodes in the entire tree and to verify that subtrees of the same parent node have a comparable number of nodes. (Design Choice D1.5)

To answer Question Q3: Yes, there should be restrictions when modeling the tree. Node labels should be concise and the level of tree refinement should be comparable across the entire tree.

5.4.4 Question Q4: Attribute Decoration

We restate Question Q4: What has to be considered when providing attribute values for nodes?

In the two case studies on the RFID goods managements system, numerous obstacles arose when trying to provide values for an attribute for a given node. The obstacles can roughly be classified into three groups. First, it is imperative that everyone who provides a value has a good understanding about the model and, therefore, for every node in the tree. Second, everyone who provides a value should also have the same understanding of the attribute, as well as what the possible attribute values represent. And third, if several people estimate values, they all need to have the same understanding of who estimates which nodes according to which criteria. We elaborate in more detail on the three groups in the following three paragraphs.

Understanding a node within a model To gain a detailed understanding of the scenario it is necessary that, at some point during the modeling process, the structure of the tree is frozen and no more nodes are added or removed. Structural changes made to the tree possibly falsify value estimations in other parts of the tree. (Design Choice D1.6) To avoid such complications and repetitive work the model of the tree should be sufficiently accurate before any values are assigned. Much for the same reasons, all attributes should be evaluated on the same model and not on a possibly pruned tree. (Design Choice D1.8)

However, agreeing on one definite model is not always possible. Especially in the classroom case study, the students did not completely agree with the tree structure. To mitigate this problem, two options are available. First, the introduction of the meta-attribute **confidence** allows us to provide values with `Low` **confidence**. With this option it is reasonable to ask everyone to provide a value irrespective of whether they agree with the tree or not. Second, a new attribute called **disagree with node** with a Boolean value range could be introduced. Any node or subtree that has been flagged with this attribute should then be reevaluated or discussed at the consensus meeting. (Design Choice D2.3)

Since the structure of the tree will be fixed at some point in time and later adaptation should be avoided, the resulting models may be incomplete. This phenomenon is common in practical models. Any quantitative analysis resulting from an incomplete (and thus incorrect) model has to be taken with a grain of salt. Unfortunately the ADTree methodology is not robust and any forgotten node can, in principle, influence the final outcome. Since a complete model is infeasible to attain and an incomplete model can yield wrong quantitative results, we advocate the use of *sophisticated* models. A sophisticated model is error free, sufficiently detailed and comparably refined to increase a user's understanding of the scenario. (Design Choice D1.7)

Even though freezing a certain tree structure is necessary for a common understanding of the scenario, it is not sufficient. A user's understanding of a specific node may still be incomplete or even incorrect. One of the main problems that we identified was that node labels were often not self-explanatory and led to confusion. As we mentioned in the last section, we propose to use simple labels consisting of a noun and a verb. On the one hand, this brevity allows us to graphically represent the nodes. On the other hand, short labels, such as, "Break Authentication", "Laser Cameras" or "Hide in a Box", are difficult to understand without context. (Design Choice D1.1) This implies that looking at parents, siblings and child nodes as well as the corresponding main goal is often necessary when providing attribute values for non-refined nodes. Consider, for example, the node "Enter through Door". Without taking its parent node "Get into Warehouse" into account, it is impossible to estimate the corresponding values for the **difficulty** and the **time** attributes. More explicitly, the values may differ depending on which door we are interested in: the warehouse door, the bathroom door or the administrative office door.

Hence, in hindsight, the tables provided to be filled in during the external case study made it more difficult to take the context into account. In the classroom case study we approached this problem by extensively explaining the scenario depicted by the ADTree, making the students aware of the context of the scenario and by providing them with the ADTree on a sheet of paper on which they were supposed to fill in the attribute values. Naturally, this strategy also had drawbacks. First, it is not apparent if all values that are supposed to be provided are actually provided. Second, if values are provided with the nodes' context in mind, people might be led to assign different values to nodes with the same label. Naturally, this should not happen. It could, however, be an indication that the node labels should be different. (Design Choice D4.2)

Directly related to inconsistent values is the problem of how to assign values to nodes that have the same labels. These nodes, such as "Secure warehouse" or

"Attended Visitor's Log", were mentioned twice in the initial table in the external case study and occurred twice on the tree provided in the classroom case study as can be seen in Figure 5.6. On the one hand, the values assigned to two different occurrences of the same action may be different in the case when the context is taken into account. On the other hand, if nodes are handled independently of the tree, it would be more reasonable to associate similar values with similarly labeled nodes.

Thus, it seems reasonable to conclude that nodes with the same label should be given the same values. If this appears to be unreasonable, one of the nodes should be renamed. (Design Choice D4.3)

A final issue with the understanding of nodes is whether attribute values should be assigned to non-refined nodes only or also to refined nodes. This question is closely related to the meaning of the refined nodes and the bottom-up algorithm. We discuss it further in Section 5.4.6.

Understanding the attribute Naturally, everyone who provides a value which is later used in the attribute computation should have a good understanding of this attribute. During the case studies, we learned that the users tended to estimate values by using their own interpretation of the attributes, even though they were aware of the definitions provided in Section 5.3.3. One reason was that the initial[9] descriptions were imprecise or diverged from a player's belief of what the attribute should represent. During the external case study, we apparently did not stress enough that *only* the given attribute description should be used. During the classroom experiment we, therefore, wrote the definition on the blackboard and explained it thoroughly.

In particular, the attribute descriptions in Section 5.3.3 are underspecified. For example, the **special skill** description states the use of a Boolean value domain, but does not specify which kind of special skill, e.g., **insider knowledge**, **electricity** or a certain technical **skill** is supposed to be evaluated. Moreover, the **penalty** and the **profit** attribute descriptions are only given from the perspective of an attacker. Furthermore, the **impact** attribute describes consequences from a system's owner or a defender's point of view, the scale given in Section 5.3.3, however, indicates attacks. Even an explicit reference to only an attacker, as in the case of the **detectability** attribute or a defender, as in the case of the **probability** attribute, led to confusion for the corresponding player. In summary, a reference in the attribute description to a specific role resulted in players with different roles not filling in certain values. Specific references to only one player should, therefore, be avoided. Besides adapting the **costs**, **difficulty** and **time** attribute definitions, we also avoided distinguishing between different roles in the classroom experiment and asked the students to fill in all values, for attack as well as defense nodes. (Design Choice D2.1)

Besides the attribute description, Section 5.3.3 also offers an explanation for different attribute value domains. In both case studies on the RFID goods management system, we opted to use value domains with up to 5 levels. It is entirely possible to use real numbers, intervals or even discrete probability functions as value domains.

[9] we adapted the descriptions for the **costs**, **difficulty** and **time** attributes several times during the case study

Choosing a different domain is a trade-off between the time a person spends on estimating an accurate value and the inclination of actually providing a value. Using a more fine-grained scale to achieve more exact results could be counterproductive if the number of people who estimate a value decreases. Furthermore, increasing the graining of the scale may make it more difficult to distinguish between values. Even with the chosen value domain of up to 5 levels, the provided explanation was sometimes not precise enough. For example, `High` **costs** are different for students and millionaires. In order to guarantee comparability across different people that estimate values, the categories should be given on an absolute scale. (Design Choice D2.2)

However, even knowing an attribute keyword, its description and an explanation of possible values may still not suffice. For example, simply knowing that **costs** are estimated, is not enough. Depending on whether these **costs** are supposed to represent minimal **costs**, maximal **costs** or average **costs**, a different default value represents a conservative estimation.

To overcome the above problems, we propose to always use the attributes in forms of questions, see Section 4.4 together with precise attribute and value domain descriptions.

Finally, the description and the use of the meta-attribute **confidence** should have been made more precise. One of the players in the external case study chose to use the same **confidence** level for all attributes of a given node with the intention to save time at the expense of less accurate values. Other players selected a different **confidence** level for every estimated attribute value. To make full use of meta-attributes, they should always be estimated on a per attribute value basis. (Design Choice D4.1) All players in the external case study concluded that the scale of the **confidence** meta-attribute should be reduced. We implemented this conclusion in the classroom case study by allowing only the three possible values `Unsure`, `In-between` and `Sure`. Results from both case studies could not conclusively determine which scale leads to superior outcomes. (Design Choice D2.4)

Understanding the methodology As mentioned previously, the external case study was performed by four people: two of them played a role of the attacker and two the role of the defender. However, it was not explicitly specified to which nodes the players should assign the values. This led to inconsistent data. Each of the players provided values for nodes related to his or her role, but one of the attacker players also estimated some values for the defender nodes. The use of hypothetical roles also allows us to distinguish knowledge of a user and knowledge of a role. Therefore, it could be asked whether users only take the part of the scenario corresponding to their role into account or whether they should base their decisions on knowledge about the other player and the entire scenario. Assigning different roles also made it possible that some attributes, e.g., **penalty**, **profit** or **impact**, were only estimated by either attackers or defenders. (Design Choice D3.1)

Assigning a specific role to a player initially seemed like a good idea since it minimized the players' work load. Of course, it also reduces the number of provided estimations. To obtain a larger set of values, we refrained from assigning different roles to the students in the classroom experiment. It is unclear to what extent increasing the number of different estimations increases the quality of the estimated values. (Design Choice D3.1)

Instead of increasing the size of the raw data it may be more suitable to increase the quality of the estimations. For example, if specialists within certain domains are available, their knowledge should be exploited. In this sense, a janitor could estimate nodes related to physical building security, i.e., nodes depicted in Figure 5.7, whereas a psychologist might be better suited to estimate values for nodes related to social engineering, i.e., nodes depicted in Figure 5.6. (Design Choice D3.2)

To answer Question Q4: When providing values that are used for attribute evaluation, they need to be consistent. This means, if several users estimate values for use cases, these users need to have a common understanding

- of the scenario and of every single node in the ADTree,
- of the attribute including the attribute domain, the value domain and the question and
- of the methodology in general.

5.4.5 Question Q5: The Basic Assignment

We restate Question Q5: What are suitable methods to turn the values provided for the nodes into a basic assignment?

When estimations are provided by different people, this usually yields heterogeneous data, even if we respect all instructions presented in the previous section. Since the bottom-up procedure requires exactly one value for each node as input data, it needs to be homogenized. In Section 5.3.5, we have already described one possible option to select a suitable homogenized representative: We applied Formula (5.1) for majority of the values (Design Choice D5.1) and discussed the remaining values at a consensus meeting. (Design Choice D5.3) The formula consists of a weighted average for the attribute value and an estimation of the **confidence** value, which reflects risk averseness. (Design Choice D5.2) Naturally, selecting the average, the median, the highest or the lowest value or an entirely new value instead of the weighted average is also conceivable. Similarly possible are other **confidence** estimations. The desired methods usually depend on the scenario and possibly the attribute.

When using a formula to compute a representative of a dataset, having more input values is generally preferable over having fewer input values. However, the choice of the representative may depend on the actual estimations. To better be able to differentiate the data for different nodes, we distinguished six categories. Then, for each category, a different approach determines the representative. For example, a representative could be selected using an average, using a minimum value, using ranges or deciding on the final value at the consensus meeting. The proposed categories are:

C1) Nodes with as many attribute values as players.

C2) Nodes where all estimated values have a low confidence.

C3) Nodes where the values diverge significantly.

C4) Nodes where the **disagree with node** flag is set.

C5) Nodes where at least one player has not estimated a value.

C6) Nodes where no player has estimated a value.

The categories are ordered according to a descending scale of possible automatic treatment and are not necessarily disjoint. Whereas for Category C1 it is entirely reasonable to combine the input values automatically into a single value, this is not even possible for Category C6. To gain a partition for the dataset, a node fulfilling several criteria is only placed into one category. Amongst the suitable categories, a node is placed into the one with the lowest number.

Categories C2 and C3 themselves are merely suggestions since the expressions *low confidence* and *diverge significantly* do not unambiguously specify whether or not a node fits into the category. In the external case study, we specified that all nodes with **confidence** levels 1 and 2 classify as being of low confidence. Nodes whose attribute values were not neighboring values, we specified as diverging significantly. The nodes in Categories C4 and C6 and to a lesser extent the nodes in Category C5 indicate that there exists a problem with the model description. These nodes should be discussed at a time-restricted consensus meeting. (Design Choice D5.4)

To answer Question Q5: We propose to use the presented classification to determine a basic assignment for the nodes. Naturally, other methods to select a suitable representative for each node and attribute are conceivable.

5.4.6 Question Q6: Bottom-up Computation

We restate Question Q6 What are the strengths and weaknesses of the bottom-up algorithm?

Using the basic assignment, the bottom-up computation computes attribute values for all remaining nodes. Apart from computing these values, it is alternatively also possible to assign them, following the same procedure as for the non-refined nodes. Whether such an assignment is sensible, depends on the meaning of the refined nodes. In fact, only some of them even represent understandable attacks or defenses, e.g., “Get onto Premises” or “Block Tag Reader”, while others only play the role of dummy placeholders, e.g., “Trick”. Whereas in the first case, it still seems reasonable to associate values with refined nodes, the latter case indicates that attribution of values might be more problematic. Providing values for refined nodes can not be performed without taking the corresponding context into account. Moreover, if an action is already sufficiently comprehensible such that a reliable value can be suggested, it does not need to be refined anymore. Hence, from the fundamental modeling idea behind ADTrees, assigning values to intermediate nodes seems questionable. Therefore, comparing such values with values computed with the bottom-up approach does not yield meaningful results. In the classroom case study we, therefore, opted not to ask for values of refined nodes. (Design Choice D4.4)

Comparing the values from Table 5.3 with the values calculated using the bottom-up approach supports this hypothesis. The comparison shows that the countermeasures are mistakenly usually not taken into account when node values are intuitively assigned. Therefore, such a comparison should not be performed. Figure 5.10 shows, for instance, that the **costs** for video looping is `Infinite` and thus impossible when guards are employed. In contrast to this, the estimated **costs** value given in Table 5.3, which is `Average` with a **confidence** level of 2. A similar disregard of countermeasures and subsequent counterattacks occurs when using attack trees instead of ADTrees. If all subtrees rooted in defense nodes are removed from Figure 5.10, the model no longer depicts that an attacker should keep possible defenses, such as “Barbed Wire” or “Monitor with Security Cameras”, in mind. In this case, the **costs** value of the cheapest scenario would be `Cheap`, with a **confidence** level of 4. The corresponding attack would consist of entering through the gate and the loading dock undetected. As a consequence, we, therefore, opted not to ask for values of refined nodes in the classroom case study. (Design Choice D6.2)

On the positive side, the bottom-up algorithm is flexible with regards to the specification of the attribute domain. In the minimal **costs** calculation performed in Section 5.3.6, we have chosen to use the attribute domain given by Equation (5.2) with discrete levels as value domain. Naturally, other value domains are conceivable to more accurately express how costly a combination of actions is. For instance, if the **costs** of two actions are `Cheap` and `Average`, we could simply choose to model their real **costs** by 10€ and 100€ and adapt the attribute domain accordingly. (Design Choice D2.2)

The bottom-up algorithm is also well-suited to include additional information we have about the attacker and the defender. For instance, it is possible to compute the minimal **difficulty** of an attack, assuming that the budget of the defender is limited to `Average`. This application of the ADTree methodology is another instance of bivariate questions that it can handle. From Table 5.3 and Figure 5.10 it can be seen that monitoring with biometric sensors as well as with security cameras would be too expensive for the defender. Hence, there would only be four remaining successful attack options:

- using the carpet on the barbs, climbing over the fence and entering through the door of the main building,
- using the carpet on the barbs, climbing over the fence and entering through the loading dock,
- entering through the gate and the door of the main building or
- entering through the gate and the loading dock.

Finally, adapting the predefined attribute domain can also help to obtain a more detailed answer. (Design Choice D6.1) As described in Section 4.5, any recursively defined attribute with a Boolean value domain can be used to enhance the expressiveness of the scenario analysis, provided a basic assignment is given of the Boolean attribute. For example, the minimal **costs** calculation, performed in Section 5.3.6, can be made more precise with the help of Boolean-valued attributes, which are well-suited to reason about hypothetical scenarios. Let us, for instance, consider the attribute that specifies whether or not **electricity** is needed. Then, the tree can be pruned to simulate what happens if there is a power outage. Since a power outage affects the attacker as well as the defender, the ADTree methodology is especially well-suited to model this kind of scenario. Pruning is performed according to the rules described in Definition 4.24. A similar technique can also be used to reason about parts of the scenario that satisfy a property of interest, for example, to reason about scenarios where the budget of the defender is limited.

To answer Question Q6: The bottom-up algorithm is a powerful tool for quantitative analysis. It is superior to simply estimating attribute values for refined nodes and can formally incorporate additional information about the attacker or the defender.

5.5 Design Choices and Guidelines for Case Studies

While performing the case studies, numerous design choices concerning the application of the ADTree methodology arose. Some options are outright inadmissible, some are clearly superior to others while for a third category the existence of several possible solutions shows the versatility of the ADTree methodology. The *correct* choices depend on the actual scenario, the security relevant questions to be answered, the modeling goals and, last but not least, the people performing the use case. None of the choices should be treated in isolation. Table 5.4 lists the design choices we addressed throughout the last section. The bold options indicate which of the choices we would select to perform use cases on an RFID attack scenario. This means that other choices might be more suitable for other scenarios or use cases. Table 5.4 is ordered chronologically, following the modeling process. It, therefore, deviates from the order of the design choices presented in the last section.

All design decisions have to be seen as part of four conflicting modeling goals: time, reusability, accuracy and simplicity. All four aspects have implications on the complexity of the analysis.

In modeling, *time* is always a concern. From a security point of view spending more time can only be beneficial, from an economic point of view that disregards security spending less time is efficient. As a consequence, the amount of time (and thus money) spent modeling always has to be justified by either allowing the analysis to be highly reusable or requiring a high degree of accuracy.

The use of model libraries is a natural approach to make graphical security models *reusable*. The SHIELDS project [SHI10b] has developed an online repository for (among others) attack trees. This library could be extended to also include ADTrees. While for attack scenarios a repository already exists, so far no one has spent the effort to collect and store probable attribute values. The degree of reusability may be lower for values compared to scenarios. Therefore, instead of storing concrete values, it may be preferable to store ranges of admissible values which serve as possible and not actual values. The more values are available, the more likely some information will be reusable. Using stored values may again conflict with other modeling goals, such as a fast scenario analysis (the stored node values most likely still have to be adapted) and, unless a computer tool is used, the visual appeal of

Step	Task	Design choices
1	**Create ADTree for scenario**	D1.1 Use **concise noun and verb**/detailed textual description as node label. D1.2 **Security expert**/system owner/player(s)/random person creates tree. D1.3 Create tree recursively, starting at the root node/**adapt tree from existing templates**. D1.4 **Hypothetical attacks and defenses**/All attacks and defenses. D1.5 Use the **same** level of detail across refinements/limit number of nodes. D1.6 Continuously improve trees/**freeze the tree structure at some time**. D1.7 Use incomplete/**sophisticated**/complete trees. D1.8 Allow/**disallow** pruning.
2	**Choose and describe attributes**	D2.1 **Provide**/do not provide attribute description similar to the ones given in Section 5.3.3. D2.2 Select value domains: **discrete**/real numbers/fuzzy sets/intervals/probability measures. D2.3 **Allow**/disallow the **disagree with node** attribute. D2.4 **Always**/sometimes use the meta-attribute **confidence**.
3	**Choose who estimates what**	D3.1 Who estimates attributes: attackers/defenders/**specialists**/random people. D3.2 Which nodes: according to role of a person/**according to background**/depending on attribute/all nodes.
4	**Estimate values**	D4.1 Evaluate meta-attributes for all attributes **separately**/together. D4.2 Consider nodes **in**/without context. D4.3 Allow/**disallow** different values for repetitive nodes. D4.4 **Do not estimate**/estimate values for refined nodes.
5	**Combine values**	D5.1 Apply standard combining procedure for **Categories 1–4**/for other categories. D5.2 Use **Formula** (5.1)/something else as standard procedure. D5.3 Use averaging/minimization/majority/**consensus meeting** for remaining categories. D5.4 **Restrict**/do not restrict the time in case of consensus meetings.
6	**Calculate values**	D6.1 Use **predefined**/other functions from a software tool or the literature. D6.2 Compare/**do not compare** with intermediate values.

Table 5.4: Work flow – Exemplary guidelines for the use of ADTrees for our case study. The bold options indicate which of the **choices** should be selected for use cases concerning the RFID goods management system.

the ADTrees is diminished, because the tree is cluttered.

The third conflicting modeling goal is the *accuracy* of the model and the values. It is necessary to find an acceptable compromise between the required time and necessary accuracy. Also, more accurate ADTrees and values reduce the reusability of the ADTree. Generally speaking, the coarser the value, the more raw data is available because more people feel comfortable with actually providing the value. Contrary to this, the finer the value, the more precise the result will be. However, if the scale of the demanded values is too fine, only experts may be able to estimate values. A coarse value range for a **costs** attribute would, for example, be `Low`, `Medium` and `High`, a fine grade would be present if the value was given as a real number expressing a monetary value, e.g., in euro.

The last modeling goal, is the *understandability* of the ADTree methodology. ADTrees have a simple tree structure which is a main advantage over approaches that use general graphs. Keeping the methodology simple makes it accessible to non-experts such as common users, developers, administrators and system owners. Naturally, simplicity is bought at the expense of accuracy.

As a result of the design choices and keeping the conflicting modeling goals in mind, we present the resulting six-step guideline which suggests a work flow and lists possible design choices that we recommend for applying the ADTree methodology when performing use cases. We have extended the steps from Figure 5.1, to be able to detail steps during the setup of the ADTree model.

1. *Create an ADTree for the scenario:* An ADTree is created using all available information and support tools. Provided the root of the tree is an attack node, the attack tree, obtained from an ADTree by ignoring all defense nodes and the corresponding subtrees, depicts the main attack scenario. All other nodes describe hypothetical defenses and counterattacks.

    - People with diverse knowledge about the system, e.g., developers, security experts, system owner and end users, should be involved in the tree creation.
    - Various materials, such as system specifications, floor plans, blueprints, work descriptions, attack tree libraries and attack patterns, should be used to create the tree.
    - The creation of the tree should be an iterative process which should end when there is mutual agreement between the involved parties. Modifying the tree after Step 3 has been performed should be avoided.
    - Node labels should preferably consist of a verb and a noun and concisely represent an attack or defense action.

2. *Choose and describe attributes:* Relevant attributes and meta-attributes are chosen, based on the security questions to be answered.

    - A clear, written description of chosen attributes and meta-attributes should be provided.
    - The description of each attribute and meta-attribute should include a domain specifying which values are used for quantification.

- In the case of discrete domains, a definition for each introduced category, such as `Small`, `Medium`, `Big`, should be provided.
- A user of the formalism should be allowed to express whether he disagrees with a part of the tree, e.g., by including the **disagree with node** attribute in the list of attributes.

3. *Choose who estimates attribute values:* Decide which and how many people estimate which values. Optimize the number of people with respect to the available resources.

   - In order to avoid errors and take into account different perspectives, more than one person should estimate attribute values.
   - Each person should obtain clear, written instructions detailing which values to estimate. It is not necessary that each person estimates the values for all nodes and/or all attributes, however, it should be mandatory that he provides the values he is assigned to estimate.

4. *Value estimation:* The persons that were selected in Step 3 estimate the values of the attributes chosen in Step 2 with the help of the support material identified in Step 1.

   - The values should be estimated based on knowledge about the scenario, the personal expertise and the attribute as well as the meta-attribute descriptions provided in Step 2. It should *not* be based on personal definitions.
   - When the bottom-up approach is used, the values should only be estimated for non-refined nodes.
   - The **confidence** meta-attribute should express a user's confidence into each of the provided attribute values. It should, therefore, be given for each estimated attribute value separately.
   - The attribute values should be estimated taking the node's context in the tree into account.

5. *Value combination:* When the attribute estimates from different people disagree, a combined value needs to be obtained. For a given node and a given attribute, this value should reflect the entirety of input values and be a good representative for them.

   - Nodes should be partitioned into categories, depending on clear objective criteria, such as percentage of **coverage**, as defined in Section 4.5.1.
   - The best way of deriving the representatives should be selected independently for each category for each node and attribute, e.g., use a suitable formula, the average or decide at a consensus meeting.
   - In case a consensus meeting is called for, its duration should be limited.

6. *Value calculation:* If the bottom-up approach is to be applied, suitable functions need to be chosen in order to calculate values for all the subtrees of a considered tree.

- The used functions should be in accordance with the attribute descriptions provided in Step 2.
- Scientific papers discussing attribute evaluation and existing attack tree tools can be consulted in order to define the appropriate functions.
- Estimated values of refined subtrees should not be compared with values resulting from the bottom-up algorithm.

To support the user in complying with the guidelines, we have developed a software tool. Digital models allow a *faster* tree creation, a *more detailed* analysis and *more accessible* visualization compared to models drawn by hand. Moreover, reusing digital models is effortless. In short, a software tool, like the one introduced in the following section, highly increases the efficiency of the ADTree methodology.

5.6 The ADTool

While performing case studies, it became apparent that security assessment using methodologies like ADTrees requires dedicated software tool support. The experience gained while performing the case studies helped to elicit requirements for a computer tool that supports ADTrees. Lack of such support results in numerous modeling mistakes and computational errors. In detail, when analyzing the conflicting modeling goals, the following challenges were identified.

1. ***In-depth formalism knowledge is required.*** General drawing tools are not suited to guide the creation of particular models, such as ADTrees. Thus, when using non-dedicated tools, the user himself needs to make sure that the models he constructs are syntactically correct and well-formed.

2. ***Drawing visually appealing trees is difficult.*** Trees drawn by hand or with the help of non-dedicated graphical tools do not have an appealing layout. Using dedicated tree drawing tools, such as the PSTricks package for LaTeX, is cumbersome and time-consuming.

3. ***Sharing and updating models is tedious and often impossible.*** Large real-life models, constructed manually or using non-dedicated tools, are hard to maintain, difficult to print and often cannot be displayed properly.

4. ***Verification of values provided by users is necessary.*** The consistency of values provided by several users needs to be thoroughly verified. For instance, all values should be expressed in the same units. Such checks are cumbersome, especially in the case of large-scale models.

5. ***Manual computations are highly error-prone.***

6. ***Dedicated tools are expensive and sometimes even incomplete.*** Commercial software for attack tree modeling, e.g., SecurITree [Ame12] or AttackTree+ [Iso11], is expensive. Academic tools, in turn, including Sea-Monster [Mel10] or AttackDog [Laz10], do not support quantitative analysis or do not allow us to interleave attacks and defenses.

To simultaneously overcome all of the difficulties mentioned above and to ease the use of ADTrees, the ADTool was developed. The ADTool combines the features offered by graphical tree representations with mathematical functionalities provided by ADTerms and attributes. It is the only freely available tool that allows us to create and quantitatively analyze security models uniformly integrating attack and defense components. This software tool was implemented by Piotr Kordy. It is accessible at `http://satoss.uni.lu/software/adtool`.

The goal of the ADTool is to provide security consultants as well as academic researchers with a user-friendly but rigorous application supporting the ADTree methodology. The ADTool facilitates the creation, display, sharing and management of large-scale ADTrees. Furthermore, it supports their quantitative analysis. Implemented attributes include: **costs**, **satisfiability**, **time** and **skill** level, for various owners, modalities and execution styles (see Section 4.4), the scenario's **probability of occurrence**, **reachability** of the main goal in less than x minutes, where x can be customized by the user and the maximal energy consumption. Since attack trees are a subclass of ADTrees, the tool can also be used for security modeling and assessment with the help of attack trees.

In the following, the most important features of the ADTool are presented. We show how the design of the tool addresses the challenges identified during the case studies.

The ADTool is easy to use One of the main features of the ADTool is its user-friendliness. When launching the tool, a default root node representing the main goal of an attacker (a red circle) is displayed automatically in the *ADTree Edit* window. The root can be changed to a defender's node (a green rectangle) with the help of the *Switch Attacker/Defender Roles* entry from the *Edit* menu. Starting from the root, an ADTree can be created with the help of the mouse. All options allowing to modify or refine a given node can be accessed by right-clicking the node, as shown in Figure 5.11. Alternatively, intuitive keyboard shortcuts can be used to create, alter or remove a subtree. All shortcuts are explained in the ADTool manual [12KSADTMan].

The ADTool guides the user by inserting only correct types of nodes and edges in the various places, complying with the graphical ADTree language, as described in [12KMRS]. Moreover, at each step of the model creation, only options consistent with the ADTree methodology are enabled. Thus, using the ADTool does not require an in-depth knowledge of the technical details of the underlying methodology.

The ADTool produces visually appealing trees An improved version of the Walker algorithm [Wal90] including several enhancements suggested by Buchheim et al. [BJL06] has been implemented in the ADTool to produce trees having an appealing layout.

Furthermore, when an ADTree is built, the corresponding ADTerm is immediately displayed in the *ADTerm Edit* window, see Figure 5.12. ADTerms form a compact, textual representation of ADTrees.

In order to link a textual model with its graphical counterpart, the shortest tree edit distance algorithm [DMRW09, PA11] has been implemented. It ensures that

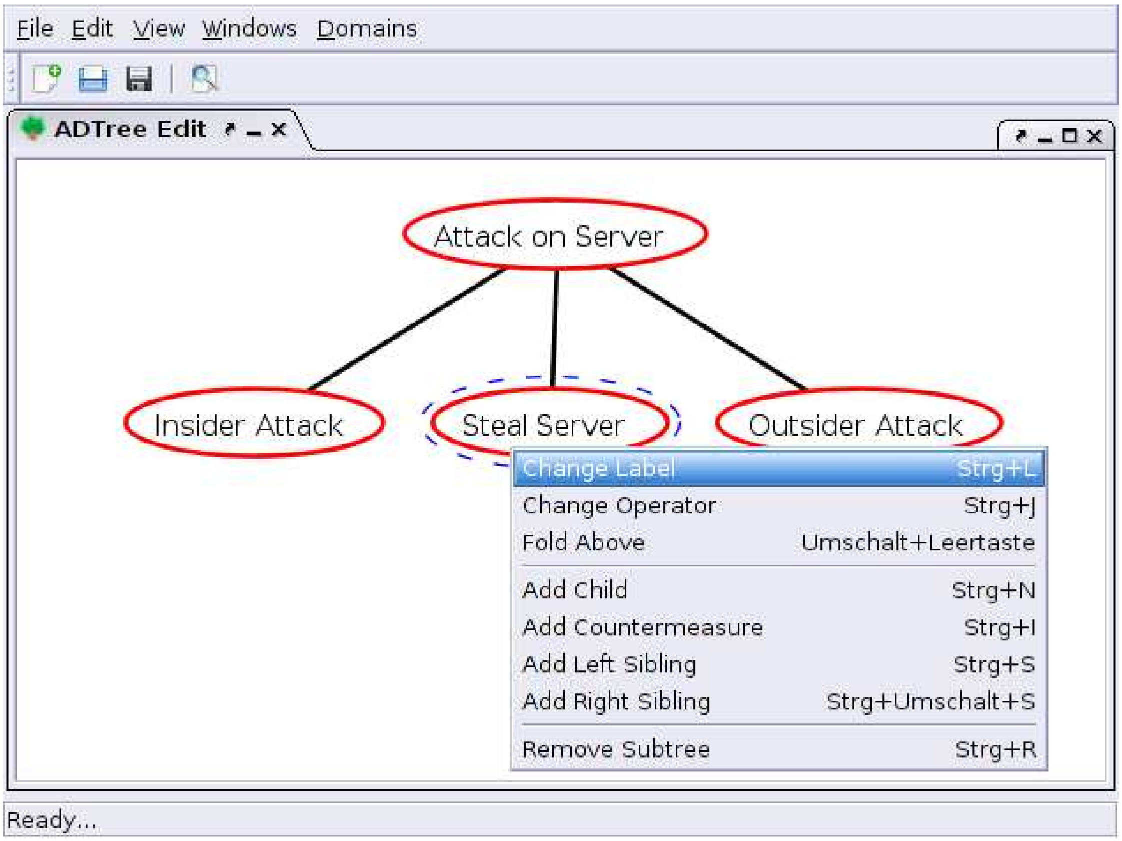


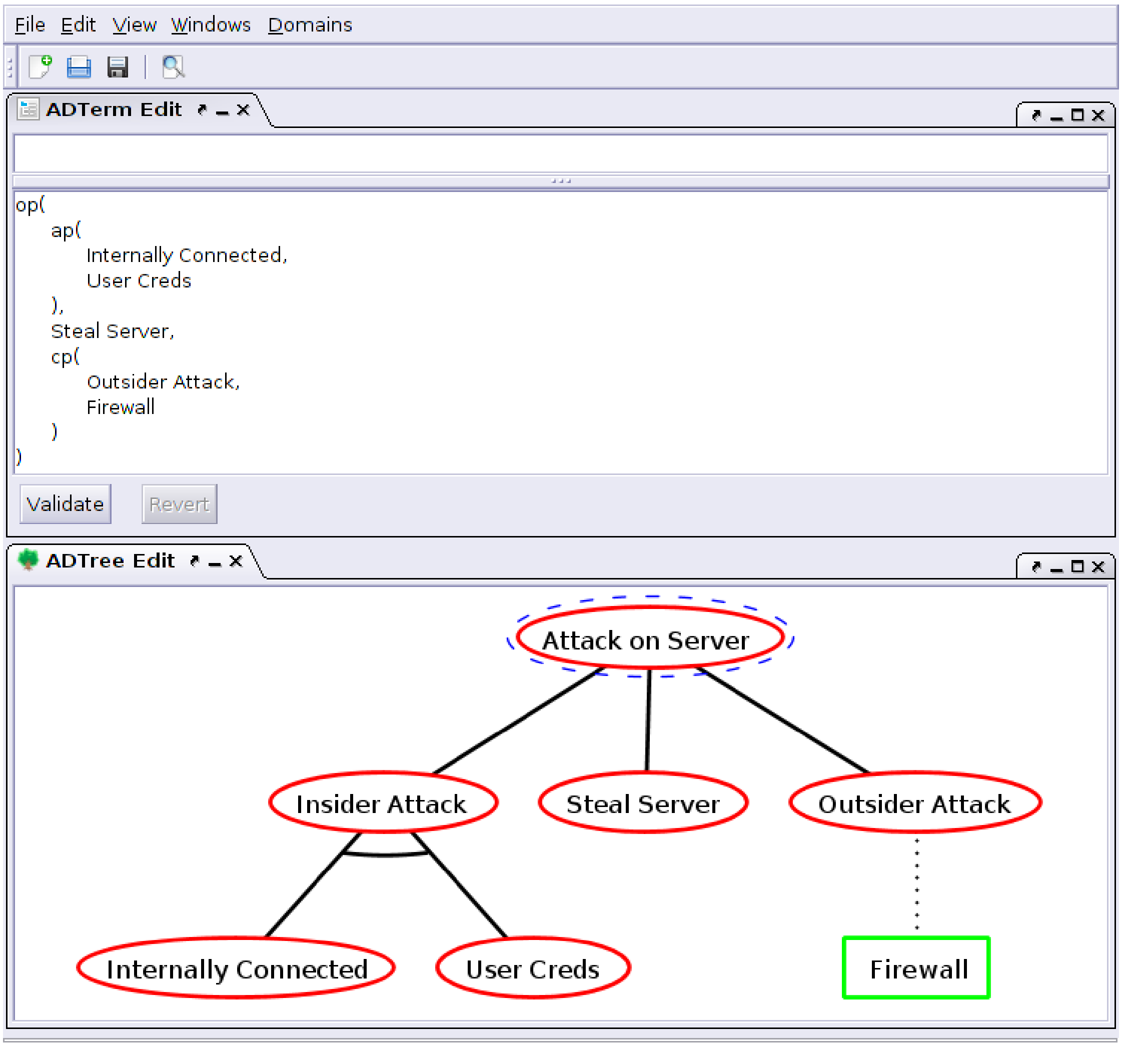
Figure 5.11: Creating an ADTree with the ADTool.

when an ADTerm is modified, the corresponding ADTree is adapted accordingly.

The ADTool provides advanced features for model management The ADTool is well-suited for analysis of large, real-life scenarios. Folding and expanding options make it possible to temporarily hide parts of a tree. This allows users to only focus on the displayed components, which is highly appreciated during industrial meetings and presentations. As different computers may have different screen sizes, several zooming features have also been implemented. They can be accessed using the scroll function of the mouse wheel, the standard keyboard shortcuts or the *Fit to Window* entry from the *View* menu.

ADTrees created with the ADTool can be saved as special .adt files. This enables their reuse and modification. Models can also be exported to vector graphics files (.pdf), raster graphics files (.png and .jpg) and LaTeX files (.tex). Resulting figures can be used as illustrations in scientific and industrial presentations, research papers and posters. Finally, large-scale trees can be printed on a specified number of pages, as illustrated in Figure 5.13.

The ADTool assists users in providing input values for computations The tool supports evaluation of eleven attributes, which can be accessed using the *Add Attribute Domain* entry from the *Domain* menu. After selecting an attribute, see Figure 5.14, the currently active ADTree is immediately decorated with default values. To initiate an attribute evaluation, a user customizes the input values of the relevant non-refined nodes. The tool ensures that the provided values are consistent. This is especially important when several specialists supply values for different



```
op(
    ap(
        Internally Connected,
        User Creds
    ),
    Steal Server,
    cp(
        Outsider Attack,
        Firewall
    )
)
```

Figure 5.12: An ADTree modeled in the ADTool.

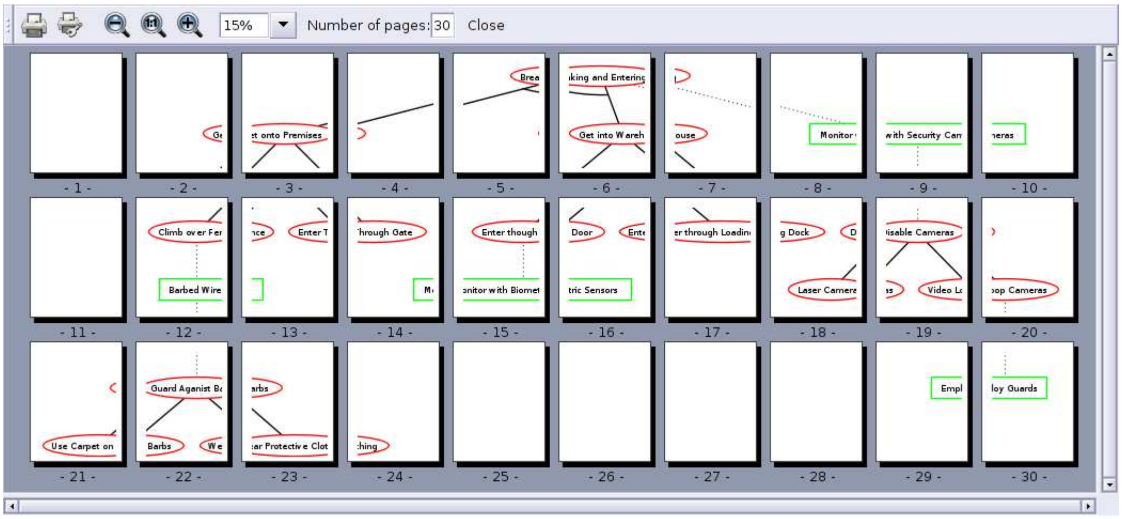
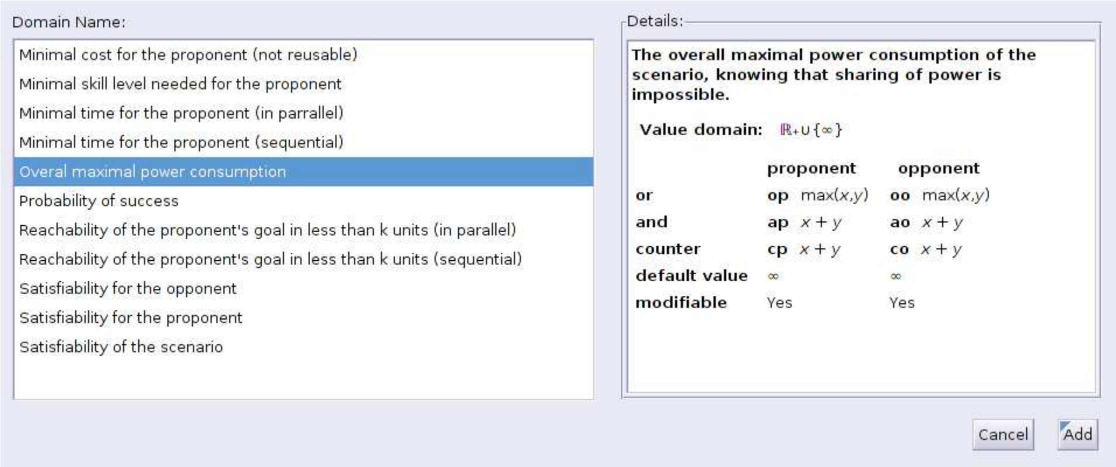


Figure 5.13: Large-scale printing in the ADTool.

parts of the tree. The application does not accept values which do not belong to the specified value domain. Furthermore, nodes labeled with the same name, i.e., representing the same actions, automatically receive the same value. This design choice is consistent with the ADTree methodology, as specified in this thesis. A table giving an overview of all non-refined nodes is accessible from the *Windows* menu after selecting *Valuations View*, see Figure 5.15. It can be used to easily modify the input values and is helpful when modeling large scenarios.



Domain Name:

- Minimal cost for the proponent (not reusable)
- Minimal skill level needed for the proponent
- Minimal time for the proponent (in parrallel)
- Minimal time for the proponent (sequential)
- Overal maximal power consumption
- Probability of success
- Reachability of the proponent's goal in less than k units (in parallel)
- Reachability of the proponent's goal in less than k units (sequential)
- Satisfiability for the opponent
- Satisfiability for the proponent
- Satisfiability of the scenario

Details:

The overall maximal power consumption of the scenario, knowing that sharing of power is impossible.

Value domain: $\mathbb{R}_+ \cup \{\infty\}$

	proponent	**opponent**
or	**op** $\max(x,y)$	**oo** $\max(x,y)$
and	**ap** $x+y$	**ao** $x+y$
counter	**cp** $x+y$	**co** $x+y$
default value	∞	∞
modifiable	Yes	Yes

Figure 5.14: Attribute selection in the ADTool.

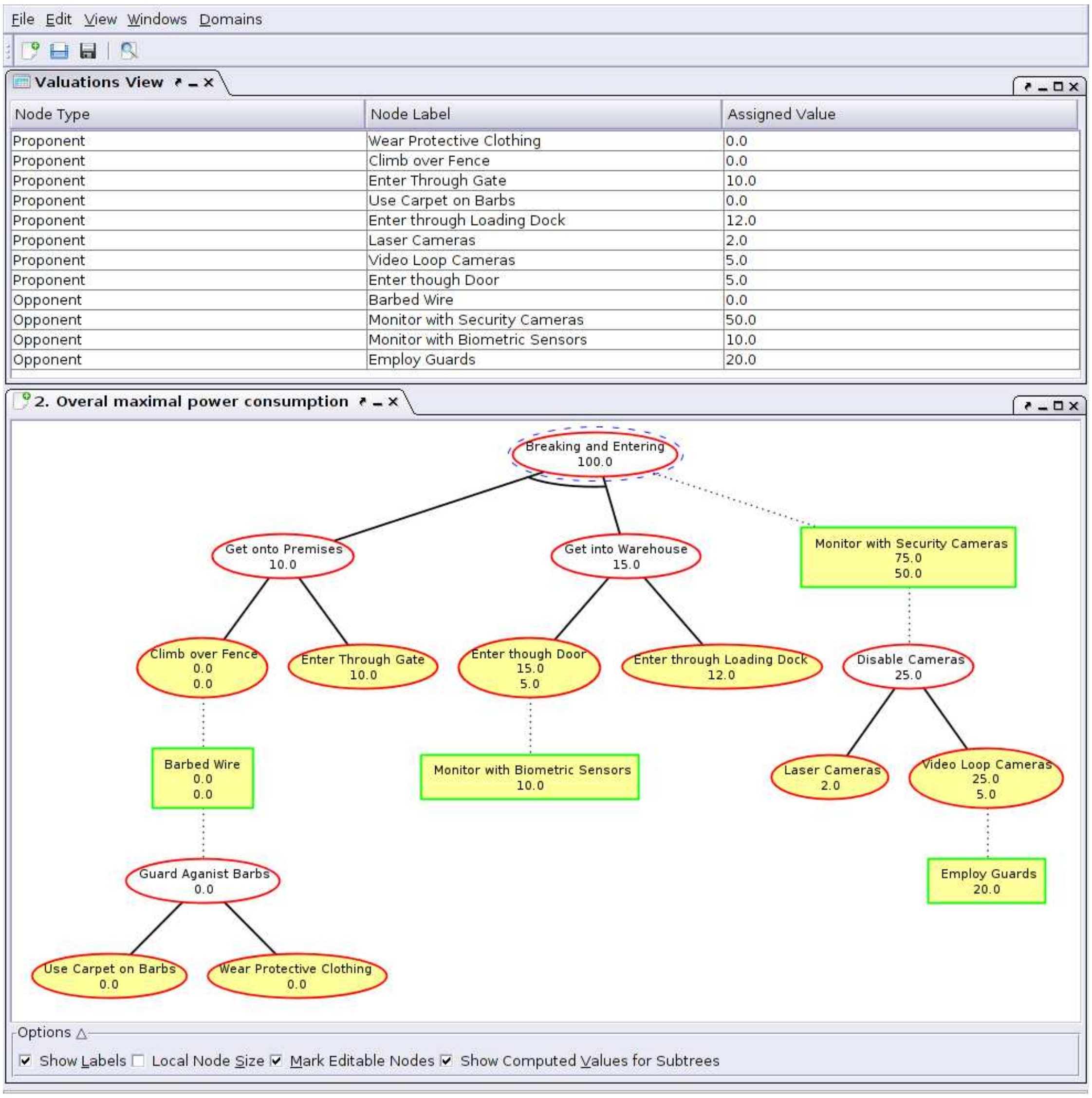
The ADTool automates computations on ADTrees The bottom-up algorithm (see Definition 4.4) for the evaluation of attributes has been implemented in the ADTool. Given an attribute domain and values for all non-refined nodes, the values for the refined nodes are calculated automatically. By restricting the user input and by automating computations, calculation errors are avoided.

Furthermore, the tool can easily be extended with new attribute domains. For this purpose, a new class implementing the set of possible values and the necessary operators needs to be created and compiled. No recompilation or other modifications of the program are required thanks to the use of Java reflection.

The ADTool is free and easy to access The ADTool is free software that runs on all common operating systems (Windows, Linux, Mac OS). It is implemented in Java and requires JDK 6 or later. Additionally, the ADTool depends on the following free libraries: abego TreeLayout [abe11], implementing an efficient and customizable tree layout algorithm in Java, and InfoNode Docking Windows [NNL09], a pure Java Swing based docking windows framework.

The ADTool is available for download[10]. It can also be launched as an online application that uses Java Web Start technology [Ora13]. Finally, a manual, explaining step by step how to use the tool, is available [12KSADTMan].

[10] http://satoss.uni.lu/software/adtool/



Node Type	Node Label	Assigned Value
Proponent	Wear Protective Clothing	0.0
Proponent	Climb over Fence	0.0
Proponent	Enter Through Gate	10.0
Proponent	Use Carpet on Barbs	0.0
Proponent	Enter through Loading Dock	12.0
Proponent	Laser Cameras	2.0
Proponent	Video Loop Cameras	5.0
Proponent	Enter though Door	5.0
Opponent	Barbed Wire	0.0
Opponent	Monitor with Security Cameras	50.0
Opponent	Monitor with Biometric Sensors	10.0
Opponent	Employ Guards	20.0

☑ Show Labels □ Local Node Size ☑ Mark Editable Nodes ☑ Show Computed Values for Subtrees

Figure 5.15: Attribute evaluation in the ADTool.

6

Formal Applications of ADTrees

In Chapters 2, 3 and 4, we designed the syntax, the semantics and a procedure for quantitative analysis for the ADTree methodology. In the previous section, we have seen that our approach is applicable in practice. In this chapter we explain that the rigorous formalization enables us to link ADTrees to other well-studied research areas. Having the view of different syntaxes and semantics provides us with different capabilities to strengthen, apply and draw conclusions about the ADTree methodology. The formal definitions enable us to prove theorems about the functionality and capabilities of ADTrees. In this chapter we outline several examples of such applications of the formal model.

Concretely, we show that propositional ADTrees increase the expressiveness of the model compared to propositional attack trees without increasing the complexity (Section 6.1). Then we show how to combine ADTrees and Bayesian networks to be able to handle computation in the presence of conditionally dependent nodes (Section 6.2). And finally, we summarize a connection between propositional ADTrees and a specific class of games that frequently occur in game theory (Section 6.3).

6.1 Complexity Considerations of the De Morgan Semantics

The objective of this section is to compare the computational complexity of the propositional ADTerms language with the computational complexity of the propositional attack terms language.

6.1.1 Positive, Negative and Monotone Boolean Functions

In order to compare the computational complexity of ADTrees, we first analyze the classes of Boolean functions represented by both languages. Of particular importance for our studies are positive, negative and monotone Boolean functions. This section makes use of the definition of a configuration, see Definition 3.6, and our related conventions. A more detailed introduction can be found in [PK11].

Definition 6.1 (Positive, negative and monotone Boolean functions in one variable) Let R be a countable set of propositional variables, let f be a Boolean function with finite domain $D \subseteq R$ and let $x \in D$ be a propositional variable.

- f is positive in x if $f(\mathbf{x}, 0) \leq f(\mathbf{x}, 1)$, for all $\mathbf{x} \in \{0,1\}^{D\setminus\{x\}}$,
- f is negative in x if $f(\mathbf{x}, 0) \geq f(\mathbf{x}, 1)$, for all $\mathbf{x} \in \{0,1\}^{D\setminus\{x\}}$,

- f is monotone in $x \in D$ if it is either positive or negative in x.

In this definition we use the notation that for a Boolean function $f\colon D \to \{0,1\}$ the expression $f(\mathbf{x}, i)$, where $\mathbf{x} \in \{0,1\}^{D\setminus\{x\}}$ expresses that i is assigned to x (which is not necessarily the last variable). Note that if $x \in R$ does not occur in the domain of a Boolean function f, then f is insensitive to the values assigned to x. In this case, we say that f is positive, negative and monotone in x.

Definition 6.2 (Positive, negative and monotone Boolean functions) A Boolean function f is positive (resp. negative) if it is positive (resp. negative) in every variable $x \in R$. It is monotone if it is either positive in all variables or negative in all variables.

Alternatively, a partial ordering can be defined on $\{0,1\}^D$ by setting $\mathbf{u} \leq \mathbf{v}$ if and only if $\mathbf{u} \vee \mathbf{v} = \mathbf{v}$. This definition is equivalent to writing $\mathbf{u} \leq \mathbf{v}$ if and only if $u_i \leq v_i$ for every $i \in D$.

The following lemma shows that the classes of Boolean functions that are positive in a variable as well as those that are negative in a variable are closed under conjunction and disjunction.

Lemma 6.3 *Let f and g be Boolean functions.*

- *If f and g are positive in x, then $f \wedge g$ and $f \vee g$ are positive in x.*
- *If f and g are negative in x, then $f \wedge g$ and $f \vee g$ are negative in x.*

Proof. Both statements follow directly from Definition 3.9 and the monotonicity of minimization and maximization. □

Note, however, that the results from Lemma 6.3 do generally not hold for monotone Boolean functions. This can be seen as follows:

Example 6.4 The Boolean function $f(x, y) = x \wedge \neg y$ is positive in x and negative in y. Thus, f is monotone in x and monotone in y. For similar reasons, the Boolean function $g(x, y) = y \wedge \neg x$ is also monotone in x and monotone in y. However, it can easily be checked that the function $f \vee g$ is neither monotone in x nor in y.

Next we show that Boolean functions that are monotone in a variable are closed under negation.

Lemma 6.5 *Let f be a Boolean function and let $x \in R$ be a propositional variable. If f is positive (resp. negative) in x, then $\neg f$ is negative (resp. positive) in x.*

Proof. Let us assume that f is positive in x and let D be the domain of f. If $x \notin D$, then, by convention, f is positive in x and $\neg f$ is negative in x. If $x \in D$, then from the positivity of f in x, we have that for the projected configurations $f(\mathbf{x}^{\downarrow D\setminus\{x\}}, 0) \leq f(\mathbf{x}^{\downarrow D\setminus\{x\}}, 1)$, for all $\mathbf{x} \in \{0,1\}^D$. Therefore,

$$(\neg f)(\mathbf{x}^{\downarrow D\setminus\{x\}}, 0) = 1 - f(\mathbf{x}^{\downarrow D\setminus\{x\}}, 0) \geq 1 - f(\mathbf{x}^{\downarrow D\setminus\{x\}}, 1) = (\neg f)(\mathbf{x}^{\downarrow D\setminus\{x\}}, 1).$$

This shows that $\neg f$ is negative in x. The proof for the other case is similar. □

Note that Lemma 6.5 holds because logical negation $\neg$ reverses the order, i.e., for $a, b \in \{0, 1\}$, we have $a \leq b$ if and only if $\neg a \geq \neg b$.

Corollary 6.6 *The Boolean functions that are monotone in a variable are closed under negation.*

This is crucial in Section 6.1.4, where we generalize this result from Boolean functions to De Morgan valuations. This extension mimics the extension of the propositional semantics to the De Morgan semantics, see Section 3.2.

From Lemmas 6.3 and 6.5, we deduce the following result.

Corollary 6.7 *If f and g are two Boolean functions, such that f is positive (resp. negative) in a variable x and g is negative (resp. positive) in x, then the Boolean function $f \wedge \neg g$ is positive (resp. negative) in x.*

6.1.2 Expressiveness of Propositional ADTerms

In order to compare the propositional attack–defense trees with the propositional attack trees, we start by analyzing the language of propositional attack terms. Attack terms constitute formal representations of attack trees. As in Section 3.5.2, let $X^G = \{x_b \mid b \in \mathbb{B}^{\mathrm{p}}\}$ be the set of propositional variables that, in the propositional semantics, correspond to the basic actions of the proponent. Recall from Definition 3.7 that the propositional variables x_b represent the Boolean indicator functions f_b. Using this notation, every propositional attack term is a formula generated by the following grammar $\mathcal{AT}$, using the start symbol P:

$$P: \quad X^G \quad | \quad P \vee P \quad | \quad P \wedge P. \qquad (\mathcal{AT})$$

The following theorem characterizes propositional attack terms using Boolean functions.

Theorem 6.8 *Boolean functions represented by propositional attack terms are positive.*

Proof. Consider the grammar $\mathcal{AT}$. The Boolean function represented by $x_b \in X^G$ is positive. The positivity of the Boolean functions represented by $P \vee P$ and $P \wedge P$ is a direct consequence of Lemma 6.3. □

Since all positive Boolean functions have at least one positive disjunctive normal form [CH11], it is easy to see that for all positive Boolean functions except the tautology there exists a corresponding propositional attack term.

In order to characterize the language of propositional ADTerms, we extend the grammar $\mathcal{AT}$ to the grammar $\mathcal{ADT}$. We have already seen in Theorem 3.34 that $\mathcal{ADT}$ generates all propositional ADTerms. We let $Y^G = \{x_b \mid b \in \mathbb{B}^{\mathrm{o}}\}$ be the set of propositional variables that, in the propositional semantics, correspond to the basic actions of the opponent. Using the start symbols P and N and the production rules

$$\begin{array}{llllllll} P: & X^G & | & P \vee P & | & P \wedge P & | & P \wedge \neg N \\ N: & Y^G & | & N \vee N & | & N \wedge N & | & N \wedge \neg P, \end{array} \qquad (\mathcal{ADT})$$

we obtain the two grammars $\mathcal{ADT}_P$ and $\mathcal{ADT}_N$, respectively. We define $\mathcal{ADT} = \mathcal{ADT}_P \cup \mathcal{ADT}_N$. Like before, the terminal symbols from X^G and Y^G could also be seen as the Boolean indicator function f_b.

In order to prove Theorem 6.10, we use the following Lemma.

Lemma 6.9 *Consider the grammar $\mathcal{ADT}$. Every Boolean function represented by a formula of the form P (resp. N) is*

- *positive (resp. negative) in every variable x_b, for $b \in \mathbb{B}^{\mathrm{p}}$,*
- *negative (resp. positive) in every variable x_b, for $b \in \mathbb{B}^{\mathrm{o}}$.*

Proof. We provide a proof for the cases of P and N, simultaneously. We reason by induction over the length of the constructed expression. If $\psi = x^{\mathrm{p}} \in \mathbb{B}^{\mathrm{p}}$, then the Boolean function generated from the start symbol P is positive in x^{p}. According to our convention, P is also positive in every other variable in $\mathbb{B}^{\mathrm{p}}$ as well as negative in every $x_b \in \mathbb{B}^{\mathrm{o}}$. Following a similar reasoning, if $\phi = x^{\mathrm{o}} \in \mathbb{B}^{\mathrm{p}}$, then the Boolean function generated from the start symbol N is negative in x^{o}.

Now, consider a formula ψ which is not a single variable and assume that the lemma holds for all formulas derivable from P or N with shorter derivation. If ψ is of the form $P \vee P$ or $P \wedge P$, the result follows from Lemma 6.3 and the induction hypothesis. Similarly, if ϕ is of the form $N \vee N$ or $N \wedge N$, the result follows from Lemma 6.3 and the induction hypothesis. If ψ is of the form $P \wedge \neg N$, the result follows from Corollary 6.7 and the induction hypothesis for N. Similarly, if ϕ is of the form $N \wedge \neg P$. □

The following theorem characterizes propositional ADTerms using Boolean functions.

Theorem 6.10 *Boolean functions represented by propositional ADTerms are in every variable either positive or negative.*

Proof. Since the sets $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are disjoint, we can conclude that every formula generated by $\mathcal{ADT}$ represents a Boolean function that is monotone in every variable. This concludes the proof. □

Note that the assumption that $\mathbb{B}^{\mathrm{p}}$ and $\mathbb{B}^{\mathrm{o}}$ are disjoint is crucial. Without this assumption, Lemma 6.9 would not hold, as shown in Example 6.4. The converse of the theorem does not hold since, for example, ADTerms cannot represent constant functions.

6.1.3 From Propositional ADTerms to Propositional Attack Terms

Since every attack term is also an ADTerm, it is obvious that all Boolean functions represented by propositional attack terms can be represented by propositional ADTerms. In this section, we show that the converse holds as well.

Theorem 6.11 *Let f be a Boolean function with domain D and let $x \in D$. We define a Boolean function g with the same domain, as follows*

$$g(\mathbf{x}, 0) = f(\mathbf{x}, 1) \quad \textit{and} \quad g(\mathbf{x}, 1) = f(\mathbf{x}, 0),$$

for all $\mathbf{x} \in \{0,1\}^{D\setminus\{x\}}$. The function g is positive (resp. negative) in x if and only if the function f is negative (resp. positive) in x.

Proof. If f is positive in $x \in D$, then, for all $\mathbf{x} \in \{0,1\}^{D\setminus\{x\}}$, we have $g(\mathbf{x}, 1) = f(\mathbf{x}, 0) \le f(\mathbf{x}, 1) = g(\mathbf{x}, 0)$, i.e., g is negative in x. The other case is similar. □

Note that the functions f and g in Theorem 6.11 are not equivalent in the sense of Definition 3.10, but there is a one-to-one correspondence between their satisfying assignments, i.e., between the elements of the sets of $f^{-1}(\{1\})$ and $g^{-1}(\{1\})$.

It follows from Theorem 6.11 that every Boolean function that is in every variable either positive or negative can always be transformed to a positive form. Moreover, Lemma 6.9 guarantees that such a transformation is linear in the size of the function's domain. Consequently, whenever we want to reason about a propositional ADTerm, we can analyze a positive Boolean function instead of a monotone one. Hence, the following result holds.

Corollary 6.12 *Propositional ADTerms represent positive Boolean functions.*

This proves that the language of propositional ADTerms and the language of propositional attack terms both represent positive Boolean functions. Practical consequences of this fact are discussed in Section 6.1.5.

6.1.4 Generalization to De Morgan ADTerms

An important feature of ADTerms is that they can be equipped with different semantics. The previous results focus on the propositional semantics, where basic actions are allowed to take propositional values and ADTerms over the set of variables D represent Boolean functions of the form $\{0,1\}^D \to \{0,1\}$. In this section we show that the transformation from ADTerms to attack terms, presented in Section 6.1.3 for the propositional semantics, also applies to all semantics induced by a De Morgan lattice. To distinguish between positive, negative and monotone functions of the form $\{0,1\}^D \to A$, over a De Morgan lattice $\langle A, +, \times, \neg\rangle$, the set A must exhibit a partial order that behaves monotonically under the operations $+$ and $\times$. Recall the monotonic partial order $\preceq$ defined on De Morgan lattices given in Equivalence (3.1): for all $a, b \in A$ it holds that $a \preceq b$ if and only if $a + b = b$.

To validate Lemma 6.5 for the De Morgan semantics, we show that in De Morgan lattices the order $\preceq$ is indeed reversed under negation.

Lemma 6.13 *In a De Morgan lattice we have $a \preceq b$ if and only if $\neg b \preceq \neg a$.*

Proof. Assume that $a \preceq b$, i.e., $a + b = b$. It follows from the definition of a De Morgan lattice that $\neg b = \neg(a + b) = (\neg a) \times (\neg b)$. Moreover, in every lattice we have $b = a \times b$ if and only if $a = a + b$, see [DP90]. Therefore, we conclude that $\neg b = (\neg a) \times (\neg b)$ if and only if $\neg a = (\neg a) + (\neg b)$. This proves that $a \preceq b$

implies $\neg b \preceq \neg a$. Conversely, assume $\neg b \preceq \neg a$. From the first part of this proof we know that $\neg b \preceq \neg a$ implies $\neg(\neg a) \preceq \neg(\neg b)$ and, therefore, we have $a \preceq b$. □

Consider a De Morgan lattice $\langle A, +, \times, \neg\rangle$, two finite sets $D, U \subseteq R$ of variables and two De Morgan valuations $f\colon \{0,1\}^D \to A$ and $g\colon \{0,1\}^U \to A$ as defined in Definition 3.16. Recall from Definition 3.2 that two De Morgan valuations f and g are said to be equivalent if and only if for every $\mathbf{x} \in \{0,1\}^{D\cup U}$ we have $f(\mathbf{x}^{\downarrow D}) = g(\mathbf{x}^{\downarrow U})$.

We define positive, negative and monotone De Morgan valuations by modifying Definition 6.1 in the obvious way: In the definition we replace Boolean functions by De Morgan valuations and the order $\leq$ by $\preceq$.

Then, Corollary 6.7 and Theorem 6.11 still hold if Boolean functions are replaced by De Morgan valuations. This means that the transformation of propositional ADTerms to propositional attack terms, described in Section 6.1.3 actually holds for De Morgan ADTerms. In other words, we can apply Theorem 6.11 and reduce De Morgan ADTerms to De Morgan attack terms. We summarize by generalizing Corollary 6.12:

Theorem 6.14 *De Morgan ADTerms represent positive De Morgan valuations.*

6.1.5 Consequences for Complexity Considerations

The results in Sections 6.1.3 and 6.1.4 help us to compare attack trees and ADTrees on a computational level. We formally proved that, under a semantics induced by a De Morgan lattice, both models represent positive valuations. Since they can be transformed into each other, they exhibit the same complexity on computational tasks. This has several consequences.

First, ADTrees extend attack trees to a richer model without increasing the computational complexity, provided that their semantics is induced by a De Morgan lattice.

This result can be applied, for instance, to query evaluation on ADTerms. A *query* on ADTerms is a function $\tilde{Q}\colon \mathbb{T}_\Sigma \to \mathcal{A}$ which assigns to every ADTerm t an element $\tilde{Q}(t) \in \mathcal{A}$ called the *answer* for $\tilde{Q}$ on t.

Example 6.15 Consider the function $\tilde{Q}_{\text{sat}}$, which assigns `true` to an ADTerm t if the corresponding Boolean function modeled by the **satisfiability** attribute admits at least one satisfying assignment and otherwise `false`. In other words, $\tilde{Q}_{\text{sat}}(t)$ is `true` if there exists a configuration $\mathbf{x}$, such that $f(\mathbf{x}) = 1$. Otherwise it is `false`. The function $\tilde{Q}_{\text{sat}}$ is an example of a query on ADTerms which is derived from the **satisfiability** attribute sat.

Second, when a semantics induced by a De Morgan lattice is used, the complexity of query evaluation on ADTerms is the same as the corresponding complexity on attack terms.

This means, when a semantics induced by a De Morgan lattice is used, a query can efficiently be solved on ADTerms if and only if it can efficiently be solved on attack terms. For example, when the propositional semantics is used, satisfiability checks on ADTerms can be performed in constant time because all positive Boolean functions are satisfiable.

Theorem 6.11 and subsequent considerations in Section 6.1.4 show that, for a large class of semantics, we can effectively transform ADTerms to attack terms. Therefore, we obtain the following implication.

When using a semantics induced by a De Morgan lattice, ADTerms can always be processed by algorithms developed for attack terms.

Finally, knowing that not all Boolean functions are positive and taking into account Corollary 6.12, we deduce that there exist Boolean functions which cannot be represented by any propositional ADTerm.

The propositional language defined by propositional ADTerms is a proper subset of the positive Boolean functions since there is no ADTerm corresponding to the tautology.

6.2 ADTrees and Dependent Nodes

In this section, we develop means to *combine* the ADTree methodology with Bayesian networks in order to evaluate probabilistic measures on attack–defense scenarios involving dependent actions. In our approach, the Bayesian network *does not replace* the information represented by the structure of an ADTree, but complements it with *additional probabilistic dependencies* between attack steps, which cannot be depicted using AND-OR relations. Hence, we keep both models separated, which allows us to reuse existing expertise and previous models.

To combine the two methodologies, we proceed as follows. In Section 6.2.1, we introduce an example showing consequences of dropping the assumption that all actions in a model occur independently. We then introduce Bayesian networks and Bayesian networks for ADTrees with the help of this example, in Section 6.2.2, before we explain how to compute probabilities of dependently occurring actions on Bayesian networks for ADTrees, in Section 6.2.3. The following Sections 6.2.4–6.2.7 explain how to speed up the calculation of the probabilities in practice. We conclude the treatment of Bayesian networks for ADTrees by extending the framework to dependent success probabilities, in Section 6.2.8.

6.2.1 Computation of Independently Occurring Actions

In this section, we provide an example that is well-suited to illustrate how additional dependencies can be incorporated into the ADTree framework.

Example 6.16 In order to infect a computer with a virus, the attacker needs to ensure that the virus file is accessible from the targeted computer and that a user of the computer executes the file. There are two possibilities to make the file accessible. An attacker can send the virus in an email attachment or distribute a USB stick to the computer user. The computer user, on his part, can protect himself against a virus with an anti-virus program (AV). For the AV to be effective, it needs to be installed and it needs to be running. A resourceful attacker, in turn,

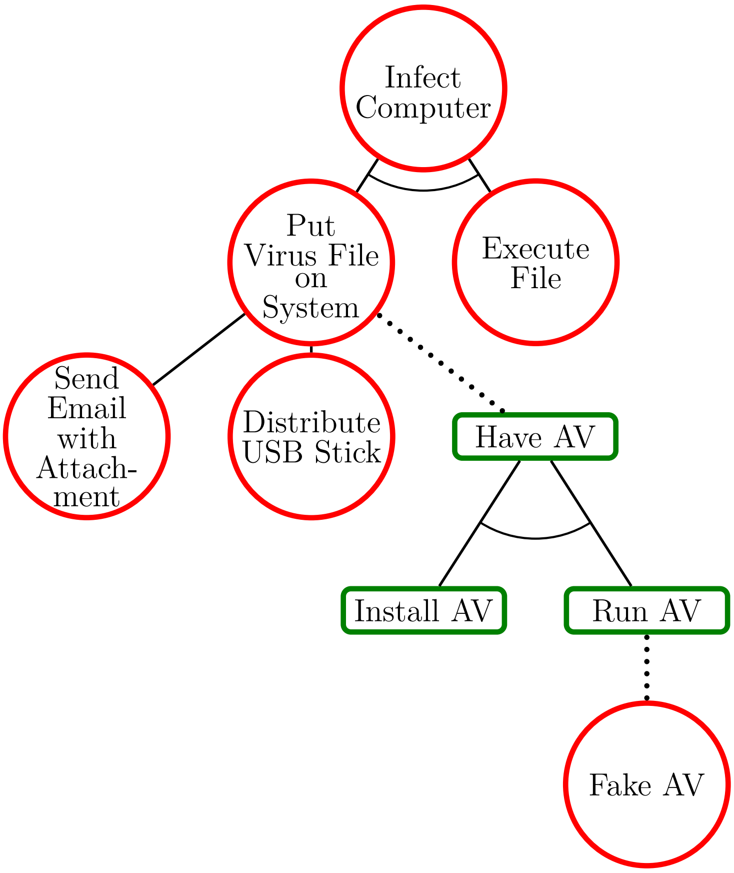


Figure 6.1: An ADTree for infecting a computer and protecting against infection using an anti-virus program (AV).

could attack the AV by distributing a fake version of an AV that only pretends to be running. The described attack–defense scenario is depicted in Figure 6.1. For this tree, the attacker is the proponent and the defender is the opponent. In the ADTree, the proponent is the attacker and his basic actions are EA ("Send Email with Attachment"), DU ("Distribute USB Stick"), FA ("Fake AV") and EF ("Execute File"). Consequently, the opponent is the defender and his basic actions are IA ("Install AV") and RA ("Run AV"). The ADTerm corresponding to the ADTree from Figure 6.1 is given by

$$t = \wedge^{\mathrm{p}}(\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{EA}, \mathrm{DU}), \wedge^{\mathrm{o}}(\mathrm{IA}, \mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{FA}))), \mathrm{EF}). \tag{6.1}$$

We recall the use of the propositional semantics that makes use of Boolean functions, as defined in Section 3.1. Following Remark 3.22, we apply Definition 3.20 to construct the propositional ADTerm associated to t in the form of Boolean functions.

Example 6.17 We continue Example 6.16 by constructing the propositional ADTerm for the ADTree given by Equation 6.1. First, we create the propositional variables x_{EA}, x_{DU}, x_{IA}, x_{RA}, x_{FA} and x_{EF} for the basic actions in t. We then construct a Boolean function $f_b\colon \{0,1\}^{\{x_b\}} \to \{0,1\}$ with $f_b(x_b = v) = v$, for $v \in \{0,1\}$ and $b \in \{\mathrm{EA}, \mathrm{DU}, \mathrm{IA}, \mathrm{RA}, \mathrm{FA}, \mathrm{EF}\}$. Since all of these Boolean functions are, in fact, indicator functions, we abuse notation and write x_b instead of f_b. This allows us to express the ADTerm t in the propositional semantics as Boolean function:

$$t_{\mathcal{P}} = f_t \equiv ((x_{\mathrm{EA}} \vee x_{\mathrm{DU}}) \wedge \neg(x_{\mathrm{IA}} \wedge (x_{\mathrm{RA}} \wedge \neg x_{\mathrm{FA}}))) \wedge x_{\mathrm{EF}}. \tag{6.2}$$

Remark 6.18 The attack–defense scenario described in the previous example is used as the running example throughout this section. Since the root of the ADTree in Figure 6.1 represents an attack goal (infecting a computer), this section is concerned with attacks on a system. In the case of an ADTree having a defensive root node, we would talk about defenses for a system.

To be able to compare ADTrees with Bayesian networks for ADTrees in Section 6.2.7, we illustrate the standard bottom-up algorithm on the attribute domain that models the **probability of occurrence**, which uses the same operators as the **probability of success** already discussed in Section 4.4.3. We use the attribute domain:

$$A_{\mathrm{occ}} = ([0,1], \oplus, \otimes, \oplus, \otimes, \circledast, \circledast),$$

where $\oplus$ and $\otimes$ denote probabilistic addition and multiplication of independent events and $\circledast$ denotes the probabilistic complement. Note that this attribute domain is used to compute the **probability of occurrence**, assuming that all actions are independently executed.

Example 6.19 Let t be as in Equation 6.1 and let us assume the basic assignment satisfies the following:

$$\begin{aligned} &\beta_{\mathrm{occ}}(\mathrm{EF}) = 0.393, \quad \beta_{\mathrm{occ}}(\mathrm{DU}) = 0.35, \quad \beta_{\mathrm{occ}}(\mathrm{FA}) = 0.3, \\ &\beta_{\mathrm{occ}}(\mathrm{EA}) = 0.48, \quad\ \ \beta_{\mathrm{occ}}(\mathrm{RA}) = 0.54, \quad \beta_{\mathrm{occ}}(\mathrm{IA}) = 0.6. \end{aligned}$$

Then $\alpha_{\mathrm{occ}}(t)$ is computed as follows:

$$\begin{aligned} \alpha_{\mathrm{occ}}(t) &= \alpha_{\mathrm{occ}}(\wedge^{\mathrm{p}}(\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{EA}, \mathrm{DU}), \wedge^{\mathrm{o}}(\mathrm{IA}, \mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{FA}))), \mathrm{EF})) \\ &= \otimes(\circledast(\oplus(\beta_{\mathrm{occ}}(\mathrm{EA}), \beta_{\mathrm{occ}}(\mathrm{DU})), \otimes(\beta_{\mathrm{occ}}(\mathrm{IA}), \circledast(\beta_{\mathrm{occ}}(\mathrm{RA}), \beta_{\mathrm{occ}}(\mathrm{FA})))), \\ &\qquad \beta_{\mathrm{occ}}(\mathrm{EF})) \\ &= \otimes(\circledast(0.48 + 0.35 - 0.48 \cdot 0.35, \otimes(0.6, (0.54 \cdot (1 - 0.3)))), 0.393) \\ &= \otimes(\circledast(0.662, (0.6 \cdot 0.378)), 0.393) \\ &= \otimes(0.662 \cdot (1 - 0.2268), 0.393) \\ &\approx 0.5119 \cdot 0.393 \approx 0.2012. \end{aligned}$$

Hence, the probability that this attack occurs is roughly 20 % assuming that all actions are independent.

Usually, the assumption that all actions occur independently, does not hold for real scenarios. In our example, for instance, the probability that the defender actually runs the AV if it is installed, is high. The probability is zero, if the AV is not installed. Unfortunately the simplistic bottom-up procedure used to compute attributes on ADTerms is not suited to handle dependencies between the involved basic actions. It is used to determine the values of the remaining nodes as a function of the values of their children. In the procedure, the value of a refined or countered node only depends on the *values* of its children and *not on their meaning*. Therefore, the bottom-up procedure cannot take dependencies between actions into account. If we, nevertheless, want to be able to deal with dependent actions, we need to choose a different strategy. We opted to combine the ADTree methodology with Bayesian networks, which are a standard formalism used when reasoning about dependent actions. In the following sections we first drop the assumption that the occurrence of actions is independent. Then we drop the assumption that every action that occurs is always successful. Before, we describe our framework in the next section, we need to introduce some terminology.

Given an ADTerm t, we denote by $\mathrm{var}(t)$ the set of propositional variables corresponding to the basic actions involved in t. A configuration $\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}$ (Definition 3.6) represents which actions are executed (the corresponding variables are set to 1) and which are not (the corresponding variables are set to 0). Following our terminological convention from Remark 6.18, we say that $\mathbf{x}$ is an *attack with respect to* t if $f_t(\mathbf{x}) = 1$. This allows us to express the partitions of the propositional semantics in the following way: The ADTerms t and t' belong to the same equivalence class if for all $\mathbf{x} \in \{0,1\}^R$ it holds that the configuration $\mathbf{x}^{\downarrow\mathrm{var}(t)}$ is an attack with respect to t if and only if the configuration $\mathbf{x}^{\downarrow\mathrm{var}(t')}$ is an attack with respect to t'. When talking about the set of all possible attacks with respect to an ADTerm t, we speak of an *attack suite for the system.*

6.2.2 Bayesian Networks for ADTerms

A *Bayesian network* [Pea88] is a graphical representation of a *joint probability distribution* of a finite set of variables with finite domains. The network itself is a directed, acyclic graph that reflects the conditional interdependencies between variables associated with nodes of the network. A directed edge from the node associated with the variable x_1 to the node associated with the variable x_2 means that x_2 conditionally depends on x_1. Each node contains a *conditional probability table* that quantifies the influence between the variables. The joint probability distribution p of a Bayesian network over $\{x_1, \ldots, x_n\}$ is given by

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid \mathrm{par}(x_i)),$$

where $\mathrm{par}(x_i)$ denotes the parents of x_i in the network, defined as all nodes that have an outgoing edge that points into x_i. If the set $\mathrm{par}(x_i)$ is empty, the conditional probability becomes an ordinary probability distribution and the operation that computes the joint probability distribution turns into a simple multiplication. This holds, in particular, for nodes (or subgraphs) of the Bayesian network that are unconnected since they represent stochastically (sets of) variables.

Example 6.20 A Bayesian network describing causal dependencies that may occur when an attacker tries to infect a computer with a virus is illustrated in Figure 6.2. We are assuming the following causal dependencies. If an attacker has managed to distribute a fake version of an AV, he is also more likely to send a virus in an email attachment or distribute the virus on USB sticks than if he has not distributed a fake AV. Furthermore, we assume that a user who has received an infected email attachment or a USB stick is more likely to execute the virus. However, if he has not received the virus via these two channels, he might still obtain the virus

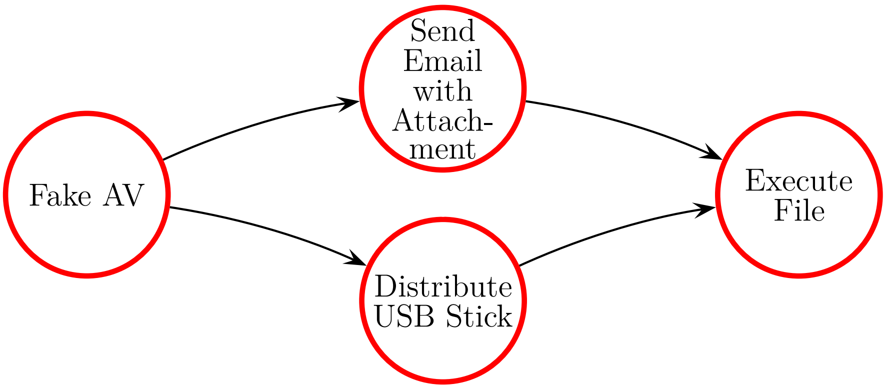


Figure 6.2: The Bayesian network BN_t associated with the ADTerm t from Equation (6.1).

by different means. The conditional probability tables expressing the probability whether certain actions occur ($x_b = 1$) or not ($x_b = 0$), are given in the following.

$$\boxed{\begin{array}{ll} p(x_{\mathrm{EF}} = 1 \mid x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 1) & = 0.9 \\ p(x_{\mathrm{EF}} = 1 \mid x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 0) & = 0.2 \\ p(x_{\mathrm{EF}} = 1 \mid x_{\mathrm{EA}} = 0, x_{\mathrm{DU}} = 1) & = 0.8 \\ p(x_{\mathrm{EF}} = 1 \mid x_{\mathrm{EA}} = 0, x_{\mathrm{DU}} = 0) & = 0.1 \end{array}}$$

$$p(x_{\mathrm{EF}} = 1) = 0.393$$

$$\boxed{p(x_{\mathrm{FA}} = 1) = 0.3}$$

$$\boxed{\begin{array}{ll} p(x_{\mathrm{DU}} = 1 \mid x_{\mathrm{FA}} = 1) & = 0.7 \\ p(x_{\mathrm{DU}} = 1 \mid x_{\mathrm{FA}} = 0) & = 0.2 \end{array}}$$

$$p(x_{\mathrm{DU}} = 1) = 0.35$$

$$\boxed{\begin{array}{ll} p(x_{\mathrm{EA}} = 1 \mid x_{\mathrm{FA}} = 1) & = 0.9 \\ p(x_{\mathrm{EA}} = 1 \mid x_{\mathrm{FA}} = 0) & = 0.3 \end{array}}$$

$$p(x_{\mathrm{EA}} = 1) = 0.48$$

The corresponding joint probability distribution is given by

$$p(x_{\mathrm{EF}}, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{FA}}) = p(x_{\mathrm{EF}} \mid x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot p(x_{\mathrm{EA}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{DU}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{FA}}). \quad (6.3)$$

In the next section, we develop a framework for computing the **probability of occurrence** of attacking a system, which makes use of Bayesian networks. In order to use this framework, we first show how to construct a Bayesian network associated with an ADTerm.

Let t be an ADTerm. A Bayesian network associated with t, denoted by BN_t, is a Bayesian network, i.e., a directed acyclic graph, over the set of propositional variables $\mathrm{var}(t)$, such that there exists a directed edge from $x_{b_1} \in \mathrm{var}(t)$ to $x_{b_2} \in \mathrm{var}(t)$ if and only if the action b_2 depends on the action b_1. Recall that in the ADTree methodology, refined nodes do not contain any additional information, other than how the information presented by their children is composed (conjunctively or disjunctively). This means that the refined nodes *do not* represent any additional actions. This is why, when constructing a Bayesian network for an ADTerm, we take only basic actions into account. The construction is illustrated in the next example.

Example 6.21 The ADTree corresponding to the ADTerm t from Equation (6.1) and the associated Bayesian network BN_t are illustrated in Figure 6.3. The conditional probability tables expressing the probability whether the actions occur ($x_b = 1$) or not ($x_b = 0$), for the attacker are given in Example 6.20. For the basic actions of the defender they are given in the following.

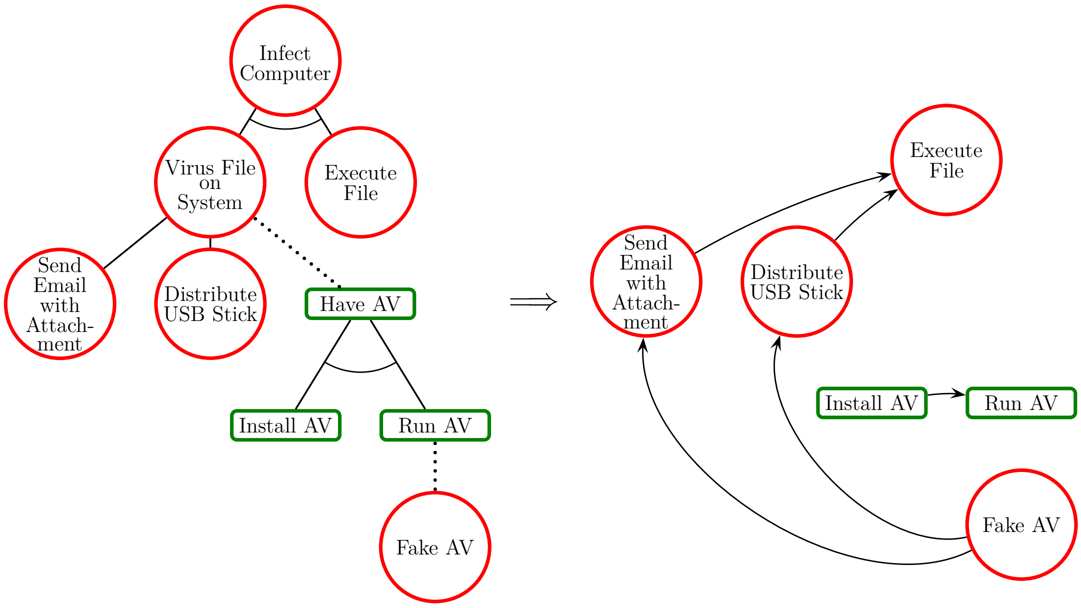


Figure 6.3: Left: An ADTree for infecting a computer corresponding to the AD-Term t from Equation (6.2). Right: A Bayesian network BN_t associated with t.

$$\boxed{\begin{array}{l} p(x_{\mathrm{RA}} = 1 \mid x_{\mathrm{IA}} = 1) \;\; = 0.9 \\ p(x_{\mathrm{RA}} = 1 \mid x_{\mathrm{IA}} = 0) \;\; = 0.0 \end{array}} \qquad \boxed{p(x_{\mathrm{IA}} = 1) = 0.6}$$

$$p(x_{\mathrm{RA}} = 1) = 0.54$$

The joint probability distributions for the defender is given by

$$p(x_{\mathrm{RA}}, x_{\mathrm{IA}}) = p(x_{\mathrm{RA}} \mid x_{\mathrm{IA}}) \cdot p(x_{\mathrm{IA}}). \tag{6.4}$$

The ADTree methodology assumes that the defender does not know which actions the attacker will actually execute and the attacker does not know which countermeasures the defender will put in place. Thus, execution of actions by the two players is considered to be independent. Hence, also the variables associated with basic actions of the attacker and those associated with basic actions of the defender are stochastically independent. This implies that the joint probability distribution for BN_t is obtained by multiplying Equations (6.3) and (6.4) to yield:

$$\begin{aligned} p(x_{\mathrm{EF}}, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{FA}}, x_{\mathrm{RA}}, x_{\mathrm{IA}}) = {} & p(x_{\mathrm{EF}} \mid x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot \\ & p(x_{\mathrm{EA}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{DU}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{FA}}) \cdot p(x_{\mathrm{RA}} \mid x_{\mathrm{IA}}) \cdot p(x_{\mathrm{IA}}). \end{aligned} \tag{6.5}$$

Let us assume that we are interested in the scenario that the attacker will attempt to install the virus file on the system by sending an email with attachment ($x_{\mathrm{EA}} = 1$ and $x_{\mathrm{DU}} = 0$), will execute the virus ($x_{\mathrm{EF}} = 1$) and install a fake AV program ($x_{\mathrm{FA}} = 1$) while the defender will install and run a real AV ($x_{\mathrm{IA}} = 1$ and $x_{\mathrm{RA}} = 1$). By instantiating Equation (6.5) appropriately, we obtain the **probability of occurrence** of *one* specific situation (attack or defense) with respect to t as:

$$\begin{aligned}
&p(x_{\mathrm{EF}} = 1, x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 0, x_{\mathrm{FA}} = 1, x_{\mathrm{RA}} = 1, x_{\mathrm{IA}} = 1)\\
&= p(x_{\mathrm{EF}} = 1 \mid x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 0) \cdot p(x_{\mathrm{EA}} = 1 \mid x_{\mathrm{FA}} = 1) \cdot p(x_{\mathrm{DU}} = 0 \mid x_{\mathrm{FA}} = 1)\\
&\qquad \cdot p(x_{\mathrm{FA}} = 1) \cdot p(x_{\mathrm{RA}} = 1 \mid x_{\mathrm{IA}} = 1) \cdot p(x_{\mathrm{IA}} = 1)\\
&= 0.2 \cdot 0.9 \cdot (1 - 0.7) \cdot 0.3 \cdot 0.9 \cdot 0.6 = 0.008748.
\end{aligned}$$

Naturally, the computed probability is not comparable to the probability computed in Example 6.19. This is caused by the fact that we are only examining one given situation which might not even lead to an attack and not all possible attacks, i.e., the attack suite.

From the example we can nevertheless draw the following conclusion.

Remark 6.22 Every Bayesian network associated with an ADTerm has at least two disjoint connected components.

6.2.3 Computing Probabilities of Attacks Using Bayesian Network

In this section, we present our framework for probability computations on an ADTerm t, taking the dependencies between the involved actions into account. Our computation makes use of the Boolean function f_t and the Bayesian network BN_t.

Given a configuration $\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}$, we define:

$$\psi_t(\mathbf{x}) = f_t(\mathbf{x}) \cdot p(\mathbf{x}), \tag{6.6}$$

where p is the joint probability distribution of BN_t. If the configuration $\mathbf{x}$ is an attack with respect to t, then $f_t(\mathbf{x}) = 1$ and $\psi_t(\mathbf{x})$ returns the probability value for $\mathbf{x}$ from the Bayesian network, representing the probability that the attack $\mathbf{x}$ is attempted. Contrarily, if $\mathbf{x}$ is not an attack with respect to t, $f_t(\mathbf{x}) = 0$ and thus $\psi_t(\mathbf{x}) = 0$. Note that ψ_t can, therefore, be seen as an unnormalized probability distribution. For an attack $\mathbf{x}$, Equation (6.6) still yields the same value as in Example 6.21. If $\mathbf{x}$, however, is not an attack, Equation (6.6) always yields 0, while $p(\mathbf{x})$ does not.

Sometimes, we might not be interested in calculating the **probability of occurrence** of a specific attack, but more generally in the probability of the set of all attacks. This represents the probability that the system is being attacked according to the ADTerm t, similarly to the subject matter of Example 6.19. This corresponds to the sum of the probabilities of all possible attacks with respect to t. Since the value for configurations that do not yield an attack is 0, this is the same as the sum over all configurations. We thus have

$$\mathrm{P}(t) = \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} \psi_t(\mathbf{x}) \overset{(6.6)}{=} \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} f_t(\mathbf{x}) \cdot p(\mathbf{x}). \tag{6.7}$$

We refer to the value $\mathrm{P}(t)$ as the **probability of occurrence** of the *attack suite related to the ADTerm* t, i.e., the occurrence of *any* attack. It is this value that we compare with the **probability of occurrence** computed for ADTrees in Section 6.2.1. Finally, to get the probability of the most probable attack with respect to the term t, we must compute

$$\mathrm{P}_{\max}(t) = \max_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \psi_t(\mathbf{x}) \overset{(6.6)}{=} \max_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} f_t(\mathbf{x}) \cdot p(\mathbf{x}). \tag{6.8}$$

The previous formula computes a value but it is not returning the corresponding configuration. In Section 6.2.6, we discuss how to make use of the fusion algorithm to obtain both the probability value and the corresponding configuration.

Example 6.23 We continue Examples 6.20 and 6.21, depicted in Figure 6.3, using the previously used conditional probability tables. To compute the **probability of occurrence** of the attack suite, we need to deduce all possible situations that yield an attack. For the scenario there exist 21 situations that lead to attacks. They are listed in Table 6.1. In Example 6.21, we have illustrated how to compute the probability that corresponds to Line 8 of Table 6.1.

	x_{EA}	x_{DU}	x_{EF}	x_{IA}	x_{RA}	x_{FA}	$p(\mathbf{x})$
1	1	1	1	1	1	1	0.091854
2	1	1	1	1	0	1	0.010206
3	1	1	1	1	0	0	0.002268
4	1	1	1	0	1	1	0
5	1	1	1	0	1	0	0
6	1	1	1	0	0	1	0.06804
7	1	1	1	0	0	0	0.01512
8	1	0	1	1	1	1	0.008748
9	1	0	1	1	0	1	0.000972
10	1	0	1	1	0	0	0.002016
11	1	0	1	0	1	1	0
12	1	0	1	0	1	0	0
13	1	0	1	0	0	1	0.00648
14	1	0	1	0	0	0	0.01344
15	0	1	1	1	1	1	0.009072
16	0	1	1	1	0	1	0.001008
17	0	1	1	1	0	0	0.004704
18	0	1	1	0	1	1	0
19	0	1	1	0	1	0	0
20	0	1	1	0	0	1	0.00672
21	0	1	1	0	0	0	0.03136

Table 6.1: All 21 configurations that lead to an attack for the scenario depicted in Figure 6.3 and their corresponding **probabilities of occurrence**.

The **probability of occurrence** of the attack suite is then obtained by summing up over the probability values. We obtain $\mathrm{P}(t) = 0.272008$. Since the marginalized probabilities are the same as the probabilities in Example 6.19, it is reasonable to compare the two results. We observe that taking the conditions between the basic actions into account, the probability actually increases from roughly 20 % to 27 %.

The complexity of this computation grows exponentially with the number of involved basic actions. Without prior elimination of the configurations that do not yield attacks, there are $2^{\text{Number of basic actions}} = 2^6 = 64$ possibles configurations to consider. For large systems, it is therefore not feasible to explicitly enumerate or store all values of the Boolean function and the joint probability distribution associated with an ADTerm. In Section 6.2.4, a factorized representation for Boolean functions is introduced. This allows us to increase the efficiency of the computation of Equations (6.7) and (6.8).

An important aspect of the ADTree methodology is to use an appropriate computational procedure in combination with an appropriate semantics, in order to guarantee compatibility, as we discussed in Section 4.3. We conclude this section by making a link to the compatibility notion and show that the framework proposed in this section is a sound extension of the ADTree methodology.

Definition 6.24 (Compatible computational procedure for ADTerms) We say that a computational procedure for ADTerms is compatible with a semantics for ADTerms, if and only if, computations performed on equivalent ADTerms result in the same numerical value.

The following theorem shows under which condition the computational framework developed in this chapter is compatible with the propositional semantics.

Theorem 6.25 *Let t and t' be two ADTerms and BN_t and $\mathrm{BN}_{t'}$ be the corresponding Bayesian networks with the joint probability distributions p and p', respectively. If t and t' are equivalent in the propositional semantics and the marginalized probability distributions over the set of common variables of p and p' are equal, then $\mathrm{P}(t) = \mathrm{P}(t')$.*

Proof. First, the condition that the marginalized probability distributions over the set of common variables of p and p' are equal translates into the following condition: $\forall \mathbf{z} \in \{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}$ it holds that

$$\sum_{\mathbf{u}\in\mathrm{var}(t)\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))} p(\mathbf{z},\mathbf{u}) = \sum_{\mathbf{u}'\in\mathrm{var}(t')\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))} p'(\mathbf{z},\mathbf{u}'). \tag{6.9}$$

Since $f_t \equiv f_{t'}$, we know that variables from $\mathrm{var}(t) \setminus (\mathrm{var}(t) \cap \mathrm{var}(t'))$ do not influence the value of f_t. Thus, for every $\mathbf{z} \in \{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}$ and every $\mathbf{u} \in \{0,1\}^{\mathrm{var}(t)\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))}$, we have $f_t(\mathbf{z},\mathbf{u}) = f_t(\mathbf{z},\mathbf{0})$, where $\mathbf{0}$ is the configuration assigning 0 to every variable from its domain. A similar property holds for $f_{t'}$ and we have $f_t(\mathbf{z},\mathbf{0}) = f_{t'}(\mathbf{z},\mathbf{0})$, for all $\mathbf{z} \in \{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}$. The domains of the two configurations $\mathbf{0}$ in the preceding equation have been omitted for the sake of readability.

We obtain,

$$\begin{aligned}
\mathrm{P}(t) &\overset{(6.7)}{=} \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \Big(f_t(\mathbf{x}) \cdot p(\mathbf{x}) \Big) \\
&= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}} \; \sum_{\mathbf{u}\in\{0,1\}^{\mathrm{var}(t)\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))}} \Big(f_t(\mathbf{z},\mathbf{u}) \cdot p(\mathbf{z},\mathbf{u}) \Big) \\
&= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}} \Big(f_t(\mathbf{z},\mathbf{0}) \cdot \sum_{\mathbf{u}\in\{0,1\}^{\mathrm{var}(t)\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))}} p(\mathbf{z},\mathbf{u}) \Big) \\
&\overset{f_t\equiv f_{t'},(6.9)}{=} \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}} \Big(f_{t'}(\mathbf{z},\mathbf{0}) \cdot \sum_{\mathbf{u}'\in\{0,1\}^{\mathrm{var}(t')\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))}} p'(\mathbf{z},\mathbf{u}') \Big) \\
&= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cap\mathrm{var}(t')}} \; \sum_{\mathbf{u}'\in\{0,1\}^{\mathrm{var}(t')\setminus(\mathrm{var}(t)\cap\mathrm{var}(t'))}} \Big(f_{t'}(\mathbf{z},\mathbf{u}') \cdot p'(\mathbf{z},\mathbf{u}') \Big) \\
&= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t')}} \Big(f_{t'}(\mathbf{x}) \cdot p'(\mathbf{x}) \Big) \overset{(6.7)}{=} \mathrm{P}(t'),
\end{aligned}$$

where the second equality follows from the fact that $(\mathrm{var}(t) \cap \mathrm{var}(t'))$ and $(\mathrm{var}(t) \setminus (\mathrm{var}(t) \cap \mathrm{var}(t')))$ are disjoint sets, the sixth equality from that fact that $(\mathrm{var}(t) \cap \mathrm{var}(t'))$ and $(\mathrm{var}(t') \setminus (\mathrm{var}(t) \cap \mathrm{var}(t')))$ are disjoint sets. □

The previous theorem implies that the probability related to an ADTerm t is the same for all ADTerms equivalent to t in the propositional semantics. This means that the choice of the representative of the equivalence class is not relevant for probability computations. This is of great value for efficiency considerations, as it allows us to choose the Boolean function we work with. For instance, when evaluating probabilities for the ADTerm $b \wedge^{\mathrm{p}} (c \vee^{\mathrm{p}} b)$, instead of using $f(x_{b_1}, x_{b_2}) = x_{b_1} \cdot \max\{x_{b_2}, x_{b_1}\}$, we can employ $f'(x_{b_1}) = x_{b_1}$ because the ADTerms $b_1 \wedge^{\mathrm{p}} (b_2 \vee^{\mathrm{p}} b_1)$ and b_1 are equivalent in the propositional semantics. Since f' has a smaller domain than f, using f' is more efficient.

6.2.4 ADTerms as Constraint Systems

In this section, we employ an encoding technique from constraint reasoning and transform an ADTerm t into its factorized *indicator function* ϕ_t. Such a function maps to 1 if and only if it represents a valid assignment with respect to the Boolean function f_t. The use of factorized indicator functions allows us to improve the efficiency of computations proposed in the previous section.

In order to obtain a factorization of the global indicator function for the Boolean function interpreting an ADTerm in the propositional semantics, we first show how to construct *local indicators* for ADTerms. Local indicators make use of additional variables, called *inner variables*, to represent composed subterms. They are constructed as follows:

1. If $t = \vee^{s}(t_1, \ldots, t_k)$, then the propositional variables $y, y_1, \ldots, y_k$ are associated with $t, t_1, \ldots, t_k$, respectively, and the local indicator function for t is

defined as:

$$\phi(y, y_1, \ldots, y_k) = \begin{cases} 1 & \text{if} \quad y = \max\{y_1, \ldots, y_k\}; \\ 0 & \text{otherwise.} \end{cases}$$

2. If $t = \wedge^s(t_1, \ldots, t_k)$, then the propositional variables $y, y_1, \ldots, y_k$ are associated with $t, t_1, \ldots, t_k$, respectively, and the local indicator function for t is defined as:

$$\phi(y, y_1, \ldots, y_k) = \begin{cases} 1 & \text{if} \quad y = y_1 \cdot \ldots \cdot y_k (= \min\{y_1, \ldots, y_k\}); \\ 0 & \text{otherwise.} \end{cases}$$

3. If $t = \mathrm{c}^s(t_1, t_2)$, then the propositional variables y, y_1 and y_2 are associated with t, t_1 and t_2 respectively, and the local indicator function for t is defined as:

$$\phi(y, y_1, y_2) = \begin{cases} 1 & \text{if} \quad y = y_1 \cdot (1 - y_2)(= \min\{y_1, 1 - y_2\}); \\ 0 & \text{otherwise.} \end{cases}$$

Example 6.26 A step-wise construction of the local indicator functions for the ADTerm t given in Example 6.16 proceeds as follows:

$$t = \underbrace{\wedge^{\mathrm{p}}(\underbrace{\mathrm{c}^{\mathrm{p}}(\underbrace{\vee^{\mathrm{p}}(\mathrm{EA}, \mathrm{DU})}_{y_1}, \underbrace{\wedge^{\mathrm{o}}(\mathrm{IA}, \underbrace{\mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{FA})}_{y_2})}_{y_3})}_{y_4}, \mathrm{EF})}_{y_t}.$$

We define:

$$\begin{array}{lll}
t_1 = \vee^{\mathrm{p}}(\mathrm{EA}, \mathrm{DU}), & \text{thus } \phi_1(y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}) = 1 & \text{exactly if } y_1 = \max(x_{\mathrm{EA}}, x_{\mathrm{DU}}). \\
t_2 = \mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{FA}), & \text{thus } \phi_2(y_2, x_{\mathrm{RA}}, x_{\mathrm{FA}}) = 1 & \text{exactly if } y_2 = x_{\mathrm{RA}} \cdot (1 - x_{\mathrm{FA}}). \\
t_3 = \wedge^{\mathrm{o}}(\mathrm{IA}, y_2), & \text{thus } \phi_3(y_3, x_{\mathrm{IA}}, y_2) = 1 & \text{exactly if } y_3 = x_{\mathrm{IA}} \cdot y_2. \\
t_4 = \mathrm{c}^{\mathrm{p}}(y_1, y_3), & \text{thus } \phi_4(y_4, y_1, y_3) = 1 & \text{exactly if } y_4 = y_1 \cdot (1 - y_3). \\
t_5 = \wedge^{\mathrm{p}}(y_4, \mathrm{EF}), & \text{thus } \phi_5(y_t, y_4, x_{\mathrm{EF}}) = 1 & \text{exactly if } y_t = y_4 \cdot x_{\mathrm{EF}}.
\end{array}$$

In this case, the inner variables are y_1, y_2, y_3, y_4 and y_t. We illustrate the meaning of an indicator for an ADTerm using the function ϕ_2 corresponding to the subterm $\mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{FA})$. The defense represented by this subterm only works ($y_2 = 1$) if an AV is running ($x_{\mathrm{RA}} = 1$) and there is no fake AV installed on the computer ($x_{\mathrm{FA}} = 0$). The configuration $\mathbf{z} = (y_2 = 1, x_{\mathrm{RA}} = 1, x_{\mathrm{FA}} = 0)$ is, therefore, logically admissible and thus $\phi_2(\mathbf{z}) = 1$. The same holds for the configuration $\mathbf{z}' = (y_2 = 0, x_{\mathrm{RA}} = 1, x_{\mathrm{FA}} = 1)$, which represents that the defense does not work ($y_2 = 0$) when the AV is running ($x_{\mathrm{RA}} = 1$) but there is a fake AV installed ($x_{\mathrm{FA}} = 1$). Thus we also have $\phi_2(\mathbf{z}') = 1$. However, the configuration $\mathbf{z}'' = (y_2 = 1, x_{\mathrm{RA}} = 1, x_{\mathrm{FA}} = 1)$ corresponds to the case when the defense is working ($y_2 = 1$) while the AV is running ($x_{\mathrm{RA}} = 1$) and a fake AV is installed ($x_{\mathrm{FA}} = 1$). Since the configuration $\mathbf{z}''$ is logically inadmissible, we have $\phi_2(\mathbf{z}'') = 0$.

Having constructed all local indicators for t, we can build the *global indicator function* ϕ_t. The domain of ϕ_t contains all variables used by the local indicator functions associated to the subterms of t, i.e., the inner variables and the variables corresponding to the basic actions of t. An assignment over all variables is valid if and only if each local assignment is valid. Hence, we may compute the global indicator function for an ADTerm t by multiplying all its local indicators. We use the following notation: Given the global indicator function ϕ_t for t, we denote by y_t the inner variable corresponding to the entire term t. The set of all inner variables of ϕ_t is denoted by D_t.

Example 6.27 For the term discussed in Example 6.26, we obtain the following global indicator function:

$$\begin{aligned}\phi_t(y_1, y_2, y_3, y_4, y_t, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{RA}}, x_{\mathrm{FA}}, x_{\mathrm{IA}}, x_{\mathrm{EF}}) = \phi_1(y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}})\cdot \\ \phi_2(y_2, x_{\mathrm{RA}}, x_{\mathrm{FA}}) \cdot \phi_3(y_3, x_{\mathrm{IA}}, y_2) \cdot \phi_4(y_4, y_1, y_3) \cdot \phi_5(y_t, y_4, x_{\mathrm{EF}}).\end{aligned} \tag{6.10}$$

Remark 6.28 The local and global indicator functions are Boolean functions. Furthermore, the construction of the indicators implies that, if t and t' are two ADTerms equivalent in the propositional semantics, then $\phi_t \equiv \phi_{t'}$.

In practice, the global indicator function can still not be computed due to the exponential growth of the number of inputs. However, its factorization, in terms of local indicators which are bounded in size, introduces additional structure that can be exploited for local computation [PK11]. In Section 6.2.6, we show that meaningful computational tasks can be performed on the factorization. This implies that the global indicator function does not need to be explicitly constructed.

The inner variables, such as $y_1, \ldots, y_4, y_t$ in Example 6.26, are important for revealing the factorization of the global indicator function, but they actually contain only redundant information.

Theorem 6.29 *Consider an ADTerm t consisting of the basic actions $b_1, \ldots, b_n$ and the global indicator function ϕ_t. Given a specific configuration $\mathbf{x}$ of the variables $x_{b_1}, \ldots, x_{b_n}$ corresponding to basic actions, there is exactly one configuration $\mathbf{y}$ of the inner variables from D_t, such that $\phi_t(\mathbf{y}, \mathbf{x}) = 1$.*

Proof. Consider an ADTerm $t = \xi^s(t_1, \ldots, t_k)$, for some $t_1, \ldots, t_k \in \mathbb{T}_\Sigma$, $\xi \in \{\mathrm{c}, \vee, \wedge\}$, $s \in S$, $k \in \mathbb{N}^+$ and the corresponding indicator function $\phi(y, y_1, \ldots, y_k)$. Let $\mathbf{z}$ be a configuration of the variables $y_1, \ldots, y_k$. There is, by definition, exactly one value $y \in \{0, 1\}$, such that for every local indicator it holds that $\phi(y, \mathbf{z}) = 1$. Since the global indicator function is obtained by multiplication, we may directly conclude the theorem. □

An immediate consequence of the previous theorem is that, for a specific configuration $\mathbf{x} \in \{0, 1\}^{\mathrm{var}(t)}$ of the variables associated with basic actions, we have

$$\max_{\mathbf{y} \in \{0,1\}^{D_t}} \phi_t(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{y} \in \{0,1\}^{D_t}} \phi_t(\mathbf{y}, \mathbf{x}) = 1. \tag{6.11}$$

6.2.5 Indicators for Probability Computation

We use the propositional interpretation for ADTerms in order to evaluate which combinations of basic actions correspond to attacks. The global indicator function ϕ_t maps all valid configurations to 1, including those that do not correspond to attacks. For instance, in the case of the function given by Equation (6.10), we have $\phi_t(y_1 = 1, y_2 = 0, y_3 = 0, y_4 = 1, y_t = 0, x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 1, x_{\mathrm{RA}} = 1, x_{\mathrm{FA}} = 1, x_{\mathrm{IA}} = 1, x_{\mathrm{EF}} = 0) = 1$. The considered configuration, although valid, does not correspond to an infection of a computer because $y_t = 0$. In general, when reasoning in terms of global indicator functions, we are only interested in configurations where the variable y_t equals 1. Therefore, we *condition* ϕ_t on $y_t = 1$, which means that we invalidate all configurations with $y_t = 0$. This is achieved by defining a *filter* F_t for the ADTerm t that satisfies $F_t(y_t) = 1$ if and only if $y_t = 1$. In other words, $F_t\colon \{0,1\}^{\{y_t\}} \to \{0,1\}$ is the indicator function for the variable y_t. Multiplying the filter F_t and the global indicator function ϕ_t results in a function which maps to 1 if and only if the assignment of values for the variables represents an attack with respect to the ADTerm t. For our running example, we obtain:

$$\begin{aligned}
&\phi_{t|_{y_t=1}}(y_1, y_2, y_3, y_4, y_t, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{RA}}, x_{\mathrm{FA}}, x_{\mathrm{IA}}, x_{\mathrm{EF}})\\
&= F_t(y_t) \cdot \phi_t(y_1, y_2, y_3, y_4, y_t, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{RA}}, x_{\mathrm{FA}}, x_{\mathrm{IA}}, x_{\mathrm{EF}})\\
&= F_t(y_t) \cdot \phi_1(y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot \phi_2(y_2, x_{\mathrm{RA}}, x_{\mathrm{FA}}) \cdot \phi_3(y_3, x_{\mathrm{IA}}, y_2)\cdot\\
&\quad \phi_4(y_4, y_1, y_3) \cdot \phi_5(y_t, y_4, x_{\mathrm{EF}}).
\end{aligned} \tag{6.12}$$

Remark 6.30 By construction, we know that, if t and t' are two ADTerms that are equivalent in the propositional semantics, then F_t and $F_{t'}$ are equivalent Boolean functions. Thus, using Remark 6.28, we have $\phi_{t|_{y_t=1}} \equiv \phi_{t'|_{y_{t'}=1}}$.

We now make use of conditioning to express the probabilistic values we are interested in. Let t be an ADTerm, ϕ_t be its global indicator function and F_t be the filter for t. Assume that we are given a specific configuration $\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}$. The configuration $\mathbf{x}$ is an attack with respect to t if and only if, there exists $\mathbf{y} \in \{0,1\}^{D_t}$, such that $\phi_{t|_{y_t=1}}(\mathbf{y}, \mathbf{x})$ maps to 1. From Equation (6.11) it follows that

$$\begin{aligned}
\max_{\mathbf{y}\in\{0,1\}^{D_t}} \Big(\phi_{t|_{y_t=1}}(\mathbf{y}, \mathbf{x})\Big) &= \sum_{\mathbf{y}\in\{0,1\}^{D_t}} \phi_{t|_{y_t=1}}(\mathbf{y}, \mathbf{x})\\
&= f_t(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \text{ is an attack;}\\ 0 & \text{if } \mathbf{x} \text{ is not an attack.}\end{cases}
\end{aligned} \tag{6.13}$$

Making use of the previous equation, the probability computation procedure developed in Section 6.2.3 can be redefined as follows.

Let t be an ADTerm and $\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}$ be a configuration of the variables corresponding to the basic actions involved in t. If $\mathbf{x}$ is an attack with respect to t, then its **probability of occurrence** is computed as:

$$\begin{aligned}
\psi_t(\mathbf{x}) \overset{(6.6)}{=} f_t(\mathbf{x}) \cdot p(\mathbf{x}) &\overset{(6.13),\text{ distrib.}}{=} \sum_{\mathbf{y}\in\{0,1\}^{D_t}} \Big(\phi_{t|_{y_t=1}}(\mathbf{y}, \mathbf{x}) \cdot p(\mathbf{x})\Big)\\
&\overset{(6.13),\text{ distrib.}}{=} \max_{\mathbf{y}\in\{0,1\}^{D_t}} \Big(\phi_{t|_{y_t=1}}(\mathbf{y}, \mathbf{x}) \cdot p(\mathbf{x})\Big).
\end{aligned} \tag{6.14}$$

The **probability of occurrence** of the attack suite related to the ADTerm t is expressed as

$$\begin{aligned}\mathrm{P}(t) &\overset{(6.7)}{=} \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \psi_t(\mathbf{x}) \overset{(6.14)}{=} \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \sum_{\mathbf{y}\in\{0,1\}^{D_t}} \Big(\phi_{t|y_t=1}(\mathbf{y},\mathbf{x})\cdot p(\mathbf{x})\Big)\\ &= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cup D_t}} \Big(\phi_{t|y_t=1}(\mathbf{z})\cdot p(\mathbf{z}^{\downarrow\mathrm{var}(t)})\Big). \qquad (6.15)\end{aligned}$$

Finally, it follows from Equation (6.14) that the **probability of occurrence** of the most probable attack with respect to t is

$$\mathrm{P}_{\max}(t) \overset{(6.8)}{=} \max_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \psi_t(\mathbf{x}) \overset{(6.14)}{=} \max_{\mathbf{z}\in\{0,1\}^{\mathrm{var}(t)\cup D_t}} \Big(\phi_{t|y_t=1}(\mathbf{z})\cdot p(\mathbf{z}^{\downarrow\mathrm{var}(t)})\Big). \qquad (6.16)$$

Example 6.31 Let U be the set $\{y_1, y_2, y_3, y_4, y_t, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{RA}}, x_{\mathrm{FA}}, x_{\mathrm{IA}}, x_{\mathrm{EF}}\}$ for our running example. Then the **probability of occurrence** of the attack suite related to the ADTerm t from Equation (6.1) gives

$$\mathrm{P}(t) \overset{(6.15),(6.12),(6.5)}{=} \sum_{\mathbf{z}\in\{0,1\}^{U}} \Big(F_t(\mathbf{z}^{\downarrow\{y_t\}})\cdot\phi_1(\mathbf{z}^{\downarrow\{y_1,x_{\mathrm{EA}},x_{\mathrm{DU}}\}})\cdot\ldots\cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}})\Big).$$

Moreover, the **probability of occurrence** $\mathrm{P}_{\max}(t)$ of the most probable attack gives

$$\max_{\mathbf{x}\in\{0,1\}^{\mathrm{var}(t)}} \psi_t(\mathbf{x}) = \max_{U}\Big(F_t(\mathbf{z}^{\downarrow\{y_t\}})\cdot\phi_1(\mathbf{z}^{\downarrow\{y_1,x_{\mathrm{EA}},x_{\mathrm{DU}}\}})\cdot\ldots\cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}})\Big).$$

While performing the previous computation is possible, it is unfortunately not efficient. For instance, every $\mathbf{z}$ has eleven components, i.e., the following shape:

$$\begin{aligned}\mathbf{z} = (y_1 = 1, y_2 = 0, y_3 = 0, y_4 = 1, y_t = 0, x_{\mathrm{EA}} = 1, x_{\mathrm{DU}} = 1,&\\ x_{\mathrm{RA}} = 1, x_{\mathrm{FA}} = 1, x_{\mathrm{IA}} = 1, x_{\mathrm{EF}} = 0).&\end{aligned}$$

Moreover, the sum that needs to be evaluated to obtain a result for $\mathrm{P}(t)$ runs over 2^{11} different possible $\mathbf{z}$. In the following two sections, we explain how to reduce the complexity of this computation.

6.2.6 Semiring Valuation Algebras

As we already pointed out, in most cases it is impossible to construct the global indicator function and the joint probability distribution efficiently. Hence, we require a method of computation that limits the size of intermediate results and, therefore, diminishes the exponential growth. In this section, we introduce an algorithm, called *fusion*, that allows us to improve the efficiency of computing Equations (6.15) and (6.16).

In order to do so, we first introduce semiring valuations. Let $D \subseteq R$ be a finite set of propositional variables and $\langle A, \oplus, \odot\rangle$ be a commutative semiring (Section 4.3.2). A *semiring valuation* over $\langle A, \oplus, \odot\rangle$ is a function $\phi\colon \{0,1\}^D \to A$ associating a value from A with each configuration from $\{0,1\}^D$. We denote by $\mathrm{dom}(\phi) = D$ the domain of valuation ϕ. Consider two valuations ϕ and ψ over a semiring $\langle A, \oplus, \odot \rangle$, with domains $\mathrm{dom}(\phi) = D$ and $\mathrm{dom}(\psi) = U$. The *combination of ϕ and ψ*, denoted by $\phi \otimes \psi$, is defined, for all $\mathbf{x} \in \{0,1\}^{D \cup U}$, as:

$$(\phi \otimes \psi)(\mathbf{x}) = \phi(\mathbf{x}^{\downarrow U}) \odot \psi(\mathbf{x}^{\downarrow W}).$$

The *elimination* of the variable $x \in D$ is defined for all $\mathbf{x} \in \{0,1\}^{D \setminus \{x\}}$ as:

$$\phi^{-x}(\mathbf{x}) = \phi(\mathbf{x}, 0) \oplus \phi(\mathbf{x}, 1).$$

Note that, due to associativity of semiring addition $\oplus$, we can eliminate variables in any order. For $\{x_1, \ldots, x_m\} \subseteq \mathrm{dom}(\phi)$ we may, therefore, write

$$\phi^{-\{x_1,\ldots,x_m\}} = \Big(\cdots \Big((\phi^{-x_1})^{-x_2} \Big) \cdots \Big)^{-x_m}.$$

The indicator functions, used in Section 6.2.4 for modeling ADTerms in the propositional semantics, are Boolean semiring valuations over $\langle \{0,1\}, \max, \cdot \rangle$. Arithmetic semiring valuations over $\langle \mathbb{R}, +, \cdot \rangle$ capture the conditional probability tables from Bayesian networks in Section 6.2.2, and product t-norm semiring valuations over $\langle [0,1], \max, \cdot \rangle$ compute maximum attack probabilities, as in Equation (6.16).

In [PK11] it was shown that semiring valuations over arbitrary commutative semirings always satisfy the axioms of a valuation algebra. The computational interest in valuation algebras arises from the *inference problem*. Given a set of (semiring) valuations $\{\phi_1, \ldots, \phi_n\}$, called *knowledgebase*, with domains $D_i = \mathrm{dom}(\phi_i)$, for $i = 1, \ldots, n$ and a set of variables $\{x_1, \ldots, x_m\} \subseteq D_1 \cup \cdots \cup D_n$, the inference problem consists of computing

$$\phi^{-\{x_1,\ldots,x_m\}} = (\phi_1 \otimes \cdots \otimes \phi_n)^{-\{x_1,\ldots,x_m\}}. \tag{6.17}$$

For instance, computing the probability in Example 6.31 amounts to solving the inference problem

$$\Big(F_t(\mathbf{z}^{\downarrow\{y_t\}}) \cdot \phi_1(\mathbf{z}^{\downarrow\{y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}\}}) \cdot \ldots \cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}}) \Big)^{-(\mathrm{var}(t) \cup D_t)}$$
$$= \sum_{\mathbf{z} \in \{0,1\}^{\mathrm{var}(t) \cup D_t}} \Big(F_t(\mathbf{z}^{\downarrow\{y_t\}}) \cdot \phi_1(\mathbf{z}^{\downarrow\{y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}\}}) \cdot \ldots \cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}}) \Big).$$

Here, the knowledgebase consists of all local indicator functions, filter F_t and all conditional probability tables, which instantiate arithmetic semiring valuations. Likewise, computing most probable attacks in Equation (6.16) amounts to solving the inference problem over the product t-norm semiring:

$$\Big(F_t(\mathbf{z}^{\downarrow\{y_t\}}) \cdot \phi_1(\mathbf{z}^{\downarrow\{y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}\}}) \cdot \ldots \cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}}) \Big)^{-(\mathrm{var}(t) \cup D_t)}$$
$$= \max_{\mathbf{z} \in \{0,1\}^{\mathrm{var}(t) \cup D_t}} \Big(F_t(\mathbf{z}^{\downarrow\{y_t\}}) \cdot \phi_1(\mathbf{z}^{\downarrow\{y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}\}}) \cdot \ldots \cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}}) \Big).$$

In most cases a direct computation of Equations (6.15), and (6.16) and, more generally, of Equation (6.17) is not possible, due to the exponential complexity when combining semiring valuations. However, because the computational tasks are stated with respect to a factorization of the global indicator function and the joint probability distribution, we may exploit the additional structure inside the factorization and perform calculations locally on the domain of the factors. This general technique is called *local computation*. Correctness of local computation algorithms for arbitrary valuation algebras has been proven in [PK11]. One such algorithm is the *fusion* [She92] or *bucket-elimination* [Dec99] algorithm presented in the following section. Fusion is typically applied to evaluate Bayesian networks. However, it can also be applied to factorizations of arbitrary (semiring) valuation algebras. Thus, we may use fusion for processing inference problems obtained from ADTerms.

6.2.7 The Fusion Algorithm

Let us first consider the elimination of a single variable $x \in \mathrm{dom}(\phi) = \mathrm{dom}(\phi_1) \cup \cdots \cup \mathrm{dom}(\phi_n)$ from a set $\{\phi_1, \ldots, \phi_n\}$ of valuations. This operation can be performed as follows:

$$\mathrm{Fus}_x(\{\phi_1, \ldots, \phi_n\}) = \{\psi^{-x}\} \cup \{\phi_i \colon x \notin \mathrm{dom}(\phi_i)\}, \tag{6.18}$$

where $\psi = \bigotimes_{\{i \colon x \in \mathrm{dom}(\phi_i)\}} \phi_i$. The fusion algorithm for removing several variables then is obtained by repeated application of this operation:

$$(\phi_1 \otimes \cdots \otimes \phi_n)^{-\{x_1,\ldots,x_m\}} = \bigotimes \mathrm{Fus}_{x_m}(\cdots(\mathrm{Fus}_{x_1}(\{\phi_1, \ldots, \phi_n\}))\cdots),$$

where $\bigotimes$ extends the binary combination operator $\otimes$ to a finite set of valuations. For proofs see [Koh03]. In every step $i = 1, \ldots, m$ of the fusion algorithm, the combination in Equation (6.18) creates an intermediate factor ψ_i with domain $\mathrm{dom}(\psi_i)$. Then, the variable x_i is eliminated only from ψ_i in Equation (6.18). We define $\lambda(i) = \mathrm{dom}(\psi_i) \setminus \{x_i\}$ and call this the *fusion label*. Observe that $\lambda(m) = (\mathrm{dom}(\phi_1) \cup \cdots \cup \mathrm{dom}(\phi_n)) \setminus \{x_1, \ldots, x_m\}$. It follows that the size of the domains of all intermediate results of the fusion algorithm are at most one larger than the largest fusion label. The smaller the fusion labels are, the more efficient fusion is. We remark that the labels depend on the chosen elimination sequence for the variables from x_1 to x_m

Finding which sequences lead to the smallest fusion labels is NP-complete [ACP87], however, there are good heuristics that achieve reasonable execution times [Dec03]. In summary, the worst case complexity of computing Equation (6.17) is still exponential, however, we improved the bound in terms of the number of variables involved in the problem to a bound in terms of the size of the largest label that occurs during fusion.

Fusion applies to computational tasks that take the form of Equation (6.17), i.e., that amount to computing projections of a factorized global semiring valuation. Due to the encoding with indicator functions, both the **probability of occurrence** of the attack suite related to an ADTerm, see Equation (6.15), and the **probability of occurrence** of the most probable attack, see Equation (6.16), can be written in this form.

Example 6.32 Let U be the set $\{y_1, y_2, y_3, y_4, y_t, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{RA}}, x_{\mathrm{FA}}, x_{\mathrm{IA}}, x_{\mathrm{EF}}\}$ as in the previous example. We recompute Example 6.23 with the help of the fusion algorithm using the following elimination sequence

$$(x_{\mathrm{IA}}, x_{\mathrm{RA}}, y_2, x_{\mathrm{FA}}, y_3, x_{\mathrm{EA}}, x_{\mathrm{DU}}, x_{\mathrm{EF}}, y_1, y_4, y_t).$$

Then, the **probability of occurrence** of the attack suite related to the ADTerm t given in Equation (6.1) is computed as follows:

$$\begin{aligned}
\mathrm{P}(t) = & \sum_{\mathbf{z}\in\{0,1\}^U} \Big(F_t(\mathbf{z}^{\downarrow\{y_t\}}) \cdot \phi_1(\mathbf{z}^{\downarrow\{y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}\}}) \cdot \ldots \cdot p(\mathbf{z}^{\downarrow\{x_{\mathrm{IA}}\}}) \Big) \\
= & \sum_U F_t(y_t) \cdot \phi_1(y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot \phi_2(y_2, x_{\mathrm{RA}}, x_{\mathrm{FA}}) \cdot \phi_3(y_3, x_{\mathrm{IA}}, y_2) \\
& \quad \cdot \phi_4(y_4, y_1, y_3) \cdot \phi_5(y_t, y_4, x_{\mathrm{EF}}) \cdot p(x_{\mathrm{EF}} \mid x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot p(x_{\mathrm{EA}} \mid x_{\mathrm{FA}}) \\
& \quad \cdot p(x_{\mathrm{DU}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{FA}}) \cdot p(x_{\mathrm{RA}} \mid x_{\mathrm{IA}}) \cdot p(x_{\mathrm{IA}}) \\
= & \sum_{y_t} F_t(y_t) \cdot \sum_{y_4} \sum_{y_1} \sum_{x_{\mathrm{EF}}} \phi_5(y_t, y_4, x_{\mathrm{EF}}) \\
& \quad \cdot \sum_{x_{\mathrm{DU}}} \sum_{x_{\mathrm{EA}}} \phi_1(y_1, x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot p(x_{\mathrm{EF}} \mid x_{\mathrm{EA}}, x_{\mathrm{DU}}) \cdot \sum_{y_3} \phi_4(y_4, y_1, y_3) \\
& \quad \cdot \sum_{x_{\mathrm{FA}}} p(x_{\mathrm{EA}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{DU}} \mid x_{\mathrm{FA}}) \cdot p(x_{\mathrm{FA}}) \cdot \sum_{y_2} \sum_{x_{\mathrm{RA}}} \phi_2(y_2, x_{\mathrm{RA}}, x_{\mathrm{FA}}) \\
& \quad \cdot \sum_{x_{\mathrm{IA}}} \phi_3(y_3, x_{\mathrm{IA}}, y_2) \cdot p(x_{\mathrm{RA}} \mid x_{\mathrm{IA}}) \cdot p(x_{\mathrm{IA}}).
\end{aligned}$$

In this computation, we have abused the notation in the sums by only indicating the exponent in the domains of the configurations. We iteratively compute the sums, starting with the sum over IA. We have illustrated this computation in Table 6.2, where $f(y_3, y_2, x_{\mathrm{RA}}, x_{\mathrm{IA}}) = \phi_3(y_3, x_{\mathrm{IA}}, y_2) \cdot p(x_{\mathrm{RA}} \mid x_{\mathrm{IA}}) \cdot p(x_{\mathrm{IA}})$.

y_3	y_2	x_{RA}	$f(y_3, y_2, x_{\mathrm{RA}}, 0)$	$f(y_3, y_2, x_{\mathrm{RA}}, 1)$	$f(y_3, y_2, x_{\mathrm{RA}}, 0) + f(y_3, y_2, x_{\mathrm{RA}}, 1)$
1	1	1	$0 \cdot 0 \cdot 0.4$	$1 \cdot 0.9 \cdot 0.6$	0.54
1	1	0	$0 \cdot 1 \cdot 0.4$	$1 \cdot 0.1 \cdot 0.6$	0.06
1	0	1	$0 \cdot 0 \cdot 0.4$	$0 \cdot 0.9 \cdot 0.6$	0
1	0	0	$0 \cdot 1 \cdot 0.4$	$0 \cdot 0.1 \cdot 0.6$	0
0	1	1	$1 \cdot 0 \cdot 0.4$	$0 \cdot 0.9 \cdot 0.6$	0
0	1	0	$1 \cdot 1 \cdot 0.4$	$0 \cdot 0.1 \cdot 0.6$	0.4
0	0	1	$1 \cdot 0 \cdot 0.4$	$1 \cdot 0.9 \cdot 0.6$	0.54
0	0	0	$1 \cdot 1 \cdot 0.4$	$1 \cdot 0.1 \cdot 0.6$	0.46

Table 6.2: An exemplary computation step in the fusion algorithm.

The result of this first computation is a conditional probability table that depends on the values of the variables that occur as factors in the sum. The table is reused in the computation of the next sum, i.e., the sum over RA. Continuing the computation, we obtain $\mathrm{P}(t) = 0.272008$, which is, as expected, the same result as in Example 6.23.

There are eleven sums in the computation, so eleven conditional probability tables need to be computed. The largest number of variables occurring in any of the tables is the size of the largest fusion label. Hence, we have to compute a maximum number of $11 \cdot 2^4 = 176$ configurations. (The actual number is lower, since not all tables have four variables.)

In summary, we have a complexity of $O(2^{\text{number of largest label}})$ compared to a complexity of $O(2^{\text{number of basic actions}})$ in Example 6.23. Since in practice, the size of the largest label is smaller than the number of basic actions, this can yield to considerable speed-ups.

Sometimes we are less interested in the concrete value, such as the maximum attack probability, but rather in finding one or more configurations that map to this value, as they describe the most probable attacks. Fusion can be modified to additionally output such *solution configurations* [Pou08] under the condition that the semiring is totally ordered and idempotent. This is the case for the product t-norm semiring so that we may use fusion to compute maximum attack probabilities and the most probable attacks.

The algorithmic technique based on semiring valuations to combine probabilistic information with the ADTree formalism also works in a broader context. From an algebraic perspective, the combination of indicator functions and probabilities is possible because the Boolean semiring for indicator functions is a subalgebra of both semirings used for expressing probabilities, i.e., the arithmetic and product t-norm semiring. Consequently, we may directly apply the same construction to other semiring valuations under the additional condition that the corresponding semiring has the Boolean semiring as subalgebra. The large family of t-norm semirings [PK11] are important candidates used in possibility and fuzzy set theory [Zad78]. In the next section, we briefly hint at one such generalized application.

6.2.8 Probability of Successful Attacks

Equations (6.6), (6.7) and (6.8) compute the probability that a certain attack or a set of attacks occurs. However, security experts may also be interested in the probability that an attack or a set of attacks is successful. In the following we elaborate how to extend the previously mentioned formulas in order to compute the *probability of success* of an attack, which in our terms is the *probability of success* of an attack suite related to an ADTerm.

First, for every basic action b, we introduce an additional propositional variable s_b representing whether the action b is successful or not. Recall that the previously used variable x_b represents whether b is executed or not, so the two variables do not represent the same events. We set

$$\begin{array}{ll}
\mathrm{P}^{\text{suc}}(s_b = 1 \mid x_b = 1) = p_{\text{suc}} & \text{(probability that } b \text{ is successful if executed)},\\
\mathrm{P}^{\text{suc}}(s_b = 0 \mid x_b = 1) = 1 - p_{\text{suc}} & \text{(probability that } b \text{ is unsuccessful if executed)},\\
\mathrm{P}^{\text{suc}}(s_b = 1 \mid x_b = 0) = 0 & \text{(probability that } b \text{ is successful if not executed)},\\
\mathrm{P}^{\text{suc}}(s_b = 0 \mid x_b = 0) = 1 & \text{(probability that } b \text{ is unsuccessful if not executed)}.
\end{array}$$

Note that p^{suc} is a probability distribution from the point of view of the player corresponding to the node's type.

Remark 6.33 To incorporate success rates, we could model the Bayesian network $\mathrm{BN}_t^{\mathrm{suc}}$ over the set of variables $\{x_b \mid b \in \mathbb{B}\} \cup \{s_b \mid b \in \mathbb{B}\}$, where for each b, there is an edge from x_b to s_b. In this case the probability distribution is denoted by p^{suc}. Note that, for $b \neq b'$, the variables s_b and $s_{b'}$ are stochastically independent. In fact, the probability of the execution of b may depend on the probability of the execution of b', but whether b succeeds is independent from whether b' succeeds.

To ease presentation, we set

$$w_b(x_b) = p^{\mathrm{suc}}(s_b = x_b \mid x_b),$$

which we call *success weights* for b.

Given a configuration, we are interested in its success probability. For b such that $\mathbf{x}(x_b) = 1$, we take $w_b(x_b = 1) = p^{\mathrm{suc}}(s_b = 1 \mid x_b = 1) = p_{\mathrm{suc}}$ and for b such that $\mathbf{x}(x_b) = 0$, we consider $w_b(x_b = 0) = p^{\mathrm{suc}}(s_b = 0 \mid x_b = 0) = 0$. We set

$$w_t(\mathbf{x}) = \prod_{x_b \in \mathrm{var}(t)} w_b(\mathbf{x}^{\downarrow\{x_b\}}).$$

Using this notation, we can extend the previously introduced formulas for computation of **probability of occurrence** (i.e., Equations (6.6), (6.7) and (6.8)) into formulas representing the **probability of success**, as follows:

$$\begin{aligned}
\psi_t^{\mathrm{suc}}(\mathbf{x}) &= f_t(\mathbf{x}) \cdot p(\mathbf{x}) \cdot w_t(\mathbf{x}),\\
P^{\mathrm{suc}}(t) &= \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} \psi_t^{\mathrm{suc}}(\mathbf{x}) = \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} f_t(\mathbf{x}) \cdot p(\mathbf{x}) \cdot w_t(\mathbf{x}),\\
P_{\max}^{\mathrm{suc}}(t) &= \max_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} \psi_t^{\mathrm{suc}}(\mathbf{x}) = \max_{\mathbf{x} \in \{0,1\}^{\mathrm{var}(t)}} f_t(\mathbf{x}) \cdot p(\mathbf{x}) \cdot w_t(\mathbf{x}).
\end{aligned}$$

It can be shown that computation of P^{suc} is compatible with the propositional semantics. In other words, by adding success weights $w(x_B)$, for $x_B \in \mathrm{var}(t) \cup \mathrm{var}(t')$ to the assumptions of Theorem 6.25, we have $P^{\mathrm{suc}}(t) = P^{\mathrm{suc}}(t')$.

6.3 ADTrees and Games

In the previous sections we have already seen how to combine the ADTrees with two mathematical concepts, namely Boolean functions and Bayesian networks, to obtain powerful results. In this section we relate the ADTree methodology to another research area. As we have seen, ADTrees are interleaving attacks and defenses, i.e., actions by attackers and defenders. This alternating notion of opposing players is a concept that is commonly found in game theory [Ras06]. A connection to game theory is desirable because on the one hand, visual ADTrees are more intuitive while game theory profits from the well-studied theoretical properties of games. In [10KMMS] we have made this connection between ADTrees and game theory explicit, which we highlight next.

To be able to provide more details, we define specific games called *two-player binary zero-sum extensive form games*, in which a proponent p and an opponent o (two-player) play against each other. In these games, we allow only for the outcomes $(1, 0)$ and $(0, 1)$ (binary, zero-sum), where $(1, 0)$ means that the proponent succeeds in his goal (breaking the system if he is the attacker, keeping the system secure if he is the defender) and $(0,1)$ means that the opponent succeeds. Moreover, the proponent is not necessarily the player who plays first in the game. Finally, we restrict ourselves to extensive form games, i.e., games in tree format. Our presentation of games differs from the usual one because we present games as terms due to technical reasons.

Definition 6.34 (Two-player binary zero-sum extensive form games) Let $S = \{\mathrm{p}, \mathrm{o}\}$ denote the set of players, L denote a leaf of a term, NL denote a non-leaf of a term and $\mathrm{Out} = \{(1,0),(0,1)\}$ the set of possible outcomes. A two-player binary zero-sum extensive form game is a term t, where

$$\begin{array}{rrcccc} t: & \psi^{\mathrm{p}} & \mid & \psi^{\mathrm{o}} & & \\ \psi^{\mathrm{p}}: & \mathrm{NL}^{\mathrm{p}}(\psi^{\mathrm{o}},\dots,\psi^{\mathrm{o}}) & \mid & \mathrm{L}^{\mathrm{p}}(1,0) & \mid & \mathrm{L}^{\mathrm{p}}(0,1) \\ \psi^{\mathrm{o}}: & \mathrm{NL}^{\mathrm{o}}(\psi^{\mathrm{p}},\dots,\psi^{\mathrm{p}}) & \mid & \mathrm{L}^{\mathrm{o}}(1,0) & \mid & \mathrm{L}^{\mathrm{o}}(0,1). \end{array}$$

We denote the set of all two-player binary zero-sum extensive form games by $\mathcal{G}$. For $s \in S$, we define the *first player* of a game ψ^s as the function $\tau\colon \mathcal{G} \to S$ such that $\tau(\psi^s) = s$. It is easy to check that this is well-defined since each term can only be derived from either ψ^{p} or ψ^{o}.

Example 6.35 An example of a two-player binary zero-sum extensive form game is the term $\mathrm{NL}^{\mathrm{p}}(\mathrm{NL}^{\mathrm{o}}(\mathrm{L}^{\mathrm{p}}(0,1),\mathrm{L}^{\mathrm{p}}(1,0)),\mathrm{L}^{\mathrm{o}}(0,1))$. This game is displayed in Figure 6.4 (left). When displaying extensive form games, we use dashed (red) edges for choices made by the proponent and solid (green) edges for those made by the opponent. In this example, the proponent first picks from two options: If he chooses the first option, the opponent is allowed to choose between the two outcomes $(0,1)$ and $(1,0)$. If the proponent chooses the second option, the game ends with outcome $(0,1)$.

In games every player has a *strategy* that determines his behavior whenever he has a choice in the game. If the strategies of both players are known, we can compute the *outcome of the game*.

When showing a connection between game theory and ADTrees, we restrict ourselves to the propositional semantics, see Section 3.1 and the **satisfiability** attribute, as introduced in Example 4.5.

Although the two formalisms have much in common, their expressive equivalence is not immediate. Whereas the mapping from games to ADTree is rather straight forward, the other direction is not. There are two notions in the domain of ADTrees that have no direct correspondence in the world of games: conjunctive nodes and refinements. The first problem is solved by transforming conjunctive nodes for one player to disjunctive nodes for the other player. The second problem is addressed by adding additional moves in which the other player only has a single option. Due to these two problems, the transformations will not necessarily be each other's inverse.

The transformation of the game introduced in Example 6.35 into an ADTree is illustrated in Figure 6.4. Since the formal description of the mapping is rather technical and brings few insights, we refer to [10KMMS] for details.

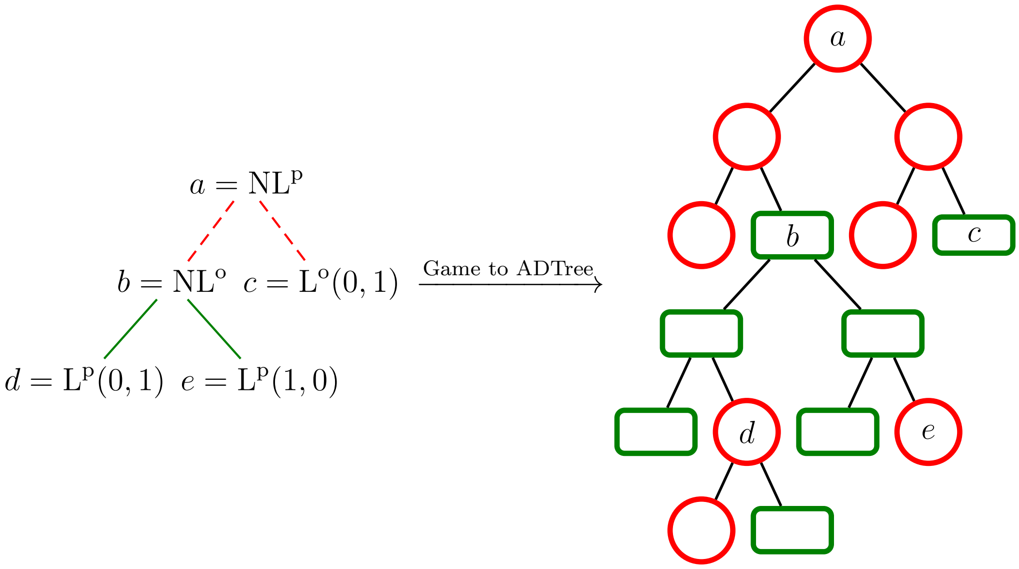

$a = \mathrm{NL}^{\mathrm{p}}$
$b = \mathrm{NL}^{\mathrm{o}}$
$c = \mathrm{L}^{\mathrm{o}}(0,1)$
$d = \mathrm{L}^{\mathrm{p}}(0,1)$
$e = \mathrm{L}^{\mathrm{p}}(1,0)$
Game to ADTree
a
b
c
d
e

Figure 6.4: A two-player binary zero-sum extensive form game is mapped to an ADTree. The letters indicate subgames and mark corresponding subtrees in the ADTree. (Other node labels are omitted.)

When conversely transforming ADTrees into games, the above mentioned problems need to be addressed. The transformation is defined inductively. The leaves for player $s \in S$ are transformed into two options for player s, a losing and a winning one. Disjunctive nodes for player s are transformed into choices for player s in the game. Conjunctive nodes are transformed into choices for the other player. This reflects the fact that a player can succeed in all his options exactly when there is no way for the other player to pick an option which allows him to succeed. Finally, countermeasures against player s are transformed into a choice for player $\overline{s}$. Here, the first option corresponds to player $\overline{s}$ not choosing the countermeasure, so that it is up to player s whether he succeeds or not while the second option corresponds to player $\overline{s}$ choosing the countermeasure. The transformation of the ADTree corresponding to ADTerm $t = \mathrm{c}^{\mathrm{p}}(\wedge^{\mathrm{p}}(E,F), \vee^{\mathrm{o}}(G))$ is illustrated in Figure 6.5.

As mentioned in Chapter 4, the semantics by itself are not enough to quantify a scenario. We have also provided a mapping between strategies in games and corresponding basic assignments in ADTrees as well as a function in the reverse direction that maps a basic assignment to a strategy in the game. We show that for every winning strategy, there exists a basic assignment that yields a satisfiable ADTree. Reciprocally, we show that if a player has a basic assignment for an ADTree in which he is successful, the corresponding strategy in the corresponding game guarantees him to win.

In conclusion, the results of [10KMMS] can be summarized by the following theorem.

Theorem 6.36 *ADTrees in propositional semantics together with the* ***satisfiability*** *attribute and two-player binary zero-sum extensive form games have equivalent expressive power.*

Proof. See [10KMMS]. □

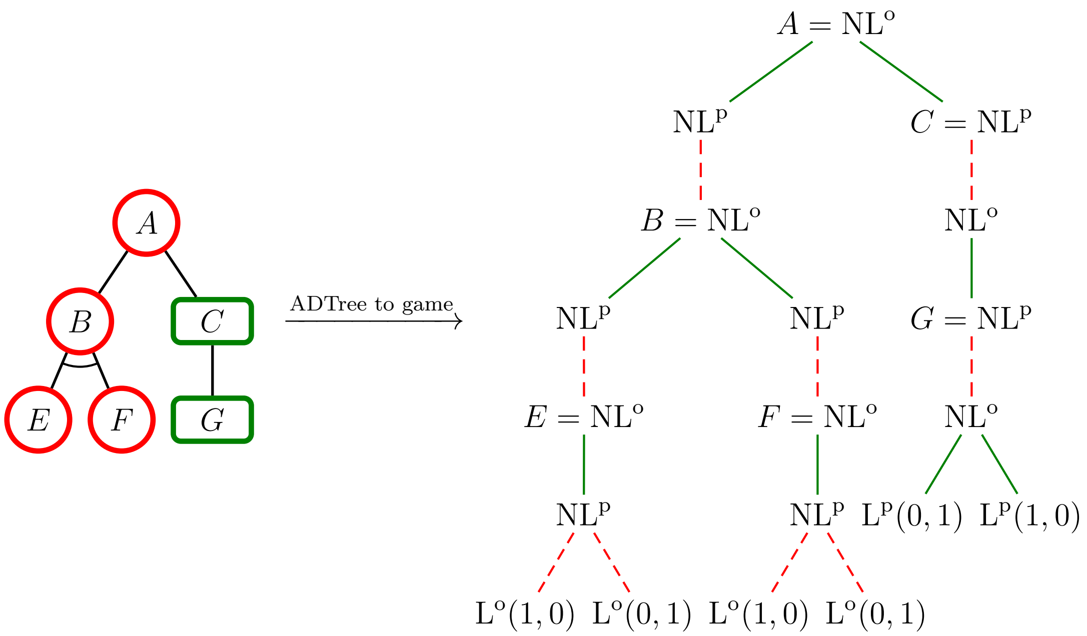


Figure 6.5: An ADTree mapped to a two-player binary zero-sum extensive form game. The letters indicate subtrees and mark corresponding subgames in the game.

Note that the theorem also shows that, when considering the **satisfiability** attribute, the class of conjunction-free ADTrees has the same expressive power as the class of all ADTrees.

7

Related Formalisms

Graphical security models provide a useful method to represent and analyze security scenarios that examine vulnerabilities of systems and organizations. They constitute a valuable support tool to facilitate threat assessment and risk management of real-life systems. The great advantage of graph-based approaches lies in the fact that they are combining user-friendly, intuitive, visual features with formal semantics and algorithms that allow for qualitative and quantitative analysis. They have become popular in the industrial sector and are, for example, used in security analysis of supervisory control and data acquisition (SCADA) systems [BFM04, TLM07, TA10], voting systems [LDEH11, BM07], vehicular communication systems [HAF$^+$09, ABD$^+$06], Internet related attacks [TLFH01, LZRL09], secure software engineering [JEBR10] and socio-technical attacks [12BKMS] and [EPPC11, RVOC08]. Historically, graph-based security formalisms have their origin in safety modeling. In the present day, however, they are regarded as separate research areas. Even though the security and the safety modeling fields are still influencing each other, security modeling approaches are a stand-alone research area. An overview presenting all graph-based approaches, is beyond the scope of this thesis. Therefore, the choice has to be made to concentrate on formalisms based on directed acyclic graphs (DAGs).

In Section 7.1, we introduce DAG-based graphical security models covering attacks and defenses. Then, in Section 7.2, we introduce keywords used in the field of graph-based security modeling and present which aspects we analyzed. In Section 7.3, we provide a template for the description of the formalisms. Section 7.4 is the main part of the related work and presents the DAG-based attack and defense modeling approaches published before 2013. In Section 7.5, we provide a concise tabular overview of the presented formalisms. We illustrate how to use the tables in order to select the most relevant modeling technique, depending on the application requirements. Section 7.6 briefly mentions alternative graphical security models.

7.1 Graphical Security Modeling on DAGs

We focus on DAG-based graphical methods for the analysis of attack and defense scenarios. We understand attack and defense scenarios in a general sense: they encompass any malicious action of an attacker who wants to harm or damage another party or its assets as well as any defense or countermeasure that could be used to prevent or mitigate such malicious actions. Generally, models for attack and defense scenarios are used to analyze cause–consequence situations. For this kind of reasoning, acyclic structures are often sufficient. This statement is supported by the existence of many models that use only acyclic structures. Today, more than 30 different approaches for the analysis of attack and defense scenarios exist. Most of them extend the original methodology in one or several aspects or dimensions which include defensive components, timed and ordered actions, dynamic aspects and different types of quantification. Moreover, methods for computation of security related parameters, such as the cost, the impact or likelihood of an attack, the efficiency of necessary protection measures, or the environmental damage of an attack, have been developed or adapted. Consequently the approaches based on trees, or more generally, on directed acyclic graphs (DAGs) form an important class of methodologies for attack and defense modeling. They can be divided into two subclasses: formalisms derived from or extending threat trees and formalisms based on Bayesian networks.

The model creation in all threat tree-based methodologies starts with the identification of a feared event represented as the root node. The refinement process is illustrated in Figure 7.1, which recreates the first threat tree model proposed by Weiss [Wei91].

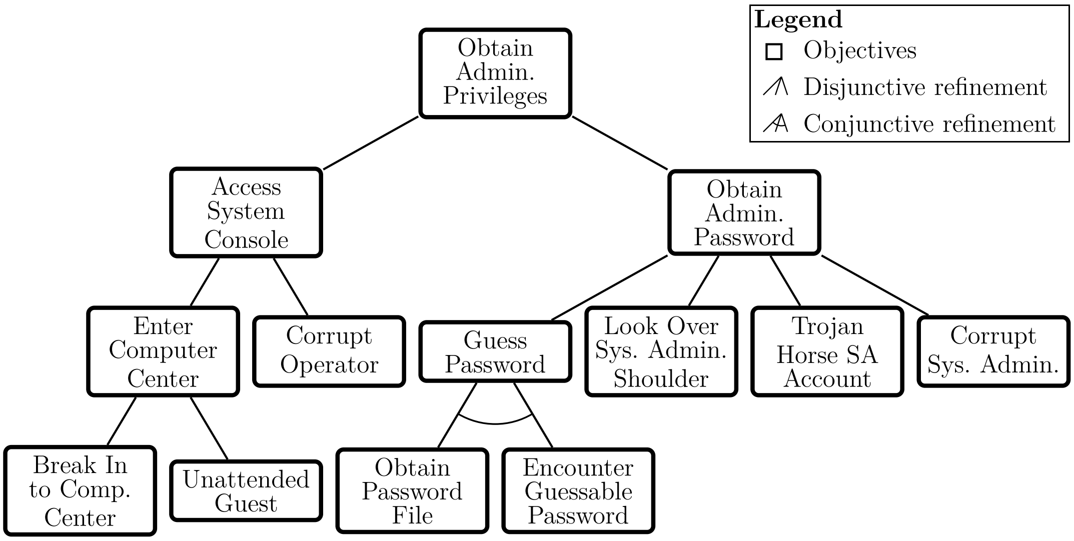
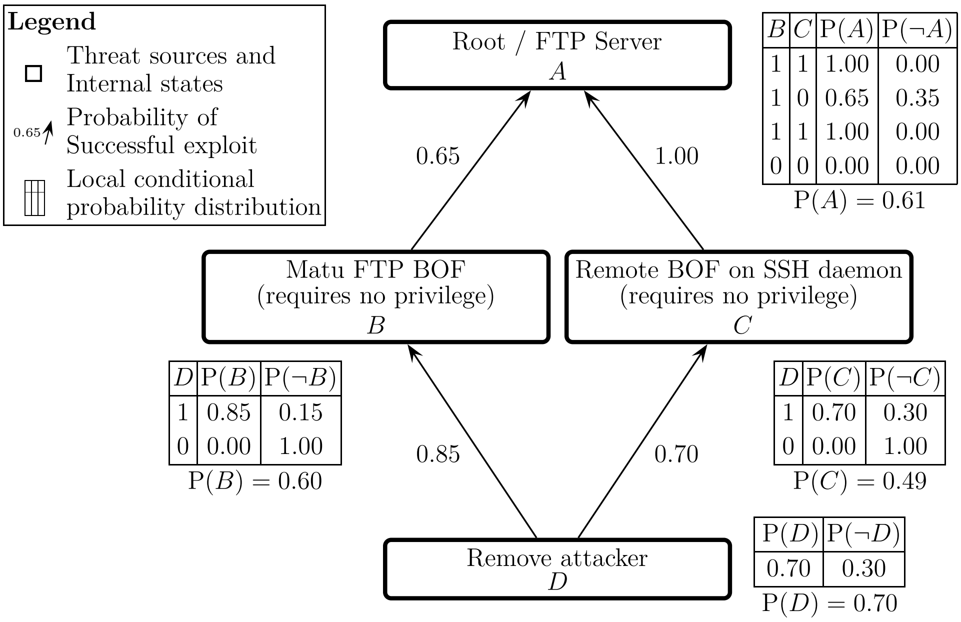


Figure 7.1: A threat logic tree taken from [Wei91]: Obtaining administrator privileges on a UNIX system.

The DAG structure allows us to use refinements with a customizable level of detail. Starting from the root, the nodes of a DAG are refined as long as the refining children provide useful and adequate information about the modeled scenario. Refinements paired with the acyclic structure allow for modularization which in turn allows different experts to work in parallel on the same model. This is highly useful in case of large-scale, complex models, where analysis of different parts requires different types of expertise. A big advantage of the DAG-based approaches is that they are scalable. They do not suffer from the state space explosion problem, which is common for models based on general graphs with cycles. In the case of trees, most of the analysis algorithms are linear with respect to the number of nodes of the model. Due to multiple incoming edges, this property is no longer true for DAGs and the complexity of the analysis methods is exponential in the worst case. However, in practice, the actual running times are still acceptable since the algorithms exploit common structural properties of the underlying cycle-free graphs.

This is, for instance, the case for Bayesian inference algorithms used for the analysis of security models based on Bayesian networks. Figure 7.2 depicts a simple Bayesian attack graph borrowed from [PDR12] and illustrates how to compute the unconditional probability of a vulnerability exploitation, see also Section 6.2.2.



B	C	P(A)	P($\neg A$)
1	1	1.00	0.00
1	0	0.65	0.35
1	1	1.00	0.00
0	0	0.00	0.00

$\mathrm{P}(A) = 0.61$

D	P(B)	P($\neg B$)
1	0.85	0.15
0	0.00	1.00

$\mathrm{P}(B) = 0.60$

D	P(C)	P($\neg C$)
1	0.70	0.30
0	0.00	1.00

$\mathrm{P}(C) = 0.49$

P(D)	P($\neg D$)
0.70	0.30

$\mathrm{P}(D) = 0.70$

Figure 7.2: A Bayesian attack graph taken from [PDR12]: A test network with local conditional probability distributions (tables) and updated unconditional probabilities (below each table).

The contribution of this chapter is to provide an almost complete overview of the field and systematize existing knowledge. More specifically, the survey

- presents the state of the art in the field of DAG-based graphical attack and defense modeling;
- identifies key aspects which allows us to compare different formalisms;
- proposes a taxonomy of the presented approaches, which helps in selecting an appropriate formalism;
- lays a foundation for future research in the field.

7.2 Keywords and Examined Aspects

In this section we introduce our terminology and make a link to existing definitions and concepts. We accept some repetition with respect to the previous chapters of this thesis in order for this section to be self-contained. We then present and define the aspects on the basis of which we have analyzed the different formalism.

When examining different models in the same context, it is imperative to have a common language. Over the last 20 years, numerous concepts and definitions have emerged in the field of graphical security modeling. This section is intended to introduce the language used throughout this chapter and to serve as a quick reference guide over the most commonly occurring concepts. Our goal here is not to point out the differences in definitions or other intricate details.

Attack and defense modeling By techniques for attack and defense modeling we understand formalisms that serve for representation and analysis of malicious behavior of an *attacker* and allow us to reason about possible defending strategies of the attacker's opponent, called the *defender*. In our survey we use attacks in a broad sense. Attacks can also be thought of as *threats*, *obstacles* and *vulnerabilities*. Contrarily, defenses can appear in the form of *protections*, *mitigations*, *responses* and *countermeasures*. They oppose, mitigate or prevent attacks.

Nodes *Nodes*, also called *vertices*, are one of the main components of graph-based security models. They are used to depict the concept that is being modeled. Nodes may represent *events*, *goals*, *objectives* and *actions*. Depending on whether the models are constructed in an inductive or deductive way, nodes may also express *causes* or *consequences*.

Root node In a rooted DAG (and, therefore, in any rooted tree) the *root* is the single designated node that does not have any predecessor. From it, all other nodes can be reached via a directed path. This distinguished node usually depicts the entire concept which is being modeled. In the context of security models, existing names for this special node include *top event*, *main goal*, *main consequence*, *main objective* or *main action*.

Leaf nodes In a DAG, nodes that do not have any children are called *leaves*. They usually display an atomic component of a scenario that is no longer refined. They are also called *primary events*, *basic components*, *elementary attacks*, *elementary components* or *basic actions*.

Edges Edges are the second main component of graph-based security models. They link nodes with each other and, in this way, determine relations between the modeled concepts. Edges are also called *arcs*, *arrows*, or *lines*. In some models, edges may have special semantics and may detail a cause–consequence relation, a specialization or some other information.

Connectors Connectors usually specify more precisely how a parent node is connected with its children. A connector may be a set of edges or a node of a special type. Connectors are also called *refinements* or *gates*. Some examples include: AND, OR, XOR, k-out-of-n, priority AND, triggers, etc.

Priority AND A *priority AND* (PAND) is a special kind of AND connector which prescribes an order in which the nodes are to be treated. The origin of the prescribed order is usually time or some priority criterion. The PAND is also called an *ordered AND*, an *O-AND* or a *sequential AND*. Sometimes the underlying reason behind the priority is specified, as in the case of the *time-based AND*.

Attributes *Attributes* represent aspects or properties that are relevant for quantitative analysis of security models. Examples of attributes, sometimes also called *metrics*, include: impact of an attack, costs of necessary defenses, risk associated with an attack etc. Proposed computation methods range from versatile approaches that can be applied for evaluation of a wide class of attributes, to specific algorithms developed for particular measures. An example of the former is the formalization of an attribute domain proposed in [MO05], which is well-suited for calculation of any attribute whose underlying algebraic structure is a semiring. An example of the latter is the specific method for probability computation proposed in [Yag06].

One of the goals of this chapter is to provide a classification of existing formalisms for attack and defense modeling. For this purpose, all DAG-based approaches, described in detail in Section 7.4, were analyzed based on the same 13 criteria, which we refer to as *aspects* and define in this section.

The formalisms are grouped according to the following two main aspects:

1. **Attack and/or defense modeling:** *Attack* modeling techniques are focused on an attacker's actions and vulnerabilities of systems; *defense* modeling techniques concentrate on defensive aspects, such as detection, reaction, responses and prevention.

2. **Static or sequential approaches:** *Sequential* formalisms take temporal aspects, such as dynamics or time variations, and dependencies between considered actions, such as order or priority, into account; *static* approaches cannot model any such relations.

The above two aspects provide a partition of all considered approaches. Furthermore, they correspond to questions that a user selecting a suitable formalism is most likely to ask, namely *"What do we want to model?"* and *"How do we want to model?"*. The proposed classification allows a reader to easily make a primary selection and identify which formalisms best fit his needs.'

Besides the two main aspects, each formalism is analyzed according to additional criteria, listed in Table 7.1. All aspects taken into account in our work, can be grouped into three categories:

- Aspects relating to the formalism's *modeling capabilities*, i.e., what we can model: attack or defense modeling, sequential or static modeling, quantification, main purpose and extensions.

- Aspects relating to the formalism's *characteristics*, i.e., how we can model: structure, connectors and formalization.

- Aspects relating to the formalism's *maturity and usability*: software tool availability, case study, external use, related research paper count and year of invention.

In Table 7.1, we define all 13 aspects in the form of questions and provide possible values that answer the questions.

7.3 The Template of the Formalism Descriptions

The description of each formalism presented in Section 7.4 complies with the following detailed template.

Aspect	**Aspect Description**	**Possible Values**	**Value Explanation**
Attack or defense	Is the formalism offensively or defensively oriented?	Attack	Only attack modeling
		Defense	Only defense modeling
		Both	Integrates attack and defense modeling
Static or sequential	Can the formalism deal with dependencies and time varying scenarios?	Static	Does not support any dependencies
		Sequential	Supports time and order dependencies
Quantification	Can numerical values be computed using the formalism?	Versatile	Supports numerous generic and diverse metrics
		Specific	Tailored for specific metrics
		No	Does not support quantification
Main purpose	Why was the formalism invented?	Sec. mod.	General security modeling
		Unification	Unification of existing formalism
		Quant.	Provide better methods for quantitative analysis
		Risk	Support risk assessment
		Soft. dev.	Support secure software development
		Int. det.	Automated intrusion detection and response analysis
		Req. eng.	Support security requirements engineering
Extensions	What are added features of the formalism with respect to the state of the art?	Structural	New connectors, extended graph structure
		Computational	How the formalism handles computations (e.g., top-down)
		Quant.	Which computations can be performed (e.g., specific attributes)
		Time	The formalism can handle temporal dependencies
		Order	The formalism can handle order dependencies
		New formalism	Entirely new formalism

Structure	Which graphical structure is the formalism based on?	Tree	Tree (possibly repeated nodes)
		DAG	Directed acyclic graph
		Unspecified	It is not specified whether the models are DAGs or trees
Connectors	What type of connectors does the formalism use?	List of connectors	AND, OR, priority AND, OWA nodes, split gate, trigger, countermeasures, counter leaves, k-out-of-n
Formalization	Is the formalism formally defined?	Formal	Defined using a mathematical framework; with clear syntax and semantics
		Semi-formal	Parts of the definitions are given verbally, parts are precise
		Informal	Models only verbally described
Tool availability	Does a software tool supporting the formalism exist?	Commercial	Commercial software exists
		Prototype	A prototype tool exists
		No	No implementation exists
Case study	Do papers or reports describing case studies exist?	Real(istic)	Real or realistic case study exists
		Toy case study	Described toy case study exists
		No	No documented case study exists
External use	Do papers or reports having a disjoint set of authors from the formalism inventors exist?	Independent	People or institutions who did not invent the formalism have used it
		Collaboration	The formalism has been used by external researchers or institutions in collaboration with its inventors
		No	The formalism has only been used by its inventors or within the inventing institution
Paper count	How many papers exist?	Number	Number of identified papers concerning the formalism[11]
Year	When was the formalism first published?	Year	Before 2013

Table 7.1: Summary of aspects taken into account in the formalism description.

[11] Different versions of the same paper (e.g., an official publication and a corresponding technical report) have been counted as the same publication.

General presentation The first paragraph mentions the name of the formalism, its authors and a list of the most relevant papers relating to the formalism. The year when the approach was proposed is given. Here we also present the main purpose for which the technique was introduced. If nothing is indicated about the formalism structure, it means that it is a generic DAG. If the structure is, more specifically, a tree, then it is indicated either in the formalism's name or in the first paragraph of the description.

Main features In the second paragraph, we briefly explain the main features of the formalism, in particular what its added features are with respect to the state of the art at the time of its invention. Moreover, we state whether the modeling technique is formalized, i.e., whether it complies with proper mathematical definitions.

Quantification The next paragraph in the descriptions of the formalisms focuses on quantitative aspects of the considered methodology. We explain whether the formalism is tailored for a couple of specific parameters or metrics, or whether a general framework has been introduced to deal with computations. In the first case, we list relevant attributes, in the second case, we briefly explain the new algorithms or calculation procedures.

Practical aspects When relevant, we mention industrialized or prototype software tools supporting the described approach. We also indicate when real or realistic scenarios have been modeled and analyzed with the help of the described approach. In this paragraph, we also refer to large research projects and Ph.D. theses applying the methodology. This paragraph is optional.

Additional remarks We finish the description of the formalism by relating it to follow-up methodologies. We mention the formalism's limitations when they have been identified by its authors or other researchers from the field. In this part we also point out other peculiarities related to the formalism. This paragraph is optional.

7.4 Description of the Formalisms

This section describes numerous DAG-based approaches for graphical attack and defense modeling according to the template outlined in Section 7.3. Models gathered within each subsection are ordered chronologically with respect to the year of their introduction.

7.4.1 Static Modeling of Attacks

Attack Trees

Inspired by research in the reliability area, Weiss [Wei91] in 1991 and a couple of years later Amoroso [Amo94] in 1994 proposed to adopt a tree-based concept of visual system reliability engineering to security. Today, *threat trees* [Amo94, SS04, HL02, MHH$^+$09, US 88], *threat logic trees* [Wei91], *cyber threat trees* [OTT$^+$10], *fault* *trees* for attack modeling [SS02] and the *attack specification language* [TLFH01] can be subsumed under *attack trees*, which are AND-OR tree structures used in graphical security modeling. The name attack trees was first mentioned by Salter et al. in 1998 [SSSW98] but is often only attributed to Schneier and cited as [Sch99,Sch04b].

In the attack tree formalism, an attacker's main goal (or a main security threat) is specified and depicted as the root of a tree. The goal is then disjunctively or conjunctively refined into subgoals. The refinement is repeated recursively, until the reached subgoals represent basic actions. Basic actions correspond to atomic components, which can easily be understood and quantified. Disjunctive refinements represent different alternative ways of how a goal can be achieved, whereas conjunctive refinements depict different steps an attacker needs to take in order to achieve a goal [QL04]. In 2005, Mauw and Oostdijk formalized attack trees by defining their semantics and specifying tree transformations consistent with their framework [MO05]. Kienzle and Wulf presented an extensive general procedure for tree construction [KW97] while other researchers were engaged in describing how to generate attack tree templates using *attack patterns* [MEL01, LM01].

Quantification of security with the help of attack trees is an active topic of research [WPWP11]. A first simple procedure for quantification using attack trees was proposed by Weiss [Wei91] and is based on a bottom-up algorithm. In this algorithm, values are provided for all leaf nodes and the tree is traversed from the leaves towards the root in order to compute values of the refined nodes. Depending on the type of refinement, different functional operators are used to combine the values of the children. This procedure allows us to analyze simple aspects, such as the costs of an attack, the time of an attack or the necessary skill level [Wei91, Amo94, SSSW98, Sch99, BFM04, HUJ+04, FCW+05, MO05, BDP06, Yag06, EDRM06, SDP08, HAF+09, LLFH09, ACK10, BP10, TA10, WPWP11]. Whenever more complicated attributes, such as probability of occurrence, probability of success, risk or similar measures are analyzed, additional assumptions, for example, mutual independence of all leaf nodes, are necessary, or methods different from the bottom-up procedure have to be used [Sch99, BFM04, BLP+06, EDRM06, Yag06, JW08, HAF+09, LLFH09, Buo10, ACK10, BFM10, OTT+10, BFM11, MTF11, WWPP11, RSK+12, RKT12a, ZY12]. Propagation of fuzzy numbers that model fuzzy preference relations has initially been proposed in [BFG11] and extended in [BF12]. Using Choquet integrals it is possible to take interactions between nodes into account.

Commercial software for attack tree modeling, such as SecurITree [Ame12] from Amenaza or AttackTree+ [Iso11] from Isograph provides a large database of attack tree templates. Academic tools, including SeaMonster [Mel10] developed within the SHIELDS project [SHI10a] offer visualization and library support. Attack trees may occur in the SQUARE methodology [MHS05]. The entire methodology and, therefore, visualization of attack trees are supported by the SQUARE tool [Car09]. AttackDog [Laz10] was developed as a prototype software tool for managing and evaluating attack trees with voting systems in mind but is believed to be much more widely applicable to evaluating security risks in systems [ACC07]. Numerous case studies during the last decade [MEL01, TLFH01, BFM04, CCF04, EHK+04, HUJ+04, BPU+05, FCW+05, MHS05, ABD+06, KS07, TLM07, CSTH08, GJ08, Mar08, NXYS08, PLC+08, RVOC08, SDP08, HAF+09, LZRL09, MMCJ09] and more recently [CKK10, FBMJ10, MPM10a, MPM+10b, TML10, EPPC11, LDEH11, MCM11, SWX11, WLR11, SS12, ZY12] account for the applicability of the attack tree methodology. Moreover, attack trees are used in large international research projects [EVI11, SHI10a, ATR12, TRE16, US 10]. They have been focus of a multitude of Ph.D. and master's theses [Kie98, Pum99, Mob00, Fos02, Sch04a, Ope05, Kar05, Edg07, Esp07, Hog07, Mäg07, Har10, Jür10, PC10, Roy10, Nie11, Ost11, Sam11, Zon11, Buo12, Koo12, Pos12].

Since attack trees only focus on static modeling and only take an attacker's behavior into account, numerous extensions that include dynamic modeling and a defender's behavior, exist. Except for formalisms involving Bayesian inference techniques, all other DAG-based formalisms refer back to the attack tree methodology. They point out a need for modeling defenses, dynamics and ordered actions, as well as propose computation procedures for probability or highly specified key figures. Neither the name attack trees, nor the initial formalization of Mauw and Oostdijk is universally accepted. Some researchers consider attack trees, threat trees or fault trees to essentially be the same [vL04, MY11, SEN09, And01, SS02] while other researchers point out specific differences [LLFH09, MM08]. As common ground, all mentioned methodologies use an AND-OR tree structure but are divided on what the tree can actually model (attacks, vulnerabilities, threats, failures, etc.).

Augmented Vulnerability Trees

Vulnerability trees [VJ03] have been proposed by Vidalis and Jones in 2003 to support the decision making process in threat assessment. Vulnerability trees are meant to represent hierarchical interdependence between different vulnerabilities of a system. In 2008, Patel, Graham and Ralston [PGR08] extended this model to *augmented vulnerability trees* which combine the concepts of vulnerability trees, fault tree analysis, attack trees and cause–consequence diagrams. The aim of augmented vulnerability trees is to express the financial risk that computer-based information systems face, in terms of a numeric value, called *degree of security.*

The root of a vulnerability tree is an event that represents a vulnerability; the branches correspond to different ways of exploiting it. The leaves of the tree symbolize steps that an attacker may perform in order to get to the parent event. The model, which is not formally defined, uses only AND and OR connectors depicted as logical gates. Vulnerability trees are similar to attack trees, they differ in how the root event is defined (vulnerability event vs. an attacker's goal). A step-wise methodology consisting of a sequence of six steps is proposed in [PGR08] to create an augmented vulnerability tree and analyze security related indices.

The authors of [VJ03] propose attributes on vulnerability trees, including a complexity value that denotes the number of steps that an attacker has to employ in order to achieve his goal, educational complexity, which measures which qualifications an attacker has to acquire in order to exploit a given vulnerability, and the time necessary to exploit a vulnerability. However, the paper [VJ03] does not detail how to compute these attributes. In [PGR08], the model is augmented with two indices: the threat-impact index and the cyber-vulnerability index. The first index, represented by a value from $[0, 100]$, expresses the financial impact of a probable cyber threat. The lower the index the smaller the impact from a successful cyber attack. The second index, also expressed by a value from $[0, 100]$, represents system flaws or undesirable events that would help an intruder to launch attacks. The lower this index, the more secure the system is.

In [TA10], the augmented vulnerability tree approach has been used to evaluate risks posed to a SCADA system exposed to the mobile and the Internet environment.

Augmented Attack Trees

In 2005, Ray and Poolsappasit[12] first developed *augmented attack trees* to provide a probabilistic measure of how far an attacker has progressed towards compromising the system [RP05]. This tree-based approach was taken up by H. Wang et al. in 2006 and extended to allow more flexibility in the probabilistic values provided for the leaf nodes [WLZ06]. When again publishing in 2007, Poolsappasit and Ray used a different definition of augmented attack trees to be able to perform a forensic analysis of log files [PR07]. Using the second definition of augmented attack trees, J. Wang et al. performed an analysis of SQL injection attacks [WPWP10b] and distributed denial of service (DDoS) attacks [WPWP10a]. They also extended augmented attack trees further to measure the quality of detectability of an attack [WPWP10c]. Poolsappasit and Ray formalized attack trees as AND-OR structure where every node is interpreted to answer a specific Boolean question, as co-authors of Dewri et al. [DPRW07, DRPW12]. This formalization is then again extended to augmented attack trees by adding to every node an indicator variable and an additional value with the help of which the residual damage is computed. On the enhanced structure they are able to optimize how to efficiently trade-off between spent money and residual damage.

The various ways of defining augmented attack trees are based on attack trees (Section 7.4.1). In the first definition, attack trees are augmented by node labels that quantify the number of compromised subgoals on the most advanced attack path as well as the least-effort needed to compromise the subgoal on the most advanced path to be able to compute the probability of attack [RP05]. H. Wang et al. generalized this definition from integer values to general weights. Both approaches include tree pruning and tree trimming algorithms to eliminate irrelevant nodes with respect to intended operations (behavior) of a user [WLZ06]. In the second definition, attack trees are augmented by descriptive edge labels and attack signatures. Each edge defines an atomic attack which is described by the label and represents a state transition from a child node to the corresponding parent. An attack signature is a sequence of groups of incidents. From it, a sequence of incidents can be formed, which, when executed, constitutes an atomic attack. The sequences are then exploited to filter log files for relevant intrusion incidences [PR07] and used to describe state transitions in SQL injection attacks using regular expressions [WPWP10b]. Moreover, they are exploited to model state transition in DDoS attacks [WPWP10a] and adapted to provide a measure for quality of service detection, called quality of detectability [WPWP10c]. In an extension of the third definition [DRPW12] the system administrator's dilemma is thoroughly examined. The purpose of this extension is to be able to compute a bounded minimization

[12] In early papers spelled Poolsapassit [RP05, PR07]

problem of the cost of the security measures while also keeping the residual damage at a minimum.

Augmented attack trees were designed with a specific quantitative purpose in mind. The first formalization of augmented attack trees was introduced to compute the probability of a system being successfully attacked. Additionally to increasing the descriptive capabilities of the methodology, the second definition is accompanied by several algorithms that help compute the quality of detectability in [WPWP10c]. As mentioned before, the third definition was advanced in order to solve the system administrator's dilemma. This is achieved by using a simple cost model and a multiobjective optimization algorithm which guides the optimization process of which security hardening measures best to employ.

The authors of the first formalism state that attempts by system administrators to protect the system will not change the outcome of their analysis. A similar shortcoming is suggested for the second formalization.

OWA Trees

In 2005, Yager proposed to extend the AND and OR nodes used in attack trees by replacing them with ordered weighted averaging (OWA) nodes. The resulting formalism is called *OWA trees* [Yag06] and it forms a general methodology for qualitative and quantitative modeling of attacks.

Regular attack trees make use of two (extreme) operators only: AND (to be used when *all* actions need to be fulfilled in order to achieve a given goal) and OR (to be used when the fulfillment of *at least one* action is sufficient to reach a desired result). OWA operators represent quantifiers such as *most*, *some*, *half of*, etc. Thus, OWA trees are well suited to model uncertainty and to reason about situations where the number of actions that need to be satisfied is unknown. OWA trees are static in the sense that they do not take interdependencies between nodes into account. They have been formally defined in [Yag06] using the notion of an *OWA weighting vector*. Since AND and OR nodes can be seen as special cases of OWA nodes, mathematically, attack trees form a subclass of OWA trees. Therefore, algorithms proposed for OWA trees are also suitable for the analysis of attack trees.

In [Yag06], Yager provides sound techniques for the evaluation of success probability and cost attributes on OWA trees. For the probability attribute, he identifies two approaches that can be explained using two different types of attackers. The first approach assumes that the attacker is able to try all available actions until he finds one that succeeds. Since in most situations such an assumption is unrealistic, the author proposes a second model, where an attacker simply chooses the action with the highest probability of success. Furthermore, [Yag06] presents two algorithms for computing the success probability attribute: one assumes independent actions which leads to a simpler calculation procedure, the other can deal with dependent actions. Finally, the author discusses how to join the two attributes together, in order to correctly compute the cheapest and most probable attack.

In [BFG11], Bortot et al. proposed the use of Choquet integrals in order to reason about OWA trees involving dependent actions.

Parallel Model for Multi-Parameter Attack Trees

In 2006, Buldas et al. initiated a series of papers on rational choice of economically relevant security measures using attack trees. The proposed model is called *multi-parameter attack trees* and was first introduced in [BLP+06]. Between 2006 and 2010, researchers from different research institutes in Estonia published six follow-up papers [BM07, JW07, JW08, WJ10, JW10, Nii10], extending and improving the original model proposed in [BLP+06].

Most approaches for quantitative analysis using attack trees, prior to [BLP+06], focus on one specific attribute, e.g., cost or feasibility of an attack. In reality, interactions between different parameters play an important role. The aim of the mentioned series of papers was to study how tree computations must be done when several interdependent parameters are considered. The model of multi-parameter attack trees assumes that the attacker's behavior is rational. This means that attacks are considered unlikely if their costs are greater than the related benefits and that the attacker always chooses the most profitable way of attacking. The parallel model for multi-parameter attack trees has been studied in [BLP+06, BM07, JW07, JW08, JW10, Jür10]. This model assumes that all elementary attacks take place simultaneously, thus the attacker does not base his decisions on success or failure of some of the elementary attacks.

Multi-parameter attack trees concentrate on the attribute called expected attacker's outcome. This outcome represents a monetary gain of the attacker and depends on the following parameters: gains of the attacker in case the attack succeeds, costs of the attack, success probability of the attack, probability of getting caught and expected penalties in case of being caught. First, a game theoretical model for estimation of the expected attacker's outcome was proposed by Buldas et al. [BLP+06], where values of all parameters are considered to be precise point estimates. In [JW07], Jürgenson and Willemson extend the computation methods proposed in [BLP+06] to the case of interval estimations. Later it turned out that the computational model from [BLP+06] was imprecise and inconsistent with the mathematical foundations of attack trees introduced in [MO05]. Hence, an improved approach for the parallel attack tree model was proposed by Jürgenson and Willemson [JW08]. Since this new approach requires exponential running time to determine the possible expected outcome of the attacker, an optimization solution, based on a genetic algorithm for fast approximate computations, has been proposed by the same authors in [JW10].

In [BM07], Buldas and Mägi applied the approach developed in [BLP+06] to evaluate the security of two real e-voting schemes: the Estonian E-voting System in use at the time and the Secure Electronic Registration and Voting Experiment performed in the USA in 2004. A detailed description of this case study is given in the master's thesis of Mägi [Mäg07]. A prototype computer tool supporting the security analysis using the multi-parameter attack trees has been implemented [And10a] and described in [And10b].

In Section 7.4.2, we describe the serial model for multi-parameter attack trees, which extends the parallel model with an order on the set of elementary components.

Extended Fault Trees

Extended fault trees (EFT) were presented by Masera et al. at the European safety and reliability conference in 2007 [MFC07] and published in an extended journal version in 2009 [FMC09]. The formalism aims at combining malicious deliberate acts, which are generally captured by attack trees (Section 7.4.1), and random failures, which are often associated with classical fault trees (Section 7.4.1).

Extended fault trees and attack trees are structurally similar. The main difference between the two formalisms is in the type of basic events that can be modeled. In EFT basic events can represent both non-malicious, accidental failures as well as attack steps or security events. Basic events of attack trees usually correspond to a malicious attacker's actions only. Logical AND and OR gates are explicitly represented in the same way as in classical fault trees. A step by step model construction process is described in [FMC09], defining how existing fault trees can be extended with attack-related components to form extended fault tree models. The modeling technique complies with proper mathematical foundations, directly adopted from fault trees as defined in the safety and reliability area.

Quantification capabilities are focused on the computation of the probability of occurrence of the top-event (root node). Generic formulas from fault tree quantitative analysis are recalled in [FMC09], including treatment of independent or mutually exclusive events. However, no concrete examples of quantification are provided.

A simple example, analyzing the different failure and attack scenarios leading to the release of a toxic substance by a chemical plant, is described in [FMC09]. No particular tool has been developed to support extended fault trees, however, all classical fault tree tools may be used directly.

One of the limitations explicitly stressed by the inventors of extended fault trees is that they do not take into account time dynamics.

7.4.2 Sequential Modeling of Attacks

Cryptographic DAGs

Meadows described *cryptographic DAGs* in 1996 (proceedings published in 1998), in order to provide a simple representation of an attack process [Mea96]. The purpose of the formalism is limited to visual description. The attack stages of the overall attack process correspond to the nodes of a DAG. The difficulty of each stage is shown by a color code. In 1996, the novelty of cryptographic DAGs was to provide a simple representation technique of sequences and dependencies of attack steps towards a given attacker's objective.

From a modeling point of view, each stage (represented as a colored box) contains a textual description of atomic actions needed for the realization of the stage. Arrows represent dependencies between the boxes. A simple arrow indicates that one stage is needed to realize another stage. Two arrows fanned out symbolize that one stage enables another one repeatedly. More generally speaking, cryptographic DAGs are an informal formalism targeted at high-level system descriptions.

Cryptographic DAGs do not support any type of quantification.

Cryptographic DAGs have been used in [Mea96] to demonstrate attacks on cryptographic protocols (with SSL and Needham-Schroeder scheme as a use cases), however, this representation technique may be used to model other types of attacks as well.

This formalism allows the representation of sequences of attack steps and dependencies between those steps, but cannot capture static relations like AND and OR. Moreover, the clarity and usability of the models depends heavily on the text inside the boxes, which is not standardized.

Fault Trees for Security

Fault tree analysis was born in 1961 and has initially been developed into a safety, reliability and risk assessment methodology [Wat61, VGRH81, SVD+02]. A short history of non-security related fault trees was published by Ericson II [Eri99] in 1999. Fault trees have also been used in software analysis [LH83,Lev95,HWS+02, HWS+07] and were even considered to be equivalent to attack trees by Steffen and Schumacher [SS02]. In 2003, Brooke and Paige adopted *fault trees for security*, extending the classical AND-OR structure of attack trees (Section 7.4.1), to include well-known concepts from safety analysis [BP03].

Based on an AND-OR structure, three additional connectors (priority AND, exclusive OR and inhibit), specific node types (basic, conditioning, undeveloped, external and intermediate) as well as transfer symbols (transfer in, transfer out) to break up larger trees are adopted from fault tree analysis in its widest sense. Fault trees for security are an aid to the analysis of security-critical systems, where first an undesired (root) event is identified. Then, new events are constructed by inserting connectors that explicitly identify the relationship of the events to each other. Several rules, like the *no miracle* rule, the *complete the gate* rule and the *no gate to gate* rule are adopted directly from fault trees. Construction stops when there are no more uncompleted intermediate events. In the end, a completed fault tree serves as an *attack handbook* by providing information about the interactions by which a security critical system fails.

In [BP03], Brooke and Paige state that in computer security "it is difficult to assign useful probabilities to the events". Consequently probabilistic quantitative analysis is debatable. Instead, the authors recommend that risk analysis is performed, which answers how the system fails based on the primary events (leaf nodes).

While [BP03] only provides a toy example, the authors state that any tool used in fault tree analysis can be used. They refer to [Dep03] as a good overview of available programs.

Bayesian Networks for Security

Starting in 2004, different researchers proposed, seemingly independently, to adopt *Bayesian networks*, whose origin lies in artificial intelligence, as a security modeling technique [Pea86, Pea88, Nea03, JN07]. Bayesian networks are also known as *belief network* or *causal network*. In Bayesian networks, nodes represent events or objects and are associated with probabilistic variables. Directed edges represent causal dependencies between nodes. Mathematical algorithms developed for Bayesian networks are suited to solve probabilistic questions on DAG structures. They achieve reasonable running times in practice while being exponential in the worst case. They are polynomial if the DAG is actually a tree.

According to Qin and Lee, the objective of Bayesian networks for security is to "use probabilistic inference techniques to evaluate the likelihood of attack goals and predict potential upcoming attacks" [QL04]. They proposed the following procedure that converts an attack tree into a Bayesian network. Every node in the attack tree is also present in the Bayesian network. An OR relationship from an attack tree is modeled in the Bayesian network with edges pointing from refining nodes that represent causes into the corresponding refined nodes that represent consequences. Deviating from regular attack trees, an AND relationship is assumed to have an explicit (or implicit) order in which the actions have to be executed. This allows us to model the AND relationship by a directed path, which starts from the first (according to the order) child and ends with the parent node. Dantu et al. follow a different strategy when using Bayesian networks to model security risk management starting from behavior-based attack graphs[13] [DLK04, DK05, DKAL07, DKaWC09]. When processing multi-parameter attack trees with estimated parameter values (Section 7.4.1), Jürgenson and Willemson use Qin and Lee's conversion of an attack tree to a Bayesian network [JW07]. An et al. propose to add a temporal dimension and to use *dynamic Bayesian networks* for intrusion detection without specifying how the graph is set up [AJC06]. Althebyan and Panda use knowledge graphs and dependency graphs as basis for the construction of a Bayesian network [AP08]. They analyze a specific type of insider attack and state that their computational procedures were inspired by Dantu et al. Another approach involving Bayesian networks is described by Xie et al. who analyze intrusion detection systems [XLO$^+$10]. They state that the key to using Bayesian networks is to "correctly identify and represent relevant uncertainties" which governs their setup of the Bayesian network.

Bayesian networks are used to analyze security under uncertainty. The DAG structure is of great value because it allows the use of efficient algorithms. On the one hand there exist efficient inference algorithms that compute a single query (variable elimination, bucket-elimination and importance, which are actually equivalent according to Pouly and Kohlas [PK11]) and on the other hand there are inference algorithms that compute multiple queries at once (bucket tree algorithm and Lauritzen-Spiegelhalter algorithm). In fact, the efficiency of these algorithms can be seen as main reason for the success of Bayesian networks since querying general graphs is an NP-hard problem [Arn85, Bod93]. Another strength of Bayesian networks is their ability to update the model, i.e., compute a posteriori distribution when new information is available.

We have not found any dedicated tools for the analysis of Bayesian networks for security. However, numerous tools exist that allow a visual treatment of standard Bayesian networks. One such tool is the Graphical Network Interface (GeNIE) that

[13] The authors do not appear to make a distinction between attack trees and attack graphs. Since their methodology is only applicable to cycle-free structures and they do not mention how to deal with cycles, we assume that the methodology is actually based on attack DAGs or attack trees.

uses the Structural Modeling, Inference and Learning Engine (SMILE) [Dec13]. It was, for example, used in [NJL+09] to analyze the interoperability of a small cluster of services and mentioned as hypothetical use in [FSEJ08]. Another one, called MulVAL [OGA05], was actually developed for attack graphs (Section 7.6.2), but used in [XLO+10] to implement a Bayesian network model. A third tool, tailored to statistical learning with Bayesian networks is bnlearn [Scu10].

There also exist isolated papers that promote the use of Bayesian networks in security without any relation to attack trees or attack graphs. Houmb et al. quantify security risk level from Common Vulnerability Scoring System (CVSS) estimates of frequency and impact using Bayesian networks [HFE09]. Feng and Xie also use Bayesian networks and provide an algorithm of how to merge two sources of information, expert knowledge and information stored in databases, into one graph [FX12]. Note that in this section we have gathered approaches that rely on Bayesian networks whose construction starts from graphs that do not contain any cycles. Graphical models that make use of Bayesian networks and that initially contain cycles are treated under the heading Bayesian attack graphs, ones that include defenses are treated in Section 7.4.4.

Bayesian Attack Graphs

Bayesian Attack Graphs combine (general) attack graphs (Section 7.6.2), with computational procedures of Bayesian networks (Section 7.4.2). However, since Bayesian inference procedures only work on cycle-free structures, the formalism includes instructions on how to remove any occurring cycles. Hence, any final Bayesian attack graph is acyclic. After the elimination of cycles, Bayesian attack graphs model causal relationships between vulnerabilities in the same way as Bayesian networks. Bayesian attack graphs were first proposed by Liu and Man in order to analyze network vulnerability scenarios with the help of Bayesian inference methods in 2005 [LM05]. Therefore, the formalism advances computational methods in security where uncertainty is considered.

The formalism of Man and Liu is not the only fusion of attack graphs and Bayesian networks. Starting in 2008 a group of researchers including Frigault, Noel, Jajodia and Wang published a paper on a modified version of Bayesian attack graphs. Their goal was to be able to calculate general security metrics regarding information system networks which also contain probabilistic dependencies [NJWS10, FW08]. Later they extended the formalism, using a second copy of the model as time slice, to also capture dynamic behavior in so called *dynamic Bayesian networks* [FWSJ08]. In 2012, Poolsappasit et al. revisited the framework to be able to deal with asset identification, system vulnerability and connectivity analysis as well as mitigation strategies [PDR12]. All three approaches eliminate cycles that possibly exist in the underlying attack graph. A shortcoming of Liu and Man is that they do not provide a specific procedure on how to achieve this. The group including Frigault refers to a paper on attack graphs [WIL+08] which removes cycles through an intricate procedure. Poolsappasit et al. state that they rather analyze *why an attack can happen* and not *how an attack can happen* and therefore, “cycles can be disregarded using the monotonicity constraint” mentioned in [AWK02].

Since Bayesian attack graphs are cycle-free, evaluation on them can make use of Bayesian inference techniques. For this it is necessary to provide probabilistic information. The three approaches differ in how they compute quantitative values. Liu and Man provide edge probabilities [LM05], Frigault et al. give conditional probability tables for nodes which are estimated according to their CVSS score [FW08] and Poolsappasit et al. use (local) conditional probability distributions for nodes [PDR12]. Furthermore, Poolsappasit et al. augment Bayesian attack graphs with additional nodes and values representing hardening measures (defenses). On the augmented structure they propose a genetic algorithm that solves a multiobjective optimization problem of how to assess the risk in a network system and select optimal defenses [PDR12].

The research group including Wang uses a Topological Vulnerability Analysis (TVA) tool [JNO05, NEJ+09] to create the attack graphs that serve as basis for constructing Bayesian attack graphs. Poolsappasit et al. have developed an unreferenced in-house tool that allows them to compute with conditional probability distributions.

Wang et al. [FW08, FWSJ08] state that their work is also based on a paper by An et al. [AJC06], who use Bayesian networks without cycles for modeling risks of violating privacy in a database.

Compromise Graphs

McQueen et al. introduced *compromise graphs* in 2006 [MBFB06]. Compromise graphs are based on directed graphs[14] and are used to assess the efficiency of technical security measures for a given network architecture. The nodes of a compromise graph represent the phases of an attack, detailing how a given target can become compromised. The edges are weighted according to the estimated time required to complete the corresponding phase for this compromise. The overall time needed for the attacker to succeed is computed and compared along different defensive settings, providing a metric to assess and compare the efficiency of these different defensive settings.

The formalism has a sound mathematical formalization: a time to compromise (TTC) metric is modeled for each edge as a random process combining three subprocesses. Each of these processes has a different probability distribution (mixing exponential, gamma and beta-like distributions). The values for the process model parameters are based on the known vulnerabilities of the considered component and the estimated skill of the attacker. A complete description and justification of such a stochastic modeling are provided by the same authors in a previous paper [MBFB05]. In compromise graphs, five types of stages, corresponding to the vertices of the graph, are modeled: recognition, breaching the perimeter, penetration, escalation of privilege, damage.

Compromise graphs are used to evaluate the efficiency of security measures, such as system hardening, firewalls or enhanced authentication. This is achieved by comparing the shortest paths (in terms of TTC) of compromise graphs with and without such measures in place.

[14] The authors do not state whether these directed graphs are acyclic or not, but the description of compromise graphs and their examples led us to consider compromise graphs as DAGs.

The approach is illustrated in [MBFB06] by modeling attacks on a SCADA system.

Leversage and Byres adopt a similar approach in [LB08, LB07], called state-time estimation algorithm (STEA), directly inspired by McQueen et al. They combine a slightly modified TTC calculation approach with a decomposition of the attack according to the architectural areas of the targeted system.

Enhanced Attack Trees

Enhanced attack trees have been introduced by Çamtepe and Yener to support an intrusion detection engine by modeling complex attacks with time dependencies. This model was first described in a technical report [ÇY06] in 2006. One year later, the corresponding proceedings [ÇY07] were published.

In addition to classical AND and OR gates, enhanced attack trees rely on the use of a new gate, the ordered AND, which allows us to capture sequential behavior and constraints on the order of attack steps. The model of enhanced attack trees has sound mathematical foundations. Additionally to the formalism description, [ÇY07] devises a new technique for the detection of attacks. The new technique is based on automata theory and it allows us to verify completeness of enhanced attack tree models with respect to the observed attacks.

The quantification capabilities described in [ÇY07] are directly related to intrusion detection (probability of a given attack occurring based on a set of observed events). A confidence attribute measured in percent is defined for subgoals as “the chance of reaching the final goal of the attacker when a subgoal is accomplished”. It is computed as the ratio of all accomplished events until a subgoal is realized, over all events of the modeled scenario. This attribute aims at supporting an early warning system, supporting decision-making and reaction before actual damages occur. Moreover, [ÇY07] introduces an original parameter called *time to live* which allows us to express that some steps are only available in a given time window.

In [MKY12], Mishra et al. also make use of ordered AND operators, referring to [ÇY07]. The authors visually describe Stuxnet and similar attacks, but do not use Çamtepe and Yener’s rigorous formalization to analyze the models.

Vulnerability Cause Graphs

Vulnerability cause graphs (VCG) were invented in 2006 by Ardi et al. as a key element of a methodology that supports security activities throughout the entire software development lifecycle [ABS06].

The formalism can be seen as a root cause analysis for security-related software failures because it relates vulnerabilities with their causes. In a VCG, every node except for one, has an outgoing directed edge. The single node without a successor is called the *exit node* and represents the considered vulnerability. All other nodes represent causes. The predecessor–successor (parent–child) relationship shows how certain conditions (nodes) may cause other conditions (nodes) to be a concern. In an improved version of VCGs [BASD06], nodes can be simple, compound or conjunctions. Simple nodes represent conditions that may lead to a vulnerability. Compound nodes facilitate reuse, maintenance and readability of the models. Conjunctions represent groups of two or more nodes. Contrarily, disjunctions occur if a node has two or more predecessors. In this case, the original node may have to be considered if either of its predecessors has to be considered. Finally, if the causes have to follow a certain order, they are modeled as sequences of nodes. To construct a VCG, the exit node is taken as a starting point and refined with causes.

In VCGs, nodes can be annotated as *blocked* if the underlying causes are mitigated. The *blocked* flag allows the user to compute whether the underlying vulnerability (exit node) is also mitigated. VCGs are also equipped with a notion of graph transformations that do not change whether the vulnerability is mitigated or not. The transformations include conversions of conjunctions, reordering of sequences, combination of nodes, conversion to compound nodes as well as derived transformations.

In [BASD06] the vulnerability CVE-2003-0161, in [BS07] the vulnerability CVE-2005-2558 and in [MCM$^+$09] the vulnerability CVE-2005-3192 is analyzed with the help of VCGs. Furthermore, [CH07] contains an additional three case studies on common software vulnerabilities which have been performed using VCGs. The SHIELDS project [SHI10a] has developed a software tool GOAT [SHI10c] to be used in conjunction with VCGs.

VCGs were developed as part of a comprehensive methodology to reduce software vulnerabilities that arise in ad hoc software development. They are the starting point to build security activity graphs (Section 7.4.3). By introducing compound nodes, the inventors of the formalism have created a model that allows different layers of abstraction, which in turn introduced a problematic design decision of how many layers of abstraction are needed.

Dynamic Fault Trees for Security

In 2009, Khand [Kha09] adapted several dynamic fault tree [DBB90, DBB92] gates to attack trees, in order to add a dynamic dimension to classical attack trees. The aim of the formalism is similar to that of attack trees (Section 7.4.1).

To overcome limitations of static fault trees, dynamic fault trees [DBB90, DBB92] were invented by Dugan et al. in the early 1990s. They aim at combining the dynamic capacities of Markovian models with the *look and feel* of fault trees. To achieve this, four dynamic gates are used: the priority AND (PAND), the sequence gate (SEQ), the functional dependency gate (FDEP) and the cold spare gate (CSP). Khand reuses directly the three first gates (although renaming FDEP gates by CSUB, for Conditional Subordination, gates), leaving out the CSP gates. The PAND gate reaches a success state if all of its input are realized in a preassigned order (from left to right in the graphical notation). The SEQ gate allows us to model that a series of events necessarily occurs in exactly one order (from left to right in the graphical notation). Once all the input events are realized, the gate is verified. The CSUB gate models the need of the realization of a trigger event to allow a possible realization of others events. The Dynamic fault tree combines dynamic gates with classical logical gates (AND and OR). Dynamic gates are formally defined with truth tables in [Kha09] and by Markov processes in the general definitions of dynamic fault trees from the safety literature [DBB90, DBB92] (although the description is still incomplete [Bou07]).

There are no quantification aspects developed in [Kha09].

The paper by Khand does not specify which tool to use in order to treat the models, but several tools exist for dynamic fault trees in the reliability area, e.g., Galileo [DSC00].

In safety studies, quantifications associated with dynamic fault trees are usually made using Markovian analysis techniques; those may be used here also although nothing is said about computational aspects.

Serial Model for Multi-Parameter Attack Trees

In 2010, the parallel model for multi-parameter attack trees (Section 7.4.1) has been extended by adding a temporal order on the set of elementary attacks [WJ10]. This new methodology is called *serial model for multi-parameter attack trees* and was studied further in [Jür10, Nii10] and [BS12].

The model described in [Jür10] and [Nii10] assumes that an adversary performs the attacks in a given prescribed order. In [BS12], the authors introduce the so-called fully-adaptive adversary model, where an attacker is allowed to try atomic attacks in an arbitrary order which is not fixed in advance and can be modified based on the results of the previous trials. With either order, the serial approach allows for a more accurate modeling of an attacker's behavior than the parallel approach. In particular, the attacker can skip superfluous elementary attacks and base his decisions on success or failure of the previously executed elementary attacks.

In [WJ10], an efficient algorithm for computing an attacker's expected outcome assuming a given order of elementary attacks is provided. Taking temporal dependencies into account allows the attacker to achieve a better expected outcome than when the parallel model (Section 7.4.1) is used. As remarked in [JW10], finding the best permutation of the elementary attacks in the serial model for multi-parameter attack trees may turn computing the optimal expected outcome into a super-exponential problem. In [Nii10], Niitsoo proposed a decision-theoretical framework which makes it possible to compute the maximal expected outcome of a goal oriented attacker in linear time. In [BS12], Buldas and Stepanenko propose a game theoretical framework to compute upper bounds of the utility of fully-adaptive adversaries.

A prototype computer tool supporting the security analysis using the serial model of multi-parameter attack trees has been implemented [And10a] and has been described in [And10b].

A thorough comparison of the parallel and the serial model for multi-parameter attack trees has been given in the Ph.D. thesis of Jürgenson [Jür10]. Baca and Petersen mention that in order to use parametrized attack trees, the user needs to have a good understanding of the motivations of the attacker [BP10]. To overcome this difficulty, cumulative voting is used in countermeasure graphs (Section 7.4.3).

Improved Attack Trees

Improved attack trees aim at dealing with security risks that arise in space-based information systems. They were proposed by Wen-ping and Wei-min [WW11] in 2011 to more precisely describe attack on the information transmitting links, acquisitions systems and ground-based supporting and application systems.

The formalism is based on attack trees and explicitly incorporates the use of the sequential AND operator. It is not defined in a formal way. Improved attack trees rely heavily on the description by Schneier and only detail how to specifically compute the system risk.

Improved attack trees provide a specific formula to evaluate a risk value for each leaf node. Starting from these risk values, the risk rate and the risk possibility are computed and multiplied to compute the overall system risk. The formulas distinguish between OR, AND and sequential AND nodes.

7.4.3 Static Modeling of Attacks and Defenses

Anti-Models

Anti-models [vLBLJ03] have been introduced by van Lamsweerde et al. in 2003. They are closely related to AND-OR goal-refinement structures [vLL00] (sometimes called goal models) used for goal analysis in requirements engineering. Anti-models extend such AND-OR goal-refinement structures with the possibility of modeling malicious and intentional obstacles to security goals, called anti-goals. They can be used to generate subtle attacks, discard non-realizable or unlikely ones and derive more effective customized resolutions.

In [vLBLJ03] and later in an extended version [vL04], van Lamsweerde et al. provide a six-step procedure for a systematic construction of anti-models. First, anti-goals, representing attackers' goals, are obtained by negating confidentiality, privacy, integrity, availability, authentication or non-repudiation requirements. For each anti-goal, the questions "Who?" and "Why?" are asked to identify potential classes of attackers and their higher-level anti-goals. An AND-OR refinement process is then applied to reach terminal anti-goals that are realizable by the attackers. The resulting AND-OR anti-models relate "attackers, their anti-goals, referenced objects and anti-operations (necessary to achieve their anti-goals) to the attackees, their goals, objects, operations and vulnerabilities". The construction of anti-models is only informally presented in [vLBLJ03]. Formal techniques developed for AND-OR goal-refinement structures (such as refinement obstacle trees) [vLL00] can be used for the generation and analysis of anti-models. In particular, real-time temporal logic can be employed to model anti-goals as sets of attack scenarios. After identifying possible anti-goals, countermeasures expressed as epistemic extensions of real-time temporal logic operators are selected based on severity or likelihood of the corresponding threat and non-functional system goals that have been identified earlier. Possible resolutions tactics, inspired by solutions proposed for the analysis of non-functional requirements in software engineering, are described in [vLL00] and [vL04]. Applying resolution operators yields new security goals to be integrated in the model. These new goals are then again refined with the help of AND-OR structures. These, in turn, may require a new round of anti-model construction and analysis.

Anti-models do not include quantitative analysis of security goals or anti-goals.

Defense Trees

Defense trees[15] are attack trees where leaf nodes are decorated with a set of countermeasures. They were introduced by Bistarelli et al. in 2006 [BFP06]. The approach combines qualitative and quantitative aspects and serves general security modeling purposes.

The approach proposed by Bistarelli et al. was a first step towards integrating a defender's behavior into models based on attack trees. The analysis methodology for defense trees proposed in [BFP06] and [BDP06] uses rigorous and formal techniques, such as calculation of economic indices and game theoretical solution concepts. However, the model itself is only introduced verbally and a formal definition is not given.

In [BFP06], the return on attack (ROA) and return on investment (ROI) indices are used for quantitative analysis of defense trees from the point of view of an attacker and a defender, respectively. The calculation of ROI and ROA is based on the following parameters: costs, impact, number of occurrences of a threat and gain. The indices provide a useful method to evaluate IT security investments and to support the risk management process. In [BDP06], game theoretical reasoning was introduced to analyze attack–defense scenarios modeled with the help of defense trees. In this paper, a defense tree represents a game between two players: an attacker and a defender. The ROI and ROA indices, are used as utility functions and allow us to evaluate the effectiveness and the profitability of countermeasures. The authors of [BDP06] propose using Nash equilibria to select the best strategy for the players.

In [BPT08], defense trees have been extended to so-called *CP-defense trees*, where modeling of preferences between countermeasures and actions is possible. Transforming CP-defense trees into answer set optimization (ASO) programs, allows us to select the most suitable set of countermeasures, by computing the optimal answer set of the corresponding ASO program. Formalisms such as attack–defense trees (Section 7.4.3), and attack countermeasure trees (Section 7.4.3) extended defense trees by allowing defensive actions to be placed at any node of the tree and not only at leaf nodes.

Protection Trees

Protection trees are a tree-based formalism which allow a user to allocate limited resources towards the appropriate defenses against specified attacks. The methodology was invented by Edge et al. in 2006, in order to incorporate defenses in the attack tree methodology [EDRM06].

[15] Papers by Bistarelli et al. use British English, thus originally, the name of their formalism is *defence trees*.

Protection trees are similar to attack trees since both decompose high-level goals into smaller manageable pieces by means of an AND-OR tree structure. The difference is that in protection trees the nodes represent protections. A protection tree is generated from an already established attack tree by finding a protection against every leaf node of the attack tree. Then the attack tree is traversed in a bottom-up way and new protection nodes are added to the protection tree if the protection nodes do not already cover the parent attack node.

The AND-OR structure of protection trees is enriched with three metrics, namely probability of success, financial costs and performance costs on which the standard bottom-up approach is applied [EDRM06, ERG⁺07, Edg07]. In [DEMR10], an additional metric, the impact, helps to further prioritize where budget should be spent.

The formalism has been investigated in case studies on how the U.S. Department of Homeland Security can allocate resources to protect their computer networks [EDRM06], how an attack on an online banking system can be mitigated cost-efficiently [ERG⁺07], how to cheaply protect against an attack on computer and RFID networks [DEMR10] as well as a mobile ad hoc network [Edg07]. When evaluating which defenses to install, the authors propose to first prune the tree according to the attacker's assumed capabilities. A larger, more applied case study to "evaluate the effectiveness of attack and protection trees in documenting the threats and vulnerabilities present in a generic Unmanned Aerial Systems (UAS) architecture" was performed by Cowan et al. [CGP08].

In [ERG⁺07] a slightly different algorithm for the creation of a protection tree was proposed. Here a designer starts by finding defenses against the root of an attack tree instead of the leaves, as in [EDRM06, Edg07]. An approach similar to protection trees has been proposed in [RHCM12] to deal with the problem of threat modeling in software development. The paper uses so-called identification trees to identify threats in software design and introduces the model of mitigation trees to describe countermeasures for identified threats. Despite an obvious modeling analogy between protection trees and mitigation trees, no connection between the two models has been made explicit in the literature.

Security Activity Graphs

In 2006, Ardi, Byers and Shahmehri introduced a formalism called *security activity graphs* (SAGs). The methodology was invented in order to "improve security throughout the software development process" [ABS06]. SAGs depict possible vulnerability cause mitigations and are algorithmically generated from vulnerability cause graphs (Section 7.4.2).

SAGs are a graphical representation of first order predicate calculus and are based loosely on ideas from fault tree analysis. In [ABS06] the root of a SAG is associated with a vulnerability, taken from a vulnerability cause graph. The vulnerability mitigations are modeled with the help of activities (leaf nodes). The syntax, furthermore, consists of AND-gates, OR-gates and split gates. The AND and OR-gates strictly follow Boolean logic, whereas the split gate allows one activity to be used in several parent activities, essentially creating a DAGs structure. The syntax of

SAGs was changed in [BS08] for a more concise illustration of the models. Split gates no longer appear in the formalism. The functionality that simple activities can be distinguished from compound activities (complex activities that may require further breakdown) was added. Moreover, cause references (possible attack points) serve as placeholders for a different SAG associated with a particular cause.

In the SAG model, Boolean variables are attached to the leaves of the SAG. A Boolean variable corresponding to an activity is true when it is implemented *perfectly* during software development otherwise, it is false. Then a value corresponding to the root of the SAG is deduced in a bottom-up fashion according to Boolean logic.

Visually representing SAGs is supported by the tools SeaMonster [MSH$^+$08] and GOAT [SHI10c]. Furthermore, SAGs have been used in [BS08, BS07] to model the vulnerability CVE-2005-2558 in MySQL that leads to denial of service or arbitrary code execution.

Even though the model was devised in order to aid the software development cycle, the authors explicitly state that SAGs "lend themselves to other applications such as process analysis". SAGs are the middle step of a broader three-step approach for secure software development, with vulnerability cause graphs as a first step, and process component definition as a final step. In 2010 SAGs were replaced by security goal models (Section 7.4.4)

Attack Countermeasure Trees

Roy et al. proposed *attack countermeasure trees* (ACTs) [RKT10a,RKT10b] in 2010 as a methodology for attack and defense modeling which unifies analysis methods proposed for attack trees (Section 7.4.1) with those introduced on defense trees (Section 7.4.3). The main difference of ACTs with respect to defense trees is that in ACTs defensive measures can be placed at any node of the tree. Also, the quantitative analysis proposed for defense trees is extended by incorporating probabilistic analysis into the model. ACTs were first introduced in [RKT10b] and then further developed in [RKT12a].

ACTs may involve three distinct classes of events: attack events, detection events and mitigation events. The set of classical AND and OR nodes, as defined for attack trees, is extended with the possibility of using k-out-of-n nodes. Generation and analysis of attack countermeasure scenarios is automated using minimal cut sets (mincuts). Mincuts help to determine possible ways of attacking and defending a system and to identify the system's most critical components.

A rigorous mathematical framework is provided for quantitative analysis of ACTs in [RKT10b] and [RKT12a]. The evaluation of the ROI and ROA attributes, as proposed for defense trees (Section 7.4.3), has been extended by adding the probability of attack, detection and mitigation events. The authors of [RKT12a] provide algorithms for probability computation on trees with and without repeated nodes. With the help of probability parameters, further metrics, including cost, impact, Birnbaum's importance measure and risk, are evaluated. The use of the Birnbaum's importance measure (also called reliability importance measure, in the case of fault trees) is used to prioritize defense mechanisms countering attack events. Furthermore, in [RKT12a], Roy et al. propose a cubic algorithm to select an optimal set of countermeasures for an ACT. This addresses the problem of state-space explosion that the intrusion response and recovery engine based on attack-response trees (Section 7.4.4) suffers from. Finally, in [RKT12b] the problem of selecting an optimal set of countermeasures with and without having probability assignments has been discussed.

The authors of [RKT12a] implemented a module for automatic description and evaluation of ACTs in a modeling tool called Symbolic Hierarchical Automated Reliability and Performance Evaluator [TS09]. This implementation uses already existing algorithms for the analysis of fault trees and extends them with algorithms to compute costs, impact and risk. Case studies concerning attacks on the Border Gateway Protocol (BGP), SCADA systems and malicious insider attacks have been performed using ACTs, as described in the master's thesis of Roy [Roy10].

The model of attack countermeasure trees is similar to attack–defense trees. The main differences between the two models are listed in Section 7.4.3.

Attack–Defense Trees

In 2010, Kordy et al. proposed *Attack–defense trees* (ADTrees) [10KMRS]. They allow for the illustration of security scenarios that involve two opposing players: an attacker and a defender. Consequently it is possible to model interleaved attacker and defender actions qualitatively and quantitatively. ADTrees can be seen as merging attack trees (Section 7.4.1) and protection trees (Section 7.4.3) into one formalism. They are the topic of discussion in this thesis.

In ADTrees, both types of nodes, attacks and defenses, can be conjunctively as well as disjunctively refined. Furthermore, the formalism allows for each node to have one child of the opposite type. Children of opposite type represent countermeasures. These countermeasures can be refined and countered again. Two sets of formal definitions build the basis of ADTrees: a graph-based definition and an equivalent term-based definition. The graph-based definition ensures a visual and intuitive handling of ADTree models. The term-based representation allows for formal reasoning about the models. The formalism is enriched through several semantics that allow us to define equivalent ADTree representations of a scenario [12KMRS]. The necessity for multiple semantics is motivated by diverse applications of ADTrees, in particular unification of other attack tree related approaches and suitability for different kinds of computations. In [11KPS], the authors showed that, for a wide class of semantics (i.e., every semantics induced by a De Morgan lattice), ADTrees extend the modeling capabilities of attack trees without increasing the computational complexity of the model. In [12KMRS] every frequently used semantics for ADTrees has been characterized by finite axiom schemes, which provides an operational method for defining equivalent ADTree representations. The authors of [10KMMS], have established a connection between game theory and graphical security assessment using ADTrees. More precisely, ADTrees under a semantics derived from propositional logics are shown to be equally expressive as two-player binary zero-sum extensive form games.

The standard bottom-up algorithm, formalized for attack trees in [MO05], has been extended to ADTrees in [12KMRS]. This required the introduction of four new operators (two for conjunction and disjunction of defense nodes and two for countermeasure links) [12KMRS]. Together with the two standard operators (for conjunctions and disjunctions of attack nodes) and a set of values, the six operators form an attribute domain. Specifying attribute domains allows the user to quantify a variety of security relevant parameters, such as time of attack, probability of defense, scenario satisfiability and environmental costs. The authors of [12KMRS] show that every attribute for which the attribute domain is based on a semiring can be evaluated on ADTrees using the bottom-up algorithm. Proper specification of attribute domains in terms of questions in natural language was presented in [12KMS].

An extensive case study on an existing, real-life RFID goods management system was performed by academic and industrial researchers with different backgrounds [12BKMS]. The case study resulted in specific guidelines about the use of attributes on ADTrees. A software tool, called the ADTool, see Section 5.6, supporting ADTree methodology, has been developed as one of the outcomes of the ATREES project [ATR12]. The main features of the tool are easy creation, efficient editing and quantitative analysis of ADTrees [13KKMS]. Since from a formal perspective, attack trees (Section 7.4.1), protection trees (Section 7.4.3), and defense trees (Section 7.4.3) are instances of attack–defense trees, the ADTool also supports all these formalisms.

Finally, ADTrees can be seen as a natural extension of defense trees (Section 7.4.3), where defenses are only allowed as leaf nodes. The ADTree formalism is similar to the formalism of attack countermeasure trees (Section 7.4.3), however, there exist a couple of fundamental differences between the two models. On the one hand, in ADTrees defense nodes can be refined and countered, which is not possible in attack countermeasure trees. On the other hand, attack countermeasure trees distinguish between detection and mitigation events which are both modeled with defense nodes in ADTrees. Another difference is that attack countermeasure trees are well-suited to compute specific parameters, including probability, return on investment (ROI) and return on attack (ROA). ADTrees, in turn, focus on general methods for attribute computation. A different formalism, also called attack–defense trees, was used by Du et al. in [DLDZ12] to perform a game-theoretic analysis of Vehicular ad hoc network security by utilizing the ROA and ROI utility functions. Despite sharing the same name with the formalism introduced in [12KMRS], the attack–defense tree approach used in [DLDZ12] is built upon defense trees (Section 7.4.3) and does not contain the possibility to refine countermeasures. Moreover, it does not consider any formal semantics.

Countermeasure Graphs

Countermeasure graphs provide a DAG-based structure to identify and prioritize countermeasures. They were introduced by Baca and Petersen [BP10] in 2010 as an integral part of the *countermeasure method for security* which aims at simplifying countermeasure selection through cumulative voting.

To build the graphical model, actors, goals, attacks and countermeasures are identified. Goals explain why actors attack a system, attacks detail how the system could get attacked and countermeasures describe how attacks could be prevented. When the representing events are related, edges are drawn between goals and actors, actors and attacks as well as between attacks and countermeasures. More specifically, an edge is drawn between a goal and an actor if the actor pursues the goal. An edge is inserted between an actor and an attack if the actor is likely to be able to execute the attack. Finally, an edge is drawn between an attack and a countermeasure if the countermeasure is able to prevent the attack. Priorities are assigned to goals, actors, attacks and countermeasures according to the rules of hierarchical cumulative voting [BS09]. The higher the assigned priority is, the higher is the threat level of the corresponding event.

With the help of hierarchical cumulative voting [BS09] the most effective countermeasures can be deduced. Clever normalization and the fact that countermeasures that prevent several attacks contribute more to the final result than isolated countermeasures guarantee that the countermeasure with the highest computed value is most efficient and should, therefore, be implemented.

The methodology is demonstrated on an open source system, a first person shooter called Code 43 [BP10].

7.4.4 Sequential Modeling of Attacks and Defenses

Insecurity Flows

In 1997, Moskowitz and Kang described a model called *insecurity flows* to support risk assessment [MK97]. It combines graph theory and discrete probability theory, offering both graphical representation and quantification capabilities to analyze "How can an invader penetrate through security holes to various protective security domains?" This analysis aims at identifying the most vulnerable paths and the most appropriate security measures to eliminate the vulnerabilities of the system.

From a high-level perspective, insecurity flows are similar to reliability block diagrams [DP08] used in reliability engineering. The source corresponds to the starting point of the attacker, the sink corresponds to the objective of the attacker and the asset under protection. An insecurity flow diagram is a circuit connecting security measures, as serial or in parallel, from the sink to the source. Serial nodes must be passed by the attacker one after another, whereas for parallel nodes only one out of n must be passed to continue on the path to the sink. The graph is used to identify insecurity flows and quantify them using probabilistic calculations. The paper provides a sound description of the formalism and the associated quantifications.

Based on the circuit, the probability that the insecurity flow can pass through the modeled security measures of a given system or architecture can be computed. Probability computation formulas for simple serial and parallel patterns are provided, whereas reduction formulas are proposed for more elaborated circuits (decomposing them into the simple patterns). Several defensive architectures can be compared along this metric.

Intrusion DAGs

Intrusion DAGs (I-DAGs) have been introduced by Wu et al. [WFM+03] as the underlying structure for attack goals representation in the Adaptive Intrusion Tolerant System, called ADEPTS in 2003. The global goal of ADEPTS is to localize and automatically respond to detected, possibly multiple and concurrent intrusions on a distributed system.

I-DAGs are directed acyclic graphs representing intrusion goals in ADEPTS. I-DAGs are not necessarily rooted DAGs, i.e., they may have multiple roots. The nodes of an I-DAG represent goals or subgoals of an attack and can be associated with an alert from the intrusion detection framework described in [WFMB03]. A goal represented by a node can only be achieved if (some of) the goals of its children are achieved. To model the connection, I-DAGs use standard AND and OR refinement features similar to the refinements in attack trees. Each node stores two information sets: a cause service set (including all services that may/must be compromised in order to achieve the goal) and an effect service set (including all services that are taken to be compromised once the goal is achieved). The method presented in [WFM+03] allows us to automatically trigger a response of appropriate severity, based on a value which expresses the confidence that the goal corresponding to a node has been achieved. This provides dynamic aspects to the ADEPTS methodology.

Three algorithms have been developed in order to support automated responses to detected incidents. The goal of the first algorithm is to classify all nodes as candidates for responses as follows. A bottom-up procedure assigns the compromised confidence index to each node situated on the paths between the node representing a detected incident and a root node. Then, a value called threshold is defined by the user and is used by a top-down procedure to label the nodes as strong, weak or non-candidates for potential responses. The second algorithm assigns the response index to nodes. The response index is a real number used to determine the response to be taken for a given node in the I-DAG. Finally, the third algorithm is based on a so-called effectiveness index. It is responsible for dynamically deciding which responses are to be taken next. Intuitively, the effectiveness index of a node is reduced for every detected failure of a response action and increased for every successful deployment.

A lightweight distributed e-commerce system has been deployed to serve as a test bed for the ADEPTS tool. The system contained 6 servers and has 26 nodes in the corresponding I-DAG. The results of the experiments and analysis are described in [WFM+03].

In [FWM+05] and [WFM+05], the authors extend the model of intrusion DAGs to intrusion graphs (I-GRAPHs). The main difference is that, contrary to I-DAGs, I-GRAPHs may contain cycles. Nodes of an I-GRAPH do not need to be independent. All dependencies between the nodes are depicted by the edges between nodes. Additionally to AND and OR refinements, I-GRAPHs also make use of quorum edges. A value called minimum required quorum is assigned to quorum edges and represents the minimal number of children that need to be achieved in order to achieve the parent node.

Bayesian Defense Graphs

In a series of papers starting in 2008, Sommestad et al. construct a Bayesian network for security (Section 7.4.2) that includes defenses to perform enterprise architecture analysis [FSEJ08, SEJ08, SEJ09, ES09, SEN09]. Their model, explicitly called *Bayesian defense graphs* in [SEJ09], is guided by the idea of depicting what exists in a system rather than what it is used for [SEJ09]. This philosophy was adapted from [JJSU07]. Bayesian defense graphs are inspired by defense trees (Section 7.4.3) and, therefore, add countermeasures to Bayesian networks. As a result, the formalism supports a holistic system view including attack and defense components.

Bayesian defense graphs build up on extended influence diagrams (Section 7.6.4), including utility nodes, decisions nodes, chance nodes and arcs. Chance nodes and decision nodes are associated with random variables that may assume one of several predefined and mutually exclusive states. The random variables are given as conditional probability tables (or matrices). Utility nodes express the combination of states in chance nodes and decision nodes. Countermeasures, which are controllable elements from the perspective of the system owner, are represented as chance nodes with adapted conditional probability tables. Finally, causal arcs (including an AND or OR label) are drawn between the nodes indicating how the conditional probabilities are related. A strength of Bayesian defense graphs is that they allow a trade-off between collecting as much data as possible and the degree of accuracy of the collected data. Through the use of iterative refinements, it is possible to reduce the complexity of the model [SEJ09].

Like all formalisms that involve Bayesian statistics, Bayesian defense graphs use conditional probability tables to answer "How do the security mechanisms influence each other?" and "How do they contribute to enterprise-wide security?" [SEJ08]. The authors of [SEJ08] exemplify how to compute the expected loss for both the current scenario and potential future scenarios. In [FSEJ08], a suitable subset of a set of 82 security metrics known as Joint Quarterly Readiness Review (JQRR) metrics has been selected and adapted to Bayesian defense graphs. The metrics serve as "a posteriori indicators on the historical success rates of hostile attacks" or "indicate the current state of countermeasures". The formalism can handle causal and uncertainty measurements at the same time, by specifying how to combine the conditional probability tables.

With the help of a software tool for abstract models [JJSU07], Bayesian defense graphs were applied by Sommestad et al. to analyze enterprise architectures on numerous occasions. In [ES09], ongoing efforts related to Bayesian defense graphs within the EU research project VIKING [VIK11] are summarized. The methodology is expanded in three follow-up papers that illustrate security assessment based on an enterprise architecture model [SEJ08, SEJ09] and information flow during a spoofing attack on a server [FSEJ08]. In [SEN09], a real case study was performed with a power distribution operator to assess the security of wide-area networks (WANs) used to operate electrical power systems. Since the results could not be published, the methodology was demonstrated on a fictitious example assessing the security of two communication links with the help of conditional probability tables [SEN09].

A similar but less developed idea of using random variables, defenses and an inference algorithm to compute the expected cost of an attack is presented by Mirembe and Muyeba [MM08].

Security Goal Indicator Trees

Peine et al. devised *security goal indicator trees* (SGITs) in 2008, in order to support security inspections of software development and documents [PJM08].

A SGIT is a tree which combines negative and positive security features that can be checked during an inspection, in order to see if a security goal (e.g., secure password management) is met. With this objective in mind, *indicators* can be linked in the resulting tree structure by three types of relations: Conditional dependencies are represented by a special kind of edge, Boolean combinations are modeled by AND and OR gates and a *specialization* relation is represented by a UML-like inheritance symbol. Moreover, a notion of *polarity* is defined for each node, attributing a positive or negative effect of a given property on security. The definition of SGITs is semi-formal.

The formalism does not support quantitative evaluations.

SGITs are implemented in a prototype tool, which was mentioned in [PJM08]. They are used to formalize security inspection processes for a distributed repository of digital cultural data in an e-tourism application in [JEBR10]. The formalism is extended to dependability inspection in [KEE10].

Attack-Response Trees

In 2009, Zonouz et al. introduced *attack-response trees* (ARTs) as a part of a methodology called response and recovery engine (RRE), which was proposed to automate the intrusion response process. The goal of the RRE is to provide an instantaneous response to intrusions and thus eliminate the delay which occurs when the response process is performed manually. The approach is modeled as a two-player Stackelberg stochastic game between the leader (RRE) and the follower (attacker). Attack-response trees have been used in [ZKSY09], for the first time. This paper constitutes a part of the Ph.D. thesis of Zonouz [Zon11].

ARTs are an extension of attack trees (Section 7.4.1) that incorporate possible response actions against attacks. They provide a formal way to describe the system security based on possible intrusion and response scenarios for the attacker and the response engine, respectively. An important difference between attack trees and attack-response trees is that the former represent all possible ways of achieving an attack goal and the latter are built based on the attack's consequences[16]. In an attack-response tree, a violation of a security property, e.g., integrity, confidentiality or availability, is assigned to the root node (main consequence). Refining nodes represent subconsequences whose occurrence implies that the parent consequence will take place. Some consequence nodes are then tagged by response nodes that represent response actions against the consequence to which they are connected.

[16] What the authors of [ZKSY09] call *subconsequences* is, in the literature also called the causes of the main consequence.

The goal of attack-response trees is to probabilistically verify whether the security property specified by the root of an attack-response tree has been violated, given the sequence of the received alerts and the successfully taken response actions. First, a simple bottom-up procedure is applied in the case when values 0 and 1 are assigned to the leaf nodes. More precisely, when a response assigned to a node v is activated (i.e., when it is assigned 1), the values in the subtree rooted in v are reset to 0. Second, [ZKSY09] also discusses the situation when uncertainties in intrusion detections and alert notifications render the determination of Boolean values impossible. In this case, satisfaction probabilities are assigned to the nodes of attack-response trees and a game-theoretic algorithm is used to decide on the optimal response action. In [ZSR$^+$11], the RRE has been extended to incorporate both IT system-level and business-level metrics to the model. Here, the combined metrics are used to recommend optimal response actions to security attacks.

The RRE has been implemented on top of the intrusion detection system (IDS) Snort 2.7, as described in [Zon11]. A validation of the approach on a SCADA system use case [ZKSY09] and a web-based retail company example [ZSR$^+$11] has shown that this dynamic method performs better than static response mechanisms based on lookup tables. The RRE allows us to recover the system with lower costs and is more helpful than static engines when a large number of IDS alerts from different parts of the system are received.

As pointed out in [RKT10b], the approach described in this section suffers from the state space explosion problem. To overcome this problem, attack countermeasure trees (Section 7.4.3) have been introduced. Their authors propose efficient algorithms for selecting an optimal set of countermeasures.

Boolean Logic Driven Markov Process

Boolean logic driven Markov processes (BDMPs) are a general security modeling formalism, which can also complete generic risk assessment procedures. The formalism was invented by Bouissou and Bon in 2003 in the safety and reliability area [BB03] and was adapted to security modeling by Piètre-Cambacédès and Bouissou in 2010 [PCB10a, PCB10b].[17] Its goal is to find a better trade-off between readability, modeling power and quantification capabilities with respect to the existing formalisms in general and attack trees in particular.

BDMPs combine the readability of classical attack trees with the modeling power of Markov chains. They change the attack tree semantics by augmenting it with links called triggers. In a first approach, triggers allow modeling of sequences and simple dependencies by conditionally *activating* subtrees of the global structure. The root (top event) of an BDMP is the objective of the attacker. The leaves correspond to attack steps or security events. They are associated with Markov processes, dynamically selected depending on the states of some other leaves. They can be connected by a wide choice of logical gates, including AND, OR and PAND gates, commonly used in dynamic fault trees (Section 7.4.2). The overall approach allows for sequential modeling in an attack tree-like structure while enabling efficient quantifications. BDMPs for security are well-formalized [PCB10a].

[17] The original idea was introduced in an abstract by the same authors in 2009 [PCB09]

Success or realization parameters (mean time to success or to realization) are associated with the leaves, depending on the basic event modeled. Defense-centric attributes can also be added, reflecting detection and reaction capabilities (the corresponding parameters are the probability or the mean-time to detection for a given leaf and the reduction of chance of success in case of detection). BDMPs for security allow for different types of quantification. These quantifications include the computation of time-domain metrics (overall mean-time to success, probability of success in a given time, ordered list of attack sequences leading to the objectives), attack tree related metrics like costs of attacks, handling of Boolean indicators (e.g., specific requirements), and risk analysis oriented tools like sensibility graphs by attack step or event [PCDB11], etc.

The model construction and its analysis are supported by an industrial tool, called KB3 [Ele12]. In [PCDB11], implementation issues and user feedback are discussed and analyzed. BDMPs are used in [PCB10c, Joh11] to integrate safety and security analyses while [KBPC12] develops a realistic use case based on the Stuxnet attack.

In several papers [PCB10b, PCB10a, PCDB11], the authors point out the intrinsic limits of BDMPs to model cyclic behaviors and loops, as well as the difficulties in assigning relevant values for the leaves.

Security Goal Models

In 2010, *Security goal models* (SGMs) were formalized by Byers and Shahmehri in order to identify the causes of software vulnerabilities and model their dependencies [BS10]. They were introduced as a more expressive replacement for attack trees (Section 7.4.1), security goal indicator trees (Section 7.4.4), vulnerability cause graphs (Section 7.4.2) and security activity graphs (Section 7.4.3). The main goal of a SGM corresponds to a vulnerability. "Starting with the root, subgoals are incrementally identified until a complete model has been created" [SMdO$^+$12].

In SGMs, a goal can be anything that affects security or some other goal, e.g., it can be a vulnerability, a security functionality, a security-related software development activity or an attack. SGMs have two types of goal refinements: one type represents dependencies and one type modeling information flow. Dependency nodes are connected with solid edges (dependence edge) and are depicted by white nodes for contributing subgoals and by black nodes for countering subgoals. Information edges are displayed with dashed edges. The formalism consists of a syntactic domain (elements that make up the model), an abstract syntax (how elements can be combined), a visual representation (using graphical symbols) and a semantic transformation from the syntactic domain to the semantic domain. The syntactic domain consists of the root, subgoals (contributing or counteracting), dependency edges, operators AND and OR that express the connection of dependency edges, annotation connected to nodes by annotation edges, stereotype (usually an annotation about a dependency edge), ports that model information flow and information edges that connect ports. The abstract syntax is defined in a UML class diagram [SMdO$^+$12].

It is possible to evaluate whether a security goal was successfully reached or not. To do this, each cause is defined with a logical predicate (true or false). Then the predicates are composed using Boolean logic and taking the information from the information edges into account.

SGMs were used in a case study about passive testing vulnerability detection, i.e., examining the traces of a software system without the need for specific test inputs. In a four step testing procedure vulnerabilities are first modeled using SGMs. In the next step, causes are formally defined before SGMs are converted into vulnerability detection conditions (VDC). In the final step vulnerabilities are checked based on the VDCs. In [SMdO^{+}12] this procedure is performed on the xine media player [xp12] where an older version contained the CVE-2009-1274 vulnerability. The case study is executed with the help of *TestInv-Code*, a program developed by Montimage that can handle VDCs.

In [BS10], the authors explicitly state that they have defined transformations to and from attack trees VCGs, SAGs and SGITs so that SGMs can be used with possibly familiar notation. (The transformations, however, were omitted due to space restrictions.)

Unified Parameterizable Attack Trees

In 2011, Wang et al. introduced *unified parameterizable attack trees*[18] [WWPP11]. As the name suggests, the formalism was created as a foundation to unify numerous existing extensions of attack trees (Section 7.4.1). The formalism generalizes the notions of connector types, edge augmentations and (node) attributes. With the help of these generalizations it is possible to describe other extensions of attack trees as structural extensions, computational extensions or hybrid extensions.

Unified parameterizable attack trees are defined as a 5-tuple, consisting of a set of nodes, a set of edges, a set of allowed connectors (O-AND i.e., a time or priority based AND, U-AND i.e., an AND with a threshold condition and OR), a set of attributes and a set of edge augmentation structures that allows us to specify edge labels. Using this definition, the authors of [WWPP11] identify defense trees (Section 7.4.3), attack countermeasure trees (Section 7.4.3), attack-response trees (Section 7.4.4), attack–defense trees (Section 7.4.3), protection trees (Section 7.4.3), OWA trees (Section 7.4.1), and augmented attack trees (Section 7.4.1) as structure-based extensions of attack tree that are covered by unified parameterizable attack trees. They classify multi-parameter attack trees (Section 7.4.1 and 7.4.2) as a computational extension of attack trees.

The formalism classifies attributes into the categories of *attack accomplishment attributes*, *attack evaluation attributes* and *victim system attributes*, but does not specify how to perform quantitative evaluations.

Unified parameterizable attack trees are primarily built upon augmented attack trees (Section 7.4.1). In fact, the authors indicate how to instantiate the node attributes, the edge augmentation and the connector type to obtain an augmented attack tree.

[18] Wang et al. use British English, thus originally, the name of their formalism is *unified parametrizable attack trees*.

7.5 Summary of the Surveyed Formalisms

In this section, we provide a consolidated view of all formalisms introduced in Section 7.4. Tables 7.2–7.4 characterize the described methodologies (ordered alphabetically) according to the 13 aspects presented in Table 7.1. The aspects are grouped into formalism features and capabilities (Table 7.2), formalism characteristics (Table 7.3) and formalism maturity and usability factors (Table 7.4). This tabular view allows the reader to compare the features of the formalisms more easily, it stresses their similarities and differences. Furthermore, the tables support a user in selecting the most appropriate formalism(s) with respect to specific modeling needs and requirements. We illustrate such a support on two exemplary situations.

Example 1 Let us assume that during a risk assessment, analysts want to investigate and compare the efficiency of different defensive measures and controls with respect to several attack scenarios. Thereto, they need quantitative elements to support the analysis technique they will choose. Furthermore, a software tool and pre-existing use cases are required to facilitate their work. Using the corresponding columns from Tables 7.2–7.4 (i.e., attack or defensive, quantification, tool availability and case study) and choosing the formalisms characterized by appropriate values (respectively: *both*, *versatile or specific*, *industrial or prototype* and *real(istic)*), would help the analysts to pre-select attack countermeasure trees, attack–defense trees, BDMPs, intrusion DAGs, and security activity graphs as potential modeling and analysis techniques. The most suitable methodology could then be selected based on more detailed information provided in Section 7.4. For instance, let us assume that the analysis requires the use of measures for probability of success, the attacker's costs and the attacker's skills. Checking descriptions of the pre-selected formalisms, given in Section 7.4, would convince the analysts that security activity graphs and intrusion DAGs would not allow them to compute the desired quantitative elements. Therefore, it would reduce the choice to attack countermeasure trees, attack–defense trees and BDMPs. A more thorough investigation of the computational procedures and algorithms described in the referred papers would help the analysts to make the final decision on the formalism that best fits their needs.

Example 2 Now, let us assume that a team of penetration testers wants to illustrate which attack paths they have used to compromise different systems. Initially, this does not significantly reduce the choice of possible formalisms since they could use all attack-oriented and all attack and defense oriented approaches. However, to keep the model as simple as possible, they start the selection process by looking at the attack-oriented methodologies only. Let us assume further that the penetration testers also do not need to represent sequences of actions. With a similar reasoning as before, they first investigate a possibility of using a static modeling technique. The team does not foresee to perform any quantitative analysis. An important requirement, however, is to employ a methodology which is already broadly used, with at least rudimentary documented use cases on which they could rely to build their own models. Using relevant columns from Tables 7.2–7.4 (i.e., attack or defense, sequential or static, paper count, use cases and quantification) and selecting formalisms characterized with appropriate values (respectively: *at-*

tack or both, *sequential or static*, > 4, *real(istic) or toy* and *versatile, specific or no*), the team obtains a large number of applicable formalisms. In order to keep the formalism as simple as possible, the analysts decide to narrow the set of values they are interested in to: *attack*, *static*, > 4, *real(istic) or toy* and *no*. This strategy yields the following most suitable formalisms: attack trees, augmented attack trees and parallel model for multi-parameter attack trees. The team would then be able to make a final choice of the methodology based on complementary investigations starting from the information and references provided by the corresponding textual descriptions from Section 7.4.

7.6 Alternative Graphical Security Methodologies

We close this survey with a short overview of alternative methodologies for security modeling and analysis. The formalisms described here are outside the scope of this thesis because they were not originally introduced for the purposes of attack and defense modeling or they are not based on the DAG structure. However, for the sake of completeness, we find it important to briefly present those approaches as well. The objective of this section is to give pointers to other existing methodological tools for security assessment, rather than to perform a thorough overview of all related formalisms. To this end, the description of the formalisms given here is less complete and structured than the information provided in Section 7.4.

7.6.1 Petri Nets for Security

In the mid 1990s, models based on Petri nets have been applied for security analysis [KS94, Dac94]. In 1994, Kumar and Spafford [KS94] adopted colored Petri nets for security modeling. They illustrate how to model reference scenarios for an intrusion detection device. Also in 1994, Dacier [Dac94] used Petri nets in his Ph.D. thesis as part of a larger quantification model that describes the progress of an attacker taking over a system. A useful property of Petri nets is their great modeling capability and in particular their ability to take into account the sequential aspect of attacks, the modeling of concurrent action and different forms of dependency. Petri nets are widely used and have various specific extensions. To corroborate this statement, we list a few existing ones. Kumar and Spafford's work relies on *colored Petri nets* [KS94], Dacier's on *stochastic Petri nets* [Dac94], McDermott's on *disjunctive Petri nets* [McD00], Horvath and Dörges's on *reference nets* [HD08], Dalton II et al.'s on *generalized stochastic Petri nets* [DMCR06], Pudar et al.'s on *deterministic time transition Petri nets* [PML10] and Xu and Nygard's on *aspect-oriented Petri nets* [XN06]. Several articles on Petri nets merge the formalism with other approaches. Horvath and Dörges combine Petri nets with the concept of *security patterns* [HD08] while Dalton II et al. [DMCR06], and more thoroughly Pudar et al. [PML10], combine Petri nets and attack trees.

In 1994, Dacier embedded Petri nets into a higher level formalism called *privilege graphs*. They model an attacker's progress in obtaining access rights for a desired target [Dac94, DD94]. In a privilege graph, a node represents a set of privileges and an edge a method for transferring these privileges to the attacker. This corresponds to the exploitation of a vulnerability. The model includes an attacker's *memory*

Name of formalism	Attack or defense	Sequential or static	Quantification	Main Purpose	Extension(s)
Anti-models (Section 7.4.3)	Both	Static	No	Req. eng.	New formalism
Attack countermeasure trees (Section 7.4.3)	Both	Static	Specific	Sec. mod.	Structural, Computational
Attack–defense trees (Section 7.4.3)	Both	Static	Versatile	Sec. mod.	Structural, Computational
Attack-response trees (Section 7.4.4)	Both	Sequential	Specific	Int. det.	Structural, Quantitative
Attack trees (Section 7.4.1)	Attack	Static	Versatile	Sec. mod.	New formalism
Augmented attack trees (Section 7.4.1)	Attack	Static	Specific	Sec. mod.	Structural, Computational
Augmented vulnerability trees (Section 7.4.1)	Attack	Static	Specific	Risk	Quantification
Bayesian attack graphs (Section 7.4.2)	Attack	Sequential	Specific	Risk	Structural, Computational
Bayesian defense graphs (Section 7.4.4)	Both	Sequential	Specific	Risk	Structural, Computational
Bayesian networks for security (Section 7.4.2)	Attack	Sequential	Specific	Risk	Structural, Computational
BDMPs (Section 7.4.4)	Both	Sequential	Versatile	Sec. mod.	Order, Time
Compromise graphs (Section 7.4.2)	Attack	Sequential	Specific	Risk	New formalism
Countermeasure graphs (Section 7.4.3)	Both	Static	Specific	Sec. mod.	Structural, Computational
Cryptographic DAGs (Section 7.4.2)	Attack	Sequential	No	Risk	New formalism
Defense trees (Section 7.4.3)	Both	Static	Specific	Sec. mod.	Structural, Computational
Dynamic fault trees for security (Section 7.4.2)	Attack	Sequential	No	Sec. mod.	Order, Time
Enhanced attack trees (Section 7.4.2)	Attack	Sequential	Specific	Int. det.	Order, Time
Extended fault trees (Section 7.4.1)	Attack	Static	Specific	Unification	Structural
Fault trees for security (Section 7.4.2)	Attack	Sequential	No	Sec. mod.	Order
Improved attack trees (Section 7.4.2)	Attack	Sequential	Specific	Risk	Structural, Computational
Insecurity flows (Section 7.4.4)	Both	Sequential	Specific	Risk	New formalism
Intrusion DAGs (Section 7.4.4)	Both	Sequential	Specific	Int. det.	Structural, Computational
OWA trees (Section 7.4.1)	Attack	Static	Specific	Quant.	Structural, Computational
Parallel model for multi-parameter attack trees (Section 7.4.1)	Attack	Static	Specific	Quant.	Quantitative, Computational
Protection trees (Section 7.4.3)	Defense	Static	Specific	Sec. mod.	New formalism
Security activity graphs (Section 7.4.3)	Both	Static	Specific	Soft. dev.	New formalism
Security goal indicator trees (Section 7.4.4)	Defense	Sequential	No	Soft. dev.	New formalism
Security goal models (Section 7.4.4)	Both	Sequential	Specific	Unification	Structural, Computational
Serial model for multi-parameter attack trees (Section 7.4.2)	Attack	Sequential	Specific	Quant.	Computational, Order
Unified parameterizable attack trees (Section 7.4.4)	Both	Sequential	Versatile	Unification	Structural
Vulnerability cause graphs (Section 7.4.2)	Attack	Sequential	Specific	Soft. dev.	Structural, Order

Table 7.2: Aspects relating to the formalism's modeling capabilities.

Name of formalism	**Structure**	**Connectors**	**Formalization**
Anti-models (Section 7.4.3)	Tree	AND, OR	Semi-formal
Attack countermeasure trees (Section 7.4.3)	Tree	AND, OR, k-out-of-n, counter leaves	Formal
Attack–defense trees (Section 7.4.3)	Tree	AND, OR, countermeasures	Formal
Attack-response trees (Section 7.4.4)	Tree	AND, OR, responses	Formal
Attack trees (Section 7.4.1)	Tree	AND, OR	Formal
Augmented attack trees (Section 7.4.1)	Tree	AND, OR	Formal
Augmented vulnerability trees (Section 7.4.1)	Tree	AND, OR	Informal
Bayesian attack graphs (Section 7.4.2)	DAG	AND, OR, conditional probabilities	Formal
Bayesian defense graphs (Section 7.4.4)	DAG	AND, OR, conditional probabilities	Formal
Bayesian networks for security (Section 7.4.2)	DAG	AND, OR, conditional probabilities	Formal
BDMPs (Section 7.4.4)	DAG	AND, OR, PAND, approx. OR, triggers	Formal
Compromise graphs (Section 7.4.2)	Unspecified	None	Formal
Countermeasure graphs (Section 7.4.3)	DAG	Countermeasures	Informal
Cryptographic DAGs (Section 7.4.2)	DAG	Dependence edges	Informal
Defense trees (Section 7.4.3)	Tree	AND, OR, counter leaves	Semi-formal
Dynamic fault trees for security (Section 7.4.2)	Tree	AND, OR, PAND, SEQ, FDEP, CSP	Informal
Enhanced attack trees (Section 7.4.2)	Tree	AND, OR, ordered AND	Formal
Extended fault trees (Section 7.4.1)	Tree	AND, OR, merge gates	Formal
Fault trees for security (Section 7.4.2)	Tree	AND, OR, PAND, XOR, inhibit	Informal
Improved attack trees (Section 7.4.2)	Tree	AND, OR, sequential AND	Informal
Insecurity flows (Section 7.4.4)	Unspecified	None	Formal
Intrusion DAGs (Section 7.4.4)	DAG	AND, OR	Semi-formal
OWA trees (Section 7.4.1)	Tree	OWA operators	Formal
Parallel model for multi-parameter attack trees (Section 7.4.1)	Tree	AND, OR	Formal
Protection trees (Section 7.4.3)	Tree	AND, OR	Informal
Security activity graphs (Section 7.4.3)	DAG	AND, OR, split gate	Semi-formal
Security goal indicator trees (Section 7.4.4)	Tree	AND, OR, dependence edge, specialization edge	Semi-formal
Security goal models (Section 7.4.4)	DAG	AND, OR, dependence edge, information edge	Formal
Serial model for multi-parameter attack trees (Section 7.4.2)	Tree	AND, OR, ordered leaves	Formal
Unified parameterizable attack trees (Section 7.4.4)	Tree	AND, OR, PAND, time-based AND, threshold AND	Formal
Vulnerability cause graphs (Section 7.4.2)	DAG	AND, OR, sequential AND	Informal

Table 7.3: Aspects relating to the formalism's characteristics.

Name of formalism	Tool availability	Case study	External use	Paper count	Year
Anti-models (Section 7.4.3)	No	No	No	3	2006
Attack countermeasure trees (Section 7.4.3)	Prototype	Real(istic)	No	4	2010
Attack–defense trees (Section 7.4.3)	Prototype	Real(istic)	Collaboration	6	2010
Attack-response trees (Section 7.4.4)	Prototype	Toy case study	No	3	2009
Attack trees (Section 7.4.1)	Commercial	Real(istic)	Independent	> 100	1991
Augmented attack trees (Section 7.4.1)	No	Real(istic)	Independent	6	2005
Augmented vulnerability trees (Section 7.4.1)	No	Real(istic)	Independent	3	2003
Bayesian attack graphs (Section 7.4.2)	Commercial	Toy case study	Independent	10	2005
Bayesian defense graphs (Section 7.4.4)	Prototype	Real(istic)	No	5	2008
Bayesian networks for security (Section 7.4.2)	Commercial	Real(istic)	Independent	14	2004
BDMPs (Section 7.4.4)	Commercial	Real(istic)	Independent	5	2010
Compromise graphs (Section 7.4.2)	No	Real(istic)	Collaboration	3	2006
Countermeasure graphs (Section 7.4.3)	No	Toy case study	No	1	2010
Cryptographic DAGs (Section 7.4.2)	No	No	No	1	1996
Defense trees (Section 7.4.3)	No	No	No	3	2006
Dynamic fault trees for security (Section 7.4.2)	No	No	No	1	2009
Enhanced attack trees (Section 7.4.2)	No	No	No	1	2007
Extended fault trees (Section 7.4.1)	No	No	No	1	2007
Fault trees for security (Section 7.4.2)	Commercial	Real(istic)	Independent	3	2003
Improved attack trees (Section 7.4.2)	No	No	No	1	2011
Insecurity flows (Section 7.4.4)	No	No	No	1	1997
Intrusion DAGs (Section 7.4.4)	Prototype	Real(istic)	No	2	2003
OWA trees (Section 7.4.1)	No	No	No	2	2005
Parallel model for multi-parameter attack trees (Section 7.4.1)	Prototype	Real(istic)	Collaboration	5	2006
Protection trees (Section 7.4.3)	No	Toy case study	No	4	2006
Security activity graphs (Section 7.4.3)	Prototype	Real(istic)	No	2	2006
Security goal indicator trees (Section 7.4.4)	Prototype	Real(istic)	No	3	2008
Security goal models (Section 7.4.4)	No	Real(istic)	No	2	2010
Serial model for multi-parameter attack trees (Section 7.4.2)	Prototype	No	No	3	2010
Unified parameterizable attack trees (Section 7.4.4)	No	No	No	1	2011
Vulnerability cause graphs (Section 7.4.2)	Commercial	Real(istic)	Independent	4	2006

Table 7.4: Aspects relating to the formalism's maturity and usability.

which forbids him to go through privilege states that he has already acquired. In addition, an attacker's *good sense* is modeled which prevents him from regressing. In [DDK96], Dacier et al. proposed to transform a privilege graph into a Markov chain corresponding to all possible successful attack scenarios. The method has been applied to help system administrators to monitor the security of their systems.

In [ZF11], Zakrzewska and Ferragut presented a model extending Petri nets in order to model real-time cyber conflicts. This formalism is able to represent situational awareness, concurrent actions, incomplete information and objective functions. Since it makes use of stochastic transitions, it is well-suited to reason about stochastic non-controlled events. The formalism is used to run simulations of cyber attacks in order to experimentally analyze cyber conflicts. The authors also performed a comparison of their *extended Petri nets* model with other security modeling techniques. In particular, they showed that extended Petri nets are more readable and more expressive than attack graphs, especially with respect to the completeness of the models.

7.6.2 Attack Graphs

The term *attack graph* was first introduced by Phillips and Swiler [PS98, SPEC01] in 1998 and has extensively been used ever since. The nodes of an attack graph represent possible states of a system during the attack. The edges correspond to changes of states due to an attacker's actions. An attack graph is generated automatically based on three types of inputs: attack templates (generic representations of attacks including required conditions), a detailed description of the system to be attacked (topology, configurations of components, etc.) and the attacker's profile (his capability, his tools, etc.). Quantifications, such as average probabilities or time to success, can be deduced by assigning weights to the edges and by finding shortest paths in the graph.

Starting in 2002, Sheyner et al. [SHJ$^+$02, She04] made extensive contributions to popularize attack graphs by associating them with model checking techniques. To limit the risk of combinatorial explosion, a large number of methods were developed. Ammann et al. [AWK02] restricted the graphs by exploiting a monotony property, thereby eliminating backtracking in terms of privilege escalation. Noel and Jajodia and others [NJOJ03, JNO05] took configuration aspects into account. A complete state of the art concerning the contributions to the field between 2002 and 2005 can be found in [LI05]. In 2006, Wang et al. introduced a relational model for attack graphs [WYSJ06]. The approach facilitates interactive analysis of the models and improves its performance. Ou et al. [OBM06] optimized the generation and representation of attack graphs by transforming them into *logical attack graphs* of polynomial size with respect to the number of components of the computer network analyzed. During the same year, Ingols et al. [ILP06] proposed *multiple-prerequisite graphs*, which also severely reduce the complexity of the graphs. In [MBZ$^+$06], Mehta et al. proposed an algorithm for the classification of states in order to identify the most relevant parts of an attack graph. In 2008, Malhotra et al. [MBG08] did the same based on the notion of an *attack surface* described in [Man08]. The vast majority of the authors mentioned have also worked on visualization aspects [NJ04, NJKJ05, WLI07, HVOM08]. Kotenko and Stepashkin [KS06] described a complete software platform for implementing concepts and metrics of attack graphs. On a theoretical level, Braynov and Jadliwala [BJ03] extended the model to several attackers.

Starting in 2003, the problem of quantitative assessment of the security of networked systems using attack graphs has been extensively studied [NJOJ03, WNJ06, WSJ07a, WSJ07b, WIL+08]. The work presented in [NJOJ03] and [WNJ06] focuses on minimal cost of removing vulnerabilities in hardening a network. In [WSJ07a], the authors introduced a metric, called attack resistance, which is used to compare the security of different network configurations. The approach was then extended in [WSJ07b] into a general abstract framework for measuring various aspects of network security. In [WIL+08], Wang et al. introduced a metric incorporating probabilities of the existence of the vulnerabilities considered in the graph.

In his master's thesis, Louthan IV [Lou11] proposed to extend the attack graph modeling framework to permit modeling of continuous, in addition to discrete, system elements and their interactions. In [WIL+08], Wang et al. addressed the problem of likelihood quantification of potential multistep attacks on networked environments that combine multiple vulnerabilities. They developed an attack graph-based probabilistic metric for network security and proposed heuristics for efficient computation. In [NJWS10], Noel et al. used attack graphs to understand how different vulnerabilities can be combined to form an attack on a network. They simulated incremental network penetration and assessed the overall security of a network system by propagating attack likelihoods. The method allows us to give scores to risk mitigation options in terms of maximizing security and minimizing cost. It can be used to study cost/benefit trade-offs for analyzing return on security investment.

Dawkins and Hale [DH04] developed a concept similar to attack graphs called *attack chains*. The model is based on a deductive tree structure approach but also allows for inductive reasoning using *goal-inducing attack chains*, to extract scenarios leading to a given aim. These models are also capable of generating attack trees, which may be quantified by conventional methods. Aspects concerning software implementation are described in [CTDH04].

7.6.3 Approaches Derived from UML Diagrams

We start this section with a short description of two formalisms derived from UML diagrams, namely the *abuse cases* of McDermott and Fox [MF99] and the *misuse cases* of Sindre and Opdahl [SO00, SO01, SOB02, Ale03, SL05] which were later extended by Røstad in [Rø06]. These techniques are not specifically intended to model attacks but rather to capture threats and abusive behavior which have to be taken into account when eliciting security requirements (for misuse cases) as well as for design and testing (for abuse cases). The flexibility of misuse and abuse cases allows for expressive graphical modeling of attack scenarios without mathematical formalization that supports quantification.

In [Fir03], Firesmith argues that misuse and abuse cases are "highly effective ways of analyzing security threats but are inappropriate for the analysis and specification of security requirements". The reasoning is that misuse cases focus on how misusers can successfully attack the system. Thus they often model specific architectural mechanisms and solutions, e.g., the use of passwords, rather than actual security requirements, e.g., authentication mechanisms. To specify security requirements, he suggested the use of *security use cases*. Security use cases focus on how an application achieves its goals. According to Firesmith, they provide "a highly-reusable way of organizing, analyzing and specifying security requirements" [Fir03].

Diallo et al. presented a comparative evaluation of the common criteria [ISO12], misuse cases and attack trees [DRMS+06]. Opdahl and Sindre [OS09] compared usability aspects and modeling features of misuse cases and attack trees. UML-based approaches can be combined with other types of models. The combination of misuse cases and attack trees appears not only to be simple but also useful and relevant [TJR10, MTJ10]. In [KSO10b], Kárpáti et al. adapted use case maps to security as *misuse case maps*. Katta et al. [KKO+10] combined *UML sequence diagrams* with misuse cases in a new formalism called *misuse sequence diagrams*. A misuse sequence diagram represents a sequence of attacker interactions with system components and depicts how the components were misused over time by exploiting their vulnerabilities. The authors of [KKO+10] performed usability and performance comparison of misuse sequence diagrams and misuse case maps. In [KSO10a], Kárpáti et al. integrated five different representation techniques in a method called *hacker attack representation method* (HARM). The methodologies used in HARM are: attack sequence descriptions (summarizing attacks in natural language), misuse case maps (depicting the system architecture targeted by the attack and visualizing the traces of the exploits), misuse case diagrams (showing threats in relation to the wanted functionality) attack trees (representing the hierarchical relation between attacks) and attack patterns (describing an attack in detail by adding information about context and solutions). Combining such diverse representation techniques has two goals. First, it provides "an integrated view of security attacks and system architecture". Second, the HARM method is especially well-suited when different stakeholders, including non-technical people preferring informal representations, are involved in modeling the security scenario.

In [Sin07], Sindre adapted UML activity diagrams to security. The resulting *mal-activity diagrams* constitute an alternative to misuse cases when the user considers the latter to be unsuitable. This is for instance the case in situations where a large numbers of interactions need to be specified within or outside a system. Case studies mainly concern social engineering attacks [KSM12].

7.6.4 Isolated Models

In this section we gather a number of isolated models. Most of the graphs that the models use, allow cycles and, therefore, are outside of the main scope of this chapter. However, we mention them because they build upon one of the formalisms described in Section 7.4.

The *stratified node topology* was proposed by Daley et al. [DLD02] as an extension of attack trees, in 2002. The formalism consists of a directed graph which is aimed at providing a context-sensitive attack modeling framework. It supports incident correlation, analysis and prediction and extends attack trees by separating the nodes into three distinct classes based on their functionality: event-level nodes, state-level nodes and top-level nodes. The directed edges between the nodes are classified into implicit and explicit links. Implicit links allow individual nodes to imply other nodes in the tree; explicit links are created when an attack provides a capability to execute additional nodes, but does not actually invoke a new instance of a node. As in attack trees, the set of linked nodes can be connected disjunctively as well as conjunctively. In comparison with attack trees, the authors drop the requirement of a designated root node, along with the requirement that the graphs have to be acyclic. Due to the functional distinction of the nodes, the stratified node topology can keep the vertical ordering, even if the modeled scenario is cyclic.

In 2010, Abdulla et al. [ACK10] described a model called *attack jungles*. When trying to use attack trees as formalized by Mauw and Oostdijk in [MO05] to illustrate the security of a GSM radio network, the authors of [ACK10] encountered modeling problems related to the presence of cycles as well as analysis problems related to reusability of nodes in real life scenarios. This led them to propose attack jungles, which extend attack trees with multiple roots, reusable nodes and cycles that allow for modeling of attacks which depend on each other. Attack jungles are formalized as multigraphs and their formal semantics extend the semantics based on multisets proposed in [MO05]. In order to find possible ways of attacking a system, a backwards reachability algorithm for the analysis of attack jungles was described. Moreover, the notion of an attribute domain for quantitative analysis, as proposed for attack trees in [MO05], is extended to fit the new structure of attack jungles. By dividing attack components (nodes) into reusable and not reusable ones, it is possible to reason about and analyze realistic scenarios. For instance, in attack jungles it is possible to indicate whether or not a component can be reused without inducing extra costs.

Extended influence diagrams [JLNS07] form another related formalism which is not based on a DAG structure. Extended influence diagrams are built upon influence diagrams, introduced by Matheson and Howard in the 1960s [MH68], which, in turn, are an extension of Bayesian networks. Influence diagrams are used to provide a high-level visualization of decision problems under uncertainty [EBv+10]. Extended influence diagrams allow us to model the relationships between decisions, events and outcomes of an enterprise architecture. They employ the following three types of nodes: ellipses which represent events (also known as chance nodes), rectangles which depict decision nodes and diamonds which represent utility nodes (or outcomes). In addition the formalism allows us to specify how a node is defined, how well it can be controlled and how the nodes relate to each other. The latter is achieved using different types of edges. Moreover, transformation rules between graphs govern switching between different levels of abstraction of a scenario (expanding and collapsing). The rules also ensure that graphs do not contradict each other. In [LJN07], the authors show how to elicit knowledge from scientific texts, generating extended influence diagrams and in [ES09] the authors outline how extended influence diagrams can be used for cyber security management.

8

Conclusion and Future Work

In this chapter, we summarize the thesis and relate it to the field of graphical security modeling. Within this bigger picture, we then suggest future work in the field in general and on the ADTree formalism in particular.

8.1 Conclusion

In this thesis we have developed the concept of attack–defense trees (ADTrees), a new graphical security method that combines visual and formal aspects into one methodology. The structure and syntax of ADTrees are intentionally kept simple. ADTrees are an extension of attack trees which allows us to depict alternation between attacks and defenses at any level in the tree.

ADTrees are equipped with three different syntaxes, allowing the user to select his preferred representation for every application. The tree representation is visually appealing. It allows us to quickly capture a rough qualitative and quantitative estimate of the modeled scenario. Subtrees that mainly contain red attack nodes, hint at possibly vulnerable areas (non-existent defenses), whereas subtrees that are dominated by green defense nodes indicate a potential abundance of available protective measures (redundant defenses). The alternation of attack and defense nodes can also be used to capture which defenses were put in place because of which specific attacks. In other words, the illustration provides details about the evolution of attack scenarios and tells the user when certain attacks or defenses have become obsolete. The algebraic ADTerm representation is concise, allows for convenient formal treatment and is especially well-suited to link the methodology to other scientific areas, in particular those based on formal definitions or algebraic structures. Moreover, sound formalization in the form of ADTerms is necessary for a reliable algorithmic treatment of attack and defense scenarios. Finally, the textual representation combines fast input with a well-arranged output. Hence, the design of the ADTree methodology provides excellent presentation methods that are at the same time especially well-suited for computation of security relevant values.

The formalism of ADTrees is also equipped with several unambiguous semantics. Semantics are defined as equivalence relations and, therefore, allow us to say which trees actually represent the same scenario. This constitutes a first step towards comparing the expressiveness and the completeness of one ADTree with another. Among graphical security formalisms, the ADTree methodology is the only formalism that explicitly allows the selection of a semantics. The two most common semantics in the literature are the propositional and the multiset semantics. Unfortunately they are often only used implicitly and, therefore, mistakenly inter-changed.

ADTrees also support quantitative analysis of security scenarios with the help of attributes. A simple bottom-up algorithm has been extended from the attack tree approach and formalized with the help of attribute domains. Security relevant parameters that can be modeled with an attribute domain include **costs**, **time**, **skill level**, **impact** and **risk**. With the help of attribute domains, these parameters are specified more precisely to, for example, express the minimal cost of an attacker who cannot reuse an attack to overcome more than one defense. We have classified precise questions that allow us to unambiguously determine an attribute domain. A great advantage of the ADTree formalism is its ability to cope with bivariate questions. This class of questions makes use of knowledge about the attacker and the defender. A compatibility criterion specifies which attributes can be used in combination with which semantics. If the criterion is violated, attribute evaluation on equivalent trees may no longer be the same.

Throughout the development of the ADTree methodology, numerous case studies helped to polish the formalization and guaranteed its continued applicability. They resulted in detailed guidelines for the realization of use cases based on the methodology. The performance of scenario evaluations is enhanced by and supported with a software tool, the ADTool. It is especially useful in the analysis of large-scale models since it supports features to enable focusing on sections and details of a scenario.

With the help of valuations, we have shown that ADTrees in the De Morgan semantics enrich the modeling power of attack trees without increasing their computational complexity. We have also shown how to combine ADTrees with Bayesian networks to model dependent events as well as demonstrated an explicit link to game theory. This makes the ADTree methodology not only a useful tool in practice, but its sound formalization lays a foundation for further research.

As is the case for any method performing quantitative analysis, our framework may suffer from the problem of identifying input values for the algorithms. We have provided several suggestions on how to circumvent this difficulty. We can construct the input from historical data, use expert estimations, use categories instead of precise values, use ranges or use sets of possible values if we want to exclude certain values.

With respect to Bayesian ADTrees, we even need to provide dependency relations between nodes and appropriate conditional probability values. In this case, we may directly use knowledge engineering techniques existing for Bayesian networks in order to collect and process data and structural information. Such approaches combine expert knowledge with machine learning techniques. They have already been successfully applied in the security context. In [SDHH98], for instance, the authors make use of probabilistic learning methods and introduce Bayesian Spam filters which learn conditional probability distributions with user interaction. Furthermore, there exist techniques, e.g., noisy OR, that simplify the construction of conditional distributions with many parent nodes [HS13]. Setting up conditional probability tables requires, in the worst case, initialization of exponentially many values. However, it is often possible to reason with parameterized families of distributions, where it is sufficient to only provide the respective parameters of the distribution. This, in turn, is no more complicated than estimating regular, non-probabilistic parameters. Finally, our algorithmic framework is also suitable to work with imprecise probabilities [Hal03], which also reduces the size of the necessary input.

The presentation of the related work provides a methodical overview over DAG-based techniques for modeling attack and defense scenarios and a short summary on alternative graphical security modeling techniques. Some of the described methodologies have extensively been studied and are widely used to support security and risk assessment processes. Others emerged from specific, practical developments and have remained isolated methods. This overview provides a systematic description of the existing formalisms, gives pointers to related papers, tools and projects and proposes a general classification of the presented approaches.

8.2 Future Work

Two general trends can be observed in the field of graphical security modeling: *unification* and *specification*. The objective of the methodologies developed within the first trend is to unify existing approaches and propose general solutions that can be used for the analysis of a broad spectrum of security scenarios. The corresponding formalisms are well-suited for reasoning about situations involving diversified aspects, such as digital, physical and social components, simultaneously. Such models usually have sound formal foundations and are extensively studied from a theoretical point of view. They are augmented with formal semantics and a general mathematical framework for quantitative analysis. The ADTree methodology falls into the unification trend. Other examples of such models are unified parameterizable attack trees, multi-parameter attack trees, OWA trees, Bayesian attack graphs and Bayesian defense graphs.

The second observed trend, i.e., the specification trend, aims at developing methodologies for addressing domain specific security problems. Studied domains include intrusion detection (e.g., attack-response trees, intrusion DAGs), secure software development (e.g., security activity graphs, security goal indicator trees) and security requirements engineering (e.g., anti-models). Formalisms developed within this trend are often based on empirical studies and practical needs. They concentrate on domain specific metrics, such as the *response index*, which is used for the analysis of intrusion DAGs. These approaches often remain isolated and seldom relate to or build upon other existing approaches.

The multitude of methodologies shows that graphical security modeling is a young but rapidly growing area. Thus, further development is necessary and new directions need to be explored before security assessment can fully benefit from graphical models. Naturally, there is an abundance of research directions from which the area would benefit.

First and foremost a more formal classification of the existing formalisms would be beneficial to the entire field. A general classification should be a community wide effort. It will, for example, support the design of a meta-language or the definition of a compatibility notion between different models. Moreover, this classification will help in defining the most general *unification* of related approaches. For attack trees, a first unification has been published by Wang et al. [WWPP11]. Contrarily, little research has been done in formalizing which other formalisms can be expressed in terms of ADTrees. While attack trees, defense trees and protection trees can straightforwardly be modeled within the ADTree framework, this connection might not be as obvious for attack countermeasure trees, attack-response graphs or any other attack tree-like methodology.

A unified model is also useful in another research direction which has possibly not yet received enough attention. The problem of how to build graphical models from pre-existing attack templates and patterns remains unsolved. Addressing this problem would make automatic model creation possible and would replace the tedious, error-prone, manual construction process. It would, therefore, relieve the industrial sector from manual model construction when building large-scale practical models.

The idea of reusing attack patterns is not new. It has already been mentioned in 2001 by Moore et al. [MEL01]. An excellent initiative was taken by the FP7 project SHIELDS [SHI10a], in which the Security Vulnerability Repository Service (SVRS) has been developed. The SVRS is an online library of different security models including attack trees [SHI10d]. Using security patterns makes threat analysis more efficient and accurate. Generating a model from existing libraries constitutes a good starting point for further model refinement and analysis. A natural follow-up step would be to propose methods for automatic or semi-automatic construction of complex, specific models from general attack or vulnerability patterns. Composition of models is the primary challenge that any security methodology has to overcome. A fast algorithm that can determine where and how two arbitrary ADTrees differ would be an excellent start when attempting to merge ADTrees.

Since unification of models is not always possible, another research topic would be the *combination* of graphical security models. We have shown that the ADTree methodology can be combined with Bayesian networks. It would be challenging to attempt to combine other methodologies. This is especially challenging when also considering non-DAG-based methodologies like STRIDE [MS05] and SQUARE [HLOS06] that were outside the scope of this thesis.

Specifically for the ADTree approach, we envision the following improvements. While the tree-based structure has the advantage of being simple and appealing, for certain situations a model based on directed acyclic graphs may be superior. Naturally, as a consequence the treatment of repetitive nodes will have to be reconsidered. Another suitable *extension* that may help improve the expressive power of ADTrees would be to introduce dynamical modeling. Similar to dynamical Bayesian networks proposed by An et al. [AJC06], dynamical ADTrees could consist of a finite number of ADTrees each representing a certain time instance or interval.

Identification of new, meaningful structures for *semantics and attributes* would increase the versatility of the ADTree approach. There exists a need for ordered or time-respecting semantics that has not yet been fulfilled. The search for new semantics and attribute domains could start from constraint semirings with associated t-norms. One could also investigate whether a semantics or an attribute domain with non-commutative operators can be defined.

The ADTree methodology could certainly benefit from more research on *algorithms* for quantitative evaluation. The provided bottom-up procedure is not the only suitable algorithm to evaluate quantitative scenarios. Evaluating a scenario top down, might uncover model inconsistencies. Another line of research could continue to use the bottom-up algorithm. It would be useful to describe a complete set of conditions that is necessary to increase the speed of the evaluation, or to, for example, construct a sublinear algorithm. To this end, it might be possible to employ supermodular functions or generalize reduction properties from the minimax algorithm. A challenging algorithmic task would be to advance equivalence testing of ADTrees in different semantics. When equivalence testing can be performed sufficiently fast, it might be possible to construct a partial (or total) order on ADTrees.

Finally, the advice to construct node labels that contain a verb and a noun has empirically proven to be useful. Limiting the pool of possible node labels even further, may come at the expense of versatility but would benefit possible algorithmic treatment. By doing so, it might be possible to introduce *role specifications* or *agents* into the ADTree language. A promising attempt in the case of attack trees was published in [PAB+12].

Bibliography

[ABD$^+$06] Amer Aijaz, Bernd Bochow, Florian Dötzer, Andreas Festag, Matthias Gerlach, Rainer Kroh, and Tim Leinmüller. Attacks on Inter Vehicle Communication Systems - an Analysis. In *Proceedings of the 3rd International Workshop on Intelligent Transportation*, pages 189–194. Hamburg Institute of Technology, March 2006.

[abe11] abego. TreeLayout. http://code.google.com/p/treelayout/, 2011. Accessed July 2, 2013.

[ABS06] Shanai Ardi, David Byers, and Nahid Shahmehri. Towards a structured unified process for software security. In *Proceedings of the 2006 International Workshop on Software Engineering for Secure Systems*, pages 3–10, New York, NY, USA, 2006. ACM.

[ACC07] ACCURATE. A Center for Correct Usable Reliable Auditable and Transparent Elections: Annual Report 2006. http://accurate-voting.org/wp-content/uploads/2007/02/AR.2007.pdf, 2007. Accessed July 2, 2013.

[ACK10] Parosh Aziz Abdulla, Jonathan Cederberg, and Lisa Kaati. Analyzing the Security in the GSM Radio Network Using Attack Jungles. In Tiziana Margaria and Bernhard Steffen, editors, *Proceedings of the 4th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, volume 6415 of *LNCS*, pages 60–74, October, 2010. Springer.

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. Complexity of Finding Embeddings in a k-Tree. *SIAM Jornal of Algebraic and Discrete Methods*, 8:277–284, 1987.

[AJC06] Xiangdong An, Dan Jutla, and Nick Cercone. Privacy intrusion detection using dynamic Bayesian networks. In *Proceedings of the 8th International Conference for Electronic Commerce*, pages 208–215, Fredericton, Canada, August 2006. ACM.

[Ale03] Ian Alexander. Misuse cases: Use cases with hostile intent. *IEEE Software*, 20(1):58–66, 2003.

[Ame12] Amenaza. SecurITree. http://www.amenaza.com/, 2001–2012. Accessed July 2, 2013.

[Amo94] Edward G. Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

[And01] Ross J. Anderson. *Security engineering - a guide to building dependable distributed systems*. John Wiley & Sons, Inc., 1st edition, 2001.

[And10a] Alexander Andrusenko. Attack Forest. http://research.cyber.ee/~alexander/, 2010. Accessed July 2, 2013.

[And10b] Alexander Andrusenko. Ründepuude Metoodika Ja Seda Toetav Tarkvaraline Raamistik. Master's thesis, Tallinn University, 2010.

[ANI14] ANIKETOS. ANIKETOS: Ensuring Trustworthiness and Security in Service Composition, FP7 project, grant agreement 257930. http://www.aniketos.eu/, 2010–2014. Accessed July 2, 2013.

[AP08] Qutaibah Althebyan and Brajendra Panda. A Knowledge-Based Bayesian Model for Analyzing a System after an Insider Attack. In Sushil Jajodia, Pierangela Samarati, and Stelvio Cimato, editors, *Proceedings of the IFIP TC 11 23rd International Information Security Conference*, volume 278 of *IFIP*, pages 557–571. Springer, 2008.

[Arn85] Stefan Arnborg. Efficient algorithms for combinatorial problems on graphs with bounded decomposability – A survey. *BIT Numerical Mathematics*, 25(1):1–23, 1985.

[ATR12] ATREES. Attack Trees, project funded by the Fonds National de la Recherche, Luxembourg under grants C08/IS/26 and PHD-09-167. http://satoss.uni.lu/projects/atrees/, 2009–2012. Accessed July 2, 2013.

[AWK02] Paul Ammann, Duminda Wijesekera, and Saket Kaushik. Scalable, graph-based network vulnerability analysis. In *Proceedings of the 9th ACM Conference on Computer and ommunications Security*, pages 217–224, Washington, DC, USA, November 2002. ACM.

[BASD06] David Byers, Shanai Ardi, Nahid Shahmehri, and Claudiu Duma. Modeling software vulnerabilities with vulnerability cause graphs. In *Proceedings of the International Conference on Software Maintenance*, pages 411–422. IEEE, September 2006.

[BB03] Marc Bouissou and Jean-Louis Bon. A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes. *Reliability Engineering & System Safety*, 82(2):149–163, November 2003.

[BDP06] Stefano Bistarelli, Marco Dall'Aglio, and Pamela Peretti. Strategic Games on Defense Trees. In Theodosis Dimitrakos, Fabio Martinelli, Peter Y. A. Ryan, and Steve A. Schneider, editors, *Proceedings of the 4th International Workshop on Formal Aspects in Security and Trust*, volume 4691 of *LNCS*, pages 1–15. Springer, 2006.

[BF12] Alessandro Buoni and Mario Fedrizzi. Consensual Dynamics and Choquet Integral in an Attack Tree-based Fraud Detection System. In Joaquim Filipe and Ana L. N. Fred, editors, *Proceedings of the 4th International Conference on Agents and Artificial Intelligence*, pages 283–288. SciTePress, 2012.

[BFG11] Silvia Bortot, Mario Fedrizzi, and Silvio Giove. Modelling fraud detection by attack trees and Choquet integral. DISA Working Papers 2011/09, Department of Computer and Management Sciences, University of Trento, Italy, August 2011.

[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*. IEEE, December 2004.

[BFM10] Alessandro Buoni, Mario Fedrizzi, and József Mezei. A Delphi-Based Approach to Fraud Detection Using Attack Trees and Fuzzy Numbers. In *Proceedings of the IASK International Conferences*, pages 21–28. International Association for the Scientific Knowledge, 2010.

[BFM11] Alessandro Buoni, Mario Fedrizzi, and József Mezei. Combining Attack Trees and Fuzzy Numbers in a Multi-Agent Approach to Fraud Detection. *International Journal of Electronic Business*, 9(3):186–202, 2011.

[BFP06] Stefano Bistarelli, Fabio Fioravanti, and Pamela Peretti. Defense Trees for Economic Evaluation of Security Investments. In *Proceedings of the 1st International Conference on Availability, Reliability and Security*, pages 416–423. IEEE Computer Society, 2006.

[BH95] Jonathan P. Bowen and Michael G. Hinchey. Ten Commandments of Formal Methods. *IEEE Computer*, 28(4):56–63, 1995.

[BH06] Jonathan P. Bowen and Michael G. Hinchey. Ten Commandments of Formal Methods ...Ten Years Later. *IEEE Computer*, 39(1):40–48, 2006.

[BJ03] Sviatoslav Braynov and Murtuza Jadliwala. Representation and analysis of coordinated attacks. In *Proceedings of the 2003 ACM Workshop on Formal Methods in Security Engineering*, pages 43–51, Washington, DC, USA, 2003. ACM.

[BJL06] Christoph Buchheim, Michael Jünger, and Sebastian Leipert. Drawing rooted trees in linear time. *Software: Practice and Experience*, 36(6):651–665, May 2006.

[BLP$^+$06] Ahto Buldas, Peeter Laud, Jaan Priisalu, Märt Saarepera, and Jan Willemson. Rational Choice of Security Measures Via Multi-parameter Attack Trees. In Javier López, editor, *Proceedings of the 1st International Workshop on Critical Information Infrastructures Security*, volume 4347 of *LNCS*, pages 235–248. Springer, 2006.

[BM07] Ahto Buldas and Triinu Mägi. Practical Security Analysis of E-Voting Systems. In Miyaji et al. [MKR07], pages 320–335.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, New York, NY, USA, 1998.

[Bod93] Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, pages 226–234, New York, NY, USA, 1993. ACM.

[Bou07] Marc Bouissou. A Generalization of Dynamic Fault Trees through Boolean logic Driven Markov Processes (BDMP). In Terje Aven and Jan Erik Vinnem, editors, *Proceedings of the 16th European Safety and Reliability Conference*, Stavanger, Norway, June 2007. Taylor & Francis Group.

[BP03] Phillip J. Brooke and Richard F. Paige. Fault trees for security system design and analysis. *Computers & Security*, 22(3):256–264, 2003.

[BP10] Dejan Baca and Kai Petersen. Prioritizing Countermeasures through the Countermeasure Method for Software Security (CM-Sec). In Muhammad Ali Babar, Matias Vierimaa, and Markku Oivo, editors, *Proceedings of the 11th International Conference on Product-Focused Software Process Improvement*, volume 6156 of *LNCS*, pages 176–190. Springer, 2010.

[BPT08] Stefano Bistarelli, Pamela Peretti, and Irina Trubitsyna. Analyzing Security Scenarios Using Defence Trees and Answer Set Programming. *Electronic Notes in Theoretical Computer Science*, 197(2):121–129, 2008.

[BPU$^+$05] Donald L. Buckshaw, Gregory S. Parnell, Willard L. Unkenholz, Donald L. Parks, James M. Wallner, and O. Sami Saydjari. Mission Oriented Risk and Design Analysis of Critical Information Systems. *Military Operations Research*, 10(2):19–38, 2005.

[BS07] David Byers and Nahid Shahmehri. Design of a Process for Software Security. In *Proceedings of the 2nd International Conference on Availability, Reliability and Security*, pages 301–309. IEEE Computer Society, April 2007.

[BS08] David Byers and Nahid Shahmehri. A Cause-Based Approach to Preventing Software Vulnerabilities. In *Proceedings of the 2008 Third International Conference on Availability, Reliability and Security*, pages 276–283, Washington, DC, USA, 2008. IEEE Computer Society.

[BS09] Patrik Berander and Mikael Svahnberg. Evaluating two ways of calculating priorities in requirements hierarchies - An experiment on hierarchical cumulative voting. *Journal of Systems and Software*, 82(5):836–850, May 2009.

[BS10] David Byers and Nahid Shahmehri. Unified modeling of attacks, vulnerabilities and security activities. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, pages 36–42, New York, NY, USA, 2010. ACM.

[BS12] Ahto Buldas and Roman Stepanenko. Upper Bounds for Adversaries' Utility in Attack Trees. In Jens Grossklags and Jean C. Walrand, editors, *Proceedings of the 3rd International Conference on Decision and Game Theory for Security*, volume 7638 of *LNCS*, pages 98–117. Springer, 2012.

[Buo10] Alessandro Buoni. Fraud Detection: From Basic Techniques to a Multi-Agent Approach. In *Proceedings of the 2010 International Conference on Management and Service Science*, pages 1516–1519. IEEE, August 2010.

[Buo12] Alessandro Buoni. *Fraud Detection in the Banking Sector*. PhD thesis, Åbo Akademi University, Finland, 2012.

[Car09] Carnegie Mellon University. SQUARE: System Quality Requirements Engineering. http://www.cert.org/sse/square-tool.html, 2004–2009. Accessed July 2, 2013.

[CCF04] Sean Convery, David Cook, and Matt Franz. An Attack Tree for the Border Gateway Protocol. http://tools.ietf.org/html/draft-ietf-rpsec-bgpattack-00, 2004. Accessed July 2, 2013.

[CDG+07] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. http://www.grappa.univ-lille3.fr/tata, 2007. Accessed July 2, 2013.

[CGP08] Robert Cowan, Michael Grimaila, and Raju Patel. Using Attack and Protection Trees to Evaluate Risk in an Embedded Weapon System. In *Proceedings of the 3rd International Conference on Information Warfare and Security*, pages 97–108, Omaha, Nebraska, USA, April 2008.

[CH07] Nicolas Chaufette and Tommie Haag. Vulnerability Cause Graphs: A Case of Study. http://www.ida.liu.se/~TDDD17/oldprojects/2007/projects/3.pdf, 2007. Accessed July 2, 2013.

[CH11] Yves Crama and Peter L. Hammer. *Boolean Functions: Theory, Algorithms, and Applications*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2011.

[CK00] Horatiu Cirstea and Claude Kirchner. The simply typed rewriting calculus. *Electronic Notes in Theoretical Computer Science*, 36:24–42, 2000.

[CKK10] Giovanni Cagalaban, Taihoon Kim, and Seoksoo Kim. Improving SCADA control systems security with software vulnerability analysis. In *Proceedings of the 12th WSEAS International Conference on Automatic Control, Modelling &#38; Simulation*, pages 409–414, Stevens Point, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

[CS05] Maria Chudnovsky and Paul D. Seymour. The structure of claw-free graphs. In Bridget S. Webb, editor, *Surveys in Combinatorics*, volume 327 of *London Mathematical Society Lecture Note Series*, pages 153–171. Cambridge University Press, 2005.

[CSTH08] Kevin Clark, Ethan Singleton, Stephen Tyree, and John Hale. StrataGem: risk assessment through mission modeling. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, pages 51–58, Alexandria, VA, USA, October 2008. ACM.

[CTDH04] Kevin Clark, Stephen Tyree, Jerald Dawkins, and John Hale. Qualitative and quantitative analytical techniques for network security assessment. In *Proceedings of the 5th IEEE Systems, Man and Cybernetics Information Assurance Workshop*, pages 321–328, West Point, NY, USA, June 2004.

[CW96] Edmund M. Clarke and Jeannette M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[ÇY06] Seyit Ahmet Çamtepe and Bülent Yener. A Formal Method for Attack Modeling and Detection. Technical Report TR-06-01, Rensselaer Polytechnic Institute, Troy, NY, USA, 2006.

[ÇY07] Seyit Ahmet Çamtepe and Bülent Yener. Modeling and detection of complex attacks. In *Proceedings of the 3rd International Conference on Security and Privacy in Communications Networks*, pages 234–243, Nice, France, September 2007. IEEE.

[Dac94] Marc Dacier. *Vers une évaluation quantitative de la sécurité informatique.* PhD thesis, Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS), 1994.

[DBB90] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Fault Trees and Sequence Dependencies. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 286–293, Los Angeles, CA, USA, January 1990. IEEE Computer Society.

[DBB92] Joanne Bechta Dugan, Salvatore J. Bavuso, and Mark A. Boyd. Dynamic fault tree models for fault tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.

[DD94] Marc Dacier and Yves Deswarte. Privilege graph: An extension to the typed access matrix model. In Dieter Gollmann, editor, *Proceedings of the 3rd European Symposium on Research in Computer Security*, volume 875 of *LNCS*, pages 319–334. Springer, 1994.

[DDK96] Marc Dacier, Yves Deswarte, and Mohamed Kaâniche. Models and tools for quantitative assessment of operational security. In Sokratis K. Katsikas and Dimitris Gritzalis, editors, *Information Systems Security, Facing the information society of the 21st Century*, volume 54 of *IFIP*, pages 177–186. Chapman & Hall, 1996.

[Dec99] Rina Dechter. Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1–2):41–85, 1999.

[Dec03] Rina Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

[Dec13] Decision Systems Laboratory, University of Pittsburgh. GeNIe & SMILE. `http://genie.sis.pitt.edu/`, 1996–2013. Accessed July 2, 2013.

[DEMR10] 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. *International Journal of Security and Networks*, 5(2):87–95, 2010.

[Dep03] Department of Engineering, University of Maryland. Fault Tree Analysis Programs. `http://www.enre.umd.edu/tools/ftap.htm`, 2003. Accessed October 5, 2012.

[DH04] Jerald Dawkins and John Hale. A systematic approach to multi-stage network attack analysis. In *Proceedings of the 2nd IEEE International Information Assurance Workshop*, pages 48–56, Charlotte, NC, USA, April 2004. IEEE.

[DK05] Ram Dantu and Prakash Kolan. Risk Management Using Behavior Based Bayesian Networks. In Paul B. Kantor, Gheorghe Muresan, Fred Roberts, Daniel Dajun Zeng, Fei-Yue Wang, Hsinchun Chen, and Ralph C. Merkle, editors, *Proceedings of the 2005 IEEE International Conference on IEEE Intelligence and Security Informatics*, volume 3495 of *LNCS*, pages 115–126. Springer, 2005.

[DKAL07] Ram Dantu, Prakash Kolan, Robert Akl, and Kall Loper. Classification of attributes and behavior in risk management using bayesian networks. In *Proceedings of the 2007 IEEE International Conference on Intelligence and Security Informatics*, pages 71–74. IEEE, 2007.

[DKaWC09] Ram Dantu, Prakash Kolan, and Jo ao W. Cangussu. Network risk management using attacker profiling. *Security and Communication Networks*, 2(1):83–96, 2009.

[DLD02] Kristopher Daley, Ryan Larson, and Jerald Dawkins. A Structural Framework for Modeling Multi-Stage Network Attacks. In *Proceedings of the 31st International Conference on Parallel Processing Workshops*, pages 5–10. 31st International Conference on Parallel Processing Workshops, August 2002.

[DLDZ12] Suguo Du, Xiaolong Li, Junbo Du, and Haojin Zhu. An attack-and-defence game for security assessment in vehicular ad hoc networks. *Peer-to-Peer Networking and Applications*, 5(1):1–14, 2012.

[DLK04] Ram Dantu, Kall Loper, and Prakash Kolan. Risk management using behavior based attack graphs. In *Proceedings of the International Conference on Information Technology: Coding and Computing*, volume 1, pages 445–449, April 2004.

[DMCR06] George C. Dalton II, Robert F. Mills, John M. Colombi, and Richard A. Raines. Analyzing Attack Trees using Generalized Stochastic Petri Nets. In *Proceedings of the IEEE Information Assurance Workshop*, pages 116–123, West Point, NY, USA, June 2006. IEEE.

[DMRW09] Erik D. Demaine, Shay Mozes, Benjamin Rossman, and Oren Weimann. An Optimal Decomposition Algorithm for Tree Edit Distance. *ACM Transactions on Algorithms*, 6(1):2:1–2:19, December 2009.

[DP90] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, 1990.

[DP08] Salvatore Distefano and Antonio Puliafito. Dependability evaluation using dynamic reliability block diagrams and dynamic fault trees. *IEEE Transactions on Dependable and Secure Computing*, 6(1):4–17, 2008.

[DPRW07] Rinku Dewri, Nayot Poolsappasit, Indrajit Ray, and Darrell Whitley. Optimal security hardening using multi-objective optimization on attack tree models of networks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 204–213, New York, NY, USA, 2007. ACM.

[DRMS+06] Mamadou H. Diallo, Jose Romero-Mariona, Susan E. Sim, Thomas A. Alspaugh, and Debra J. Richardson. A Comparative Evaluation of Three Approaches to Specifying Security Requirements. In *Proceedings of the 12th International Working Conference on Requirements Engineering: Foundation for Software Quality*, June 2006.

[DRPW12] Rinku Dewri, Indrajit Ray, Nayot Poolsappasit, and Darrell Whitley. Optimal security hardening on attack tree models of networks: a cost-benefit analysis. *International Journal of Information Security*, 11(3):167–188, June 2012.

[DSC00] Joanne Bechta Dugan, Kevin J. Sullivan, and David Coppit. Developing a Low-Cost, High-Quality Software Tool for Dynamic Fault Tree Analysis. *IEEE Transactions on Reliability*, 49(1):49–59, 2000.

[EBv+10] Barry Charles Ezell, Steven P. Bennett, Detlof von Winterfeldt, John Sokolowski, and Andrew J. Collins. Probabilistic risk analysis and terrorism risk. *Risk analysis an official publication of the Society for Risk Analysis*, 30(4):575–589, 2010.

[Edg07] Kenneth S. Edge. *A Framework for Analyzing and Mitigating the Vulnerabilities of Complex Systems via Attack and Protection Trees*. PhD thesis, Air Force Institute of Technology, Wright Patterson Air Force Base, OH, USA, July 2007.

[EDRM06] Kenneth S. Edge, George C. Dalton II, Richard A. Raines, and Robert F. Mills. Using Attack and Protection Trees to Analyze Threats and Defenses to Homeland Security. In *Proceedings of the 2006 Military Communications Conference*, pages 1–7. IEEE, 2006.

[EHK+04] Shelby Evans, David Heinbuch, Elizabeth Kyule, John Piorkowski, and James Wallner. Risk-based systems security engineering: stopping attacks with intention. *IEEE Security and Privacy*, 2(6):59–62, 2004.

[Ele12] Electricité de France - Research and Development. KB3 Platform tools. `http://research.edf.com/research-and-the-scientific-community/software/kb3-44337.html`, 2011–2012. Accessed July 2, 2013.

[EPPC11] Jung-Ho Eom, Min-Woo Park, Seon-Ho Park, and Tai-Myoung Chung. A Framework of Defense System for Prevention of Insider's Malicious Behaviors. In *Proceedings of the 13th International Conference on Advanced Communication Technology*, pages 982–987. IEEE, February 2011.

[ERG$^+$07] Kenneth Edge, Richard Raines, Michael Grimaila, Rusty Baldwin, Robert Bennington, and Christopher Reuter. The Use of Attack and Protection Trees to Analyze Security for an Online Banking System. In *Proceedings of the 40th Annual Hawaii International Conference on System Sciences*, page 144b. IEEE, January 2007.

[Eri99] Clifton A. Ericson II. Fault Tree Analysis - A History. In *Proceedings of the 17th International System Safety Conference*, Orlando, FL, USA, August 1999.

[ES09] Mathias Ekstedt and Teodor Sommestad. Enterprise architecture models for cyber security analysis. In *Proceedings of the 2009 IEEE/PES Power System Conference and Exposition*, pages 1–6, Seattle, USA, March 2009. IEEE.

[Esp07] Jeanne H. Espedalen. Attack Trees Describing Security in Distributed Internet-Enabled Metrology. Master's thesis, Gjøvik University, 2007.

[EVI11] EVITA. E-safety vehicle intrusion protected applications: FP7 project, grant agreement 224275. http://www.evita-project.org/, 2008–2011. Accessed July 2, 2013.

[FBMJ10] Plínio César Simões Fernandes, Tania Basso, Regina Moraes, and Mario Jino. Attack Trees Modeling for Security Tests in Web Applications. In *Proceedings of the 4th Brazilian Workshop on Systematic and Automated Software Testing*, pages 3–12. SBC, November 2010.

[FCW$^+$05] Casey Fung, Yi-Liang Chen, Xinyu Wang, J. Lee, R. Tarquini, M. Anderson, and R. Linger. Survivability analysis of distributed systems using attack tree methodology. In *Proceedings of the 2005 IEEE Military Communications Conference*, volume 1, pages 583–589. IEEE, October 2005.

[Fir03] Donald J. Firesmith. Security Use Cases. *Journal of Object Technology*, 2(3):53–64, May 2003.

[FJN93] Ralph Freese, Jaroslav Ježek, and James B. Nation. Term Rewrite Systems for Lattice Theory. *Journal of Symbolic Computation*, 16(3):279–288, 1993.

[FMC09] Igor Nai Fovino, Marcelo Masera, and Alessio De Cian. Integrating cyber attacks within fault trees. *Reliability Engineering & System Safety*, 94(9):1394–1402, September 2009.

[Fos02] Nathalie L. Foster. *The application of software and safety engineering techniques to security protocol development.* PhD thesis, University of York, 2002.

[FSEJ08] Ulrik Franke, Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. Defense Graphs and Enterprise Architecture for Information Assurance Analysis. In *Proceedings of the 26th Army Science Conference*, Orlando, FL, USA, December 2008.

[FW08] Marcel Frigault and Lingyu Wang. Measuring Network Security Using Bayesian Network-Based Attack Graphs. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications*, pages 698–703. IEEE, Jul–Aug 2008.

[FWM$^+$05] Bingrui Foo, Yu-Sung Wu, Yu-Chun Mao, Saurabh Bagchi, and Eugene Spafford. ADEPTS: adaptive intrusion response using attack graphs in an e-commerce environment. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 508–517. IEEE Computer Society, Jun–Jul 2005.

[FWSJ08] Marcel Frigault, Lingyu Wang, Anoop Singhal, and Sushuil Jajodia. Measuring network security using dynamic Bayesian network. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, pages 23–30, Alexandria, VA, USA, October 2008. ACM.

[FX12] Nan Feng and Jing Xie. A Bayesian networks-based security risk analysis model for information systems integrating the observed cases with expert experience. *Scientific Research and Essays*, 7(10):1103–1112, 2012.

[GGR93] Jens Grabowski, Peter Graubmann, and Ekkart Rudolph. The Standardization of Message Sequence Charts. In *Proceedings of the Software Engineering Standards Symposium*, pages 48–63, 1993.

[GJ08] Lars Grunske and David Joyce. Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles. *Journal of Systems and Software*, 81(8):1327–1345, 2008.

[Grä03] George A. Grätzer. *General Lattice Theory.* Birkhäuser, 2003.

[GTWW77] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. Initial Algebra Semantics and Continuous Algebras. *Journal of the ACM*, 24(1):68–95, 1977.

[Gur89] Eitan M. Gurari. *Introduction to the Theory of Computation.* Computer Science Press, 1989.

[HAF$^+$09] Olaf Henniger, Ludovic Apvrille, Andreas Fuchs, Yves Roudier, Alastair Ruddle, and Benjamin Weyl. Security requirements for automotive on-board networks. In *Proceedings of the 9th International Conference on Intelligent Transport Systems Telecommunications*, pages 641–646, Lille, France, October 2009. IEEE.

[Hal03] Joseph Y. Halpern. *Reasoning about Uncertainty.* MIT Press, Cambridge, MA, USA, 2003.

[Har10] Patrick D. Harrington. *Noncooperative potential Games to improve network security.* PhD thesis, Oklahoma State University, USA, 2010.

[HD08] Viktor Horvath and Till Dörges. From security patterns to implementation using petri nets. In *Proceedings of the 4th International Workshop on Software Engineering for Secure Systems*, pages 17–24, New York, NY, USA, 2008. ACM.

[HFE09] Siv Hilde Houmb, Virginia N. L. Franqueira, and Erlend A. Engum. Quantifying security risk level from CVSS estimates of frequency and impact. *Journal of Systems and Software*, 83(9):1662–1634, 2009.

[HL02] Michael Howard and David LeBlanc. *Writing Secure Code*. Microsoft Press, 2nd edition, 2002.

[HLOS06] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover Security Design Flaws Using The STRIDE Approach. `http://msdn.microsoft.com/en-us/magazine/cc163519.aspx`, 2006. Accessed July 2, 2013.

[Hog07] Ida Hogganvik. *A graphical approach to security risk analysis*. PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo, 2007.

[HS13] Yoni Halpern and David Sontag. Unsupervised Learning of Noisy-Or Bayesian Networks. In *Proceedings of the 29th Conference on Uncertainty in Artificial Intelligence* , pages 272–281. AUAI Press, 2013.

[HUJ+04] Victoria Higuero, Juan José Unzilla, Eduardo Jacob, Purificación Sáiz, and David Luengo. Application of 'Attack Trees' Technique to Copyright Protection Protocols Using Watermarking and Definition of a New Transactions Protocol SecDP (Secure Distribution Protocol). In *Proceedings of the 2nd International Workshop on Multimedia Interactive Protocols and Systems*, volume 3311 of *LNCS*, pages 264–275, Grenoble, France, September 2004. Springer.

[HVOM08] John Homer, Ashok Varikuti, Xinming Ou, and Miles A. McQueen. Improving Attack Graph Visualization through Data Reduction and Attack Grouping. In *Proceedings of the 5th International Workshop on Visualization For Computer Security*, pages 68–79, Cambridge, MA, USA, September 2008. Springer.

[HWS+02] Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, and Robyn Lutz. A Software Fault Tree Approach to Requirements Analysis of an Intrusion Detection System. *Journal of Requirements Engineering*, 7(4):207–220, December 2002.

[HWS+07] Guy Helmer, Johnny Wong, Mark Slagell, Vasant Honavar, Les Miller, Yanxin Wang, Xia Wang, and Natalia Stakhanova. Software fault tree and coloured Petri net-based specification, design and implementation of agent-based intrusion detection systems. *International Journal of Information and Computer Security*, 1(1/2):109–142, 2007.

[ILP06] Kyle W. Ingols, Richard Lippmann, and Keith Piwowarski. Practical Attack Graph Generation for Network Defense. In *Proceedings of the 22nd Annual Computer Security Applications Conference*, pages 121–130, Washington, DC, USA, December 2006. IEEE Computer Society.

[Iso11] Isograph. AttackTree+. `http://www.isograph-software.com/2011/software/attacktree/`, 1986–2011. Accessed July 2, 2013.

[ISO12] ISO/IEC 15408. Common Criteria for Information Technology Security Evaluation (version 3.1, revision 4). `http://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4.pdf`, 2012. Accessed July 2, 2013.

[JEBR10] Christian Jung, Frank Elberzhager, Alessandra Bagnato, and Fabio Raiteri. Practical Experience Gained from Modeling Security Goals: Using SGITs in an Industrial Project. In *Proceedings of the 5th International Conference on Availability, Reliability and Security*, pages 531–536, Los Alamitos, CA, USA, February 2010. IEEE Computer Society.

[JJSU07] Pontus Johnson, Erik Johansson, Teodor Sommestad, and Johan Ullberg. A Tool for Enterprise Architecture Analysis. In *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference*, pages 142–156, Annapolis, MD, USA, October 2007. IEEE Computer Society.

[JLNS07] Pontus Johnson, Robert Lagerström, Per Närman, and Mårten Simonsson. Enterprise architecture analysis with extended influence diagrams. *Information Systems Frontiers*, 9(2-3):163–180, July 2007.

[JN07] Finn V. Jensen and Thomas D. Nielsen. *Bayesian Networks and Decision Graphs*. Springer, 2nd edition, 2007.

[JNO05] Sushil Jajodia, Steven Noel, and Brian O'Berry. *Managing Cyber Threats: Issues, Approaches, and Challenges*, chapter Topological Analysis of Network Attack Vulnerability, pages 247–266. Springer, 2005.

[Joh11] Chris W. Johnson. Using Assurance Cases and Boolean logic Driven Markov Processes to Formalise Cyber Security Concerns for Safety-Critical Interaction with Global Navigation Satellite Systems. *Electronic Communication of the European Association of Software Science and Technology*, 45:1–18, 2011.

[Jür10] Aivo Jürgenson. *Efficient Semantics of Parallel and Serial Models of Attack Trees*. PhD thesis, Tallinn University of Technology, Faculty of Information Technology, Department of Informatics, 2010.

[JW07] Aivo Jürgenson and Jan Willemson. Processing Multi-Parameter Attack-trees with Estimated Parameter Values. In Miyaji et al. [MKR07], pages 308–319.

[JW08] Aivo Jürgenson and Jan Willemson. Computing Exact Outcomes of Multi-parameter Attack Trees. In Robert Meersman and Zahir Tari, editors, *Proceedings of the OTM Conferences (2)*, volume 5332 of *LNCS*, pages 1036–1051. Springer, 2008.

[JW10] Aivo Jürgenson and Jan Willemson. On Fast and Approximate Attack Tree Computations. In *Proceedings of the 6th International Conference on Information Security Practice and Experience*, pages 56–66. Springer, 2010.

[Kar05] Kaarina Karppinen. Security Measurement Based on Attack Trees in a Mobile Ad Hoc Network Environment. Master's thesis, VTT and University of Oulu, 2005.

[KBPC12] Siwar Kriaa, Marc Bouissou, and Ludovic Piètre-Cambacédès. Modeling the Stuxnet Attack with BDMP: Towards More Formal Risk Assessments. In *Proceedings of the 7th International Conference on Risks and Security of Internet and Systems*, Cork, Ireland, October 2012. IEEE Computer Society.

[KEE10] Johannes Kloos, Frank Elberzhager, and Robert Eschbach. Systematic Construction of Goal Indicator Trees for Indicator-Based Dependability Inspections. In *Proceedings of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 279–282. IEEE, September 2010.

[Kha09] Parvaiz Ahmed Khand. System level security modeling using attack trees. In *Proceedings of the 2nd International Conference on Computer, Control and Communication*, pages 115–120, Karachi, Pakistan, February 2009. IEEE.

[Kie98] Darrell M. Kienzle. *Practical Computer Security Analysis.* PhD thesis, School of Engineering and Applied Science, University of Virginia, USA, 1998.

[Kim05] Suh-Ryung Kim. Graphs with One Hole and Competition Number One. *Journal of the Korean Mathematical Society*, 42(6):1251–1264, 2005.

[KKO$^+$10] Vikash Katta, Péter Kárpáti, Andreas L. Opdahl, Christian Raspotnig, and Guttorm Sindre. Comparing Two Techniques for Intrusion Visualization. In Patrick van Bommel, Stijn Hoppenbrouwers, Sietse Overbeek, Erik Proper, and Joseph Barjis, editors, *Proceedings of the 3rd IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling*, volume 68 of *LNBIP*, pages 1–15. Springer, 2010.

[Koh03] Jürg Kohlas. *Information Algebras: Generic Structures for Inference.* Springer, 2003.

[Koo12] Laurens Koot. *Security of mobile TAN on smartphones.* PhD thesis, Radboud University Nijmegen, Faculty of Science, The Netherlands, 2012.

[KS94] Sandeep Kumar and Eugene H. Spafford. A Pattern-Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, Baltimore, USA, October 1994.

[KS06] Igor Kotenko and Mikhail Stepashkin. Analyzing Network Security using Malefactor Action Graphs. *International Journal of Computer Science and Network Security*, 6(6):226–235, 2006.

[KS07] Parvaiz Ahmed Khand and Poong Hyun Seong. An Attack model development process for the Cyber Security of Safety Related Nuclear Digital I&C Systems. In *Proceedings of the Korean Nuclear Society, Fall Meeting*, Korea, October 2007.

[KSM12] Péter Kárpáti, Guttorm Sindre, and Raimundas Matulevicius. Comparing Misuse Case and Mal-Activity Diagrams for Modelling Social Engineering Attacks. *International Journal of Secure Software Engineering*, 3(2):54–73, 2012.

[KSO10a] Péter Kárpáti, Guttorm Sindre, and Andreas L. Opdahl. Towards a Hacker Attack Representation Method. In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 92–101. Springer, July 2010.

[KSO10b] Péter Kárpáti, Guttorm Sindre, and Andreas L. Opdahl. Visualizing cyber attacks with misuse case maps. In *Proceedings of the 16th International Working Conference on Requirements Engineering: Foundation for Software Quality*, volume 6182 of *LNCS*, pages 262–275, Essen, Germany, June 2010. Springer.

[KSZM13] Martin Korp, Christian Sternagel, Harald Zankl, and Aart Middeldorp. Tyrolean Termination Tool 2. http://cl-informatik.uibk.ac.at/software/ttt2/index.php, 2009–2013. Accessed July 2, 2013.

[KW97] Darrell M. Kienzle and William A. Wulf. A Practical Approach to Security Assessment. In *Proceedings of the 1997 New Security Paradigms Workshop*, pages 5–16. ACM, September 1997.

[Laz10] Eric L. Lazarus. AttackDog. http://decisionsmith.com/doc/adog, 2010. Accessed July 21, 2010.

[LB07] David John Leversage and Eric James Byres. Comparing Electronic Battlefields: Using Mean Time-To-Compromise as a Comparative Security Metric. In *Proceedings of the 4th International Conference on Methods, Models, and Architectures for Network Security*, pages 213–227, St Petersburg, Russia, September 2007. Springer.

[LB08] David John Leversage and Eric James Byres. Estimating a System's Mean Time-to-Compromise. *IEEE Security and Privacy*, 6(1):52–60, January 2008.

[LDEH11] Eric L. Lazarus, David L. Dill, Jeremy Epstein, and Joseph Lorenzo Hall. Applying a Reusable Election Threat Model at the County Level. In *Proceedings of the 2011 Conference on Electronic voting Technology / Workshop on Trustworthy Elections*, pages 1–14, Berkeley, CA, USA, August 2011. USENIX Association.

[Lev95] Nancy G. Leveson. *Safeware: System Safety and Computers*. Addison-Wesley Professional, April 1995.

[LH83] Nancy G. Leveson and Peter R. Harvey. Software fault tree analysis. *Journal of Systems and Software*, 3(2):173–181, 1983.

[LI05] Richard Lippmann and Kyle W. Ingols. An annotated review of past papers on attack graphs. Project Report ESC-TR-2005-054, Massachusetts Institute of Technology (MIT), Lincoln Laboratory, March 2005.

[LJN07] Robert Lagerström, Pontus Johnson, and Per Närman. Extended Influence Diagram Generation. In Ricardo Jardim-Gonçalves, Jörg P. Müller, Kai Mertins, and Martin Zelm, editors, *Proceedings of the 3th International Conference on Interoperability for Enterprise Software and Applications*, pages 599–602. Springer, 2007.

[LLFH09] Xiaohong Li, Ran Liu, Zhiyong Feng, and Ke He. Threat modeling-oriented attack path evaluating algorithm. *Transactions of Tianjin University*, 15(3):162–167, 2009.

[LM01] Richard C. Linger and Andrew P. Moore. Foundations for Survivable System Development: Service Traces, Intrusion Traces, and Evaluation Models. http://www.cert.org/archive/pdf/01tr029.pdf, 2001. Accessed July 2, 2013.

[LM05] Yuan Liu and Hong Man. Network vulnerability assessment using Bayesian networks. In *SPIE Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security*, volume 5812, pages 61–71, Orlando, FL, USA, March 2005.

[Lou11] George Robert Louthan IV. Hybrid Attack Graphs for Modeling Cyber-physical Systems. Master's thesis, University of Tulsa, USA, 2011.

[LZRL09] Xiaoli Lin, Pavol Zavarsky, Ron Ruhl, and Dale Lindskog. Threat Modeling for CSRF Attacks. In *Proceedings of the International Conference on Computational Science and Engineering*, volume 3, pages 486–491. IEEE Computer Society, August 2009.

[Mäg07] Triinu Mägi. Practical Security Analysis of E-voting Systems. Master's thesis, Tallin University of Technology, Faculty of Information Technology, Department of Informatics, Estonia, 2007.

[Man08] Pratyusa K. Manadhata. *An Attack Surface Metric*. PhD thesis, Carnegie Mellon University, December 2008.

[Mar08] Charles Marshall. Attack Trees and Their Uses in BGP and SMTP Analysis. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.3609, 2008. Accessed July 2, 2013.

[MBFB05] Miles A. McQueen, Wayne F. Boyer, Mark A. Flynn, and George A. Beitel. Time-to-compromise model for cyber risk reduction estimation. In *Proceedings of the 1st Workshop on Quality of Protection*, pages 49–64, Milan, Italy, September 2005. Springer.

[MBFB06] Miles A. McQueen, Wayne F. Boyer, Mark A. Flynn, and George A. Beitel. Quantitative Cyber Risk Reduction Estimation Methodology for a Small SCADA Control System. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, volume 9, pages 226–237, Hawaii, USA, January 2006. IEEE.

[MBG08] Samresh Malhotra, Somak Bhattacharya, and S. K. Ghosh. A Vulnerability and Exploit Independent Approach for Attack Path Prediction. In *Proceedings of the 8th IEEE International Conference on Computer and Information Technology Workshops*, pages 282–287, Sydney, Australia, July 2008. IEEE Computer Society.

[MBZ+06] Vaibhav Mehta, Constantinos Bartzis, Haifeng Zhu, Edmund Clarke, and Jeannette Wing. Ranking Attack Graphs. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection*, volume 4219 of *LNCS*, pages 127–144, Hamburg, Germany, September 2006. Springer.

[McD00] John P. McDermott. Attack net penetration testing. In *Proceedings of the 2000 New Security Paradigms Workshop*, pages 15–21, Cork, Ireland, September 2000. ACM.

[MCM+09] Amel Mammar, Ana Cavalli, Edgardo Montes de Oca, Shanai Ardi, David Byers, and Nahid Shahmehri. Modélisation et Détection Formelles de Vulnérabilités Logicielles par le Test Passif. In *4ème Conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information*, page 12pp, June 2009.

[MCM11] Anderson Morais, Ana Cavalli, and Eliane Martins. A Model-Based Attack Injection Approach for Security Validation. In *Proceedings of the 4th International Conference on Security of Information and Networks*, pages 103–110, New York, NY, USA, 2011. ACM.

[Mea96] Catherine Meadows. A representation of Protocol Attacks for Risk Assessment. In *Proceedings of the DIMACS Workshop on Network Threats*, pages 1–10, New Brunswick, NJ, USA, December 1996. American Mathematical Society.

[MEL01] Andrew P. Moore, Robert J. Ellison, and Richard C. Linger. Attack Modeling for Information Security and Survivability. Technical Note CMU/SEI-2001-TN-001, Carnegie Mellon University, March 2001.

[Mel10] Per Håkon Meland. SeaMonster. http://sourceforge.net/projects/seamonster/, 2010. Accessed July 2, 2013.

[MF99] John P. McDermott and Chris Fox. Using abuse case models for security requirements analysis. In *Proceedings of the 15th Annual Computer Security Applications Conference*, pages 55–64, Phoenix, USA, December 1999. IEEE Computer Society.

[MFC07] Marcelo Masera, Igor Nai Fovino, and Alessio De Cian. Integrating cyber attacks within fault trees. In Terje Aven and Jan Erik Vinnem, editors, *Proceedings of the 16th European Safety and Reliability Conference*, pages 1–8, Stavanger, Norway, 2007. Taylor & Francis Group.

[MH68] Jim E. Matheson and Ron A. Howard. *An Introduction to Decision Analysis.* Strategic Decisions Group, Menlo Park, CA, 1968.

[MHH+09] Aaron Marback, Do Hyunsook, Ke He, Samuel Kondamarri, and Dianxiang Xu. Security test generation using threat trees. In *Proceedings of the 2009 ICSE Workshop on Automation of Software Testing*, pages 62–69. IEEE, May 2009.

[MHS05] Nancy R. Mead, Eric D. Hough, and Theodore R. Stehney II. Security Quality Requirements Engineering (SQUARE) Methodology. Technical Report CMU/SEI-2005-TR-009, Carnegie Mellon University, 2005.

[MHW09] Luke Mirowski, Jacqueline Hartnett, and Raymond Williams. An RFID Attacker Behavior Taxonomy. *IEEE Pervasive Computing*, 8(4):79–84, 2009.

[MK97] Ira S. Moskowitz and Myong H. Kang. An insecurity flow model. In *Proceedings of the 1997 New Security Paradigms Workshop*, pages 61–74. ACM, September 1997.

[MKR07] Atsuko Miyaji, Hiroaki Kikuchi, and Kai Rannenberg, editors. *Proceedings of the 2nd International Workshop on Security Advances in Information and Computer Security*, volume 4752 of *LNCS*. Springer, October 2007.

[MKY12] Shivani Mishra, Krishna Kant, and R. S. Yadav. Multi Tree View of Complex Attack – Stuxnet. In *Proceedings of the 2nd International Conference on Advances in Computing and Information Technology*, pages 171–188, Chennai, India, July 2012.

[MM08] Drake Patrick Mirembe and Maybin Muyeba. Threat Modeling Revisited: Improving Expressiveness of Attack. In *Proceedings of the 2008 2nd UKSIM European Symposium on Computer Modeling and Simulation*, pages 93–98, Washington, DC, USA, 2008. IEEE Computer Society.

[MMCJ09] Anderson Nunes Paiva Morais, Eliane Martins, Ana R. Cavalli, and Willy Jimenez. Security Protocol Testing Using Attack Trees. In *Proceedings of the International Conference on Computational Science and Engineering*, pages 690–697. IEEE Computer Society, August 2009.

[MO05] Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In Dongho Won and Seungjoo Kim, editors, *Proceedings of the 8th International Conference on Information Security and Cryptology*, volume 3935 of *LNCS*, pages 186–198. Springer, 2005.

[Mob00] Fredrik Moberg. Security Analysis of an Information System Using an Attack Tree-based Methodology. Master's thesis, Chalmers University of Technology, 2000.

[MPM10a] Stephen McLaughlin, Dmitry Podkuiko, and Patrick McDaniel. Energy theft in the advanced metering infrastructure. In *Proceedings of the 4th International Conference on Critical Information Infrastructures Security*, pages 176–187. Springer, 2010.

[MPM$^+$10b] Stephen McLaughlin, Dmitry Podkuiko, Sergei Miadzvezhanka, Adam Delozier, and Patrick McDaniel. Multi-vendor penetration testing in the advanced metering infrastructure. In *Proceedings of the 26th Annual Computer Security Applications Conference(ACSAC)*, pages 107–116, Austin, TX, USA,, December 2010. ACM.

[MRZ05] Claude Marché, Albert Rubio, and Hans Zantema. Termination Problem Data Base: format of input files. http://www.lri.fr/~marche/tpdb/format.html, 2005. Accessed July 2, 2013.

[MS05] Nancy R. Mead and Ted Stehney. Security quality requirements engineering (SQUARE) methodology. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–7, May 2005.

[MSH+08] Per Håkon Meland, Daniele Giuseppe Spampinato, Eilev Hagen, Egil Trygve Baadshaug, Kris-Mikael Krister, and Ketil Sandanger Velle. SeaMonster: Providing tool support for security modeling. In *Proceedings of the Norsk Informasjonssikkerhetskonferanse*. Tapir akademisk forlag, November 2008.

[MTF11] Theodore W. Manikas, Mitchell A. Thornton, and David Y. Feinstein. Using Multiple-Valued Logic Decision Diagrams to Model System Threat Probabilities. In *Proceedings of the 41st IEEE International Symposium on Multiple-Valued Logic*, pages 263 –267. IEEE, May 2011.

[MTJ10] Per Håkon Meland, Inger Anne Tøndel, and Jostein Jensen. Idea: Reusability of Threat Models - Two Approaches with an Experimental Evaluation. In *Proceedings of the 2nd International Symposium on Engineering Secure Software and Systems*, volume 5965 of *LNCS*, pages 114–122, Pisa, Italy, February 2010. Springer.

[MY11] Ikuya Morikawa and Yuji Yamaoka. Threat Tree Templates to Ease Difficulties in Threat Modeling. In *Proceedings of the 14th International Conference on Network-Based Information Systems*, pages 673–678. IEEE, September 2011.

[Nea03] Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice Hall, Inc., 2003.

[NEJ+09] Steven Noel, Matthew Elder, Sushil Jajodia, Pramod Kalapa, Scott O'Hare, and Kenneth Prole. Advances in Topological Vulnerability Analysis. In *Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 124–129, Washington, DC, USA, 2009. IEEE Computer Society.

[Nie11] Jason R. Nielsen. Evaluating Information Assurance Control Effectiveness on An Air Force Supervisory Control And Data Acquisition (SCADA) System. Master's thesis, US Air Force Institute of Technology, March 2011.

[Nii10] Margus Niitsoo. Optimal adversary behavior for the serial model of financial attack trees. In *Proceedings of the 5th International Conference on Advances in Information and Computer Security*, pages 354–370. Springer, 2010.

[NJ04] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *ACM*, pages 109–118, Fairfax, VA, USA, October 2004.

[NJKJ05] Steven Noel, Michael Jacobs, Pramod Kalapa, and Sushil Jajodia. Multiple coordinated views for network attack graphs. In *Proceedings of the 2005 IEEE Workshop on Visualization for Computer Security*, pages 99–106, Minneapolis, USA, October 2005. John Wiley & Sons, Inc.

[NJL$^+$09] Per Närman, Pontus Johnson, Robert Lagerström, Ulrik Franke, and Mathias Ekstedt. Data Collection Prioritization for System Quality Analysis. *Electronic Notes in Theoretical Computer Science*, 233:29–42, March 2009.

[NJOJ03] Steven Noel, Sushil Jajodia, Brian O'Berry, and Michael Jacobs. Efficient Minimum-cost Network Hardening via Exploit Dependency Graphs. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 86–95, Las Vegas, NV, USA, December 2003. IEEE Computer Society.

[NJWS10] Steven Noel, Sushil Jajodia, Lingyu Wang, and Anoop Singhal. Measuring Security Risk of Networks Using Attack Graphs. *International Journal of Next-Generation Computing*, 1(1):135–147, 2010.

[NNL09] NNL Technology AB. InfoNode Docking Windows. http://www.infonode.net/index.html?idw, 1998–2009. Accessed July 2, 2013.

[NXYS08] Zhu Ning, Chen Xin-yuan, Zhang Yong-fu, and Xin Si-yuan. Design and Application of Penetration Attack Tree Model Oriented to Attack Resistance Test. In *Proceedings of the International Conference on Computer Science and Software Engineering*, pages 622–626. IEEE Computer Society, 2008.

[OBM06] Xinming Ou, Wayne F. Boyer, and Miles A. McQueen. A Scalable Approach to Attack Graph Generation. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 13th ACM conference on Computer and Communications Security*, pages 336–345. ACM, 2006.

[OGA05] Xinming Ou, Sudhakar Govindavajhala, and Andrew W. Appel. MulVAL: A logic-based network security analyzer. In *Proceedings of the 14th USENIX Security Symposium*, pages 113–128, 2005.

[Ohl02] Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, New York, NY, USA, 2002.

[Ope05] Alexander Opel. Design and Implementation of a Support Tool for Attack Trees. Master's thesis, Technische Universiteit Eindhoven, March 2005.

[Ops82] Robert J. Opsut. On the Computation of the Competition Number of a Graph. *SIAM Journal on Algebraic and Discrete Methods*, 3(4):420–428, 1982.

[Ora13] Oracle. Java Web Start. http://www.java.com/en/download/faq/java_webstart.xml, 1995–2013. Accessed July 2, 2013.

[OS09] Andreas L. Opdahl and Guttorm Sindre. Experimental comparison of attack trees and misuse cases for security threat identification. *Information and Software Technology*, 51(5):916–932, 2009.

[Ost11] Ryan T. Ostler. Defensive Cyber Battle Damage Assessment through Attack Methodology Modeling. Master's thesis, Air Force Institute of Technology, Department of Electrical and Computer Engineering, USA, 2011.

[OTT$^+$10] Poramate Ongsakorn, Kyle Turney, Mitchell A. Thornton, Suku Nair, Stephen A. Szygenda, and Theodore Manikas. Cyber threat trees for large system threat cataloging and analysis. In *Proceedings of the 4th Annual IEEE Systems Conference*, pages 610–615, April 2010.

[PA11] Mateusz Pawlik and Nikolaus Augsten. RTED: A Robust Algorithm for the Tree Edit Distance. *Very Large Data Base Endowment*, 5(4):334–345, 2011.

[PAB$^+$12] Huong Phan, George Avrunin, Matt Bishop, Lori A. Clarke, and Leon J. Osterweil. A Systematic Process-Model-Based Approach for Synthesizing Attacks and Evaluating Them. In *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*, pages 1–16, Berkeley, CA, USA, 2012. USENIX Association.

[PC10] Ludovic Piètre-Cambacédès. *Des relations entre sûreté et sécurité*. PhD thesis, Télécom ParisTech, 2010.

[PCB09] Ludovic Piètre-Cambacédès and Marc Bouissou. The promising potential of the BDMP formalism for security modeling. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Supplemental Volume*, Estoril, Portugal, June 2009. IEEE.

[PCB10a] Ludovic Piètre-Cambacédès and Marc Bouissou. Attack and Defense Modeling with BDMP. In Igor Kotenko and Victor Skormin, editors, *Proceedings of the 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security*, volume 6258 of *LNCS*, pages 86–101. Springer, 2010.

[PCB10b] Ludovic Piètre-Cambacédès and Marc Bouissou. Beyond attack trees: dynamic security modeling with Boolean logic Driven Markov Processes (BDMP). In *Proceedings of the 8th European Dependable Computing Conference*, pages 199–208, Valencia, Spain, April 2010. IEEE Computer Society.

[PCB10c] Ludovic Piètre-Cambacédès and Marc Bouissou. Modeling safety and security interdepedencies with BDMP (Boolean logic Driven Markov Processes). In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, pages 2852–2861, Istanbul, Turkey, October 2010. IEEE.

[PCB13] Ludovic Piètre-Cambacédès and Marc Bouissou. Cross-fertilization between safety and security engineering. *Reliability Engineering & System Safety*, 110:110–126, February 2013.

[PCDB11] Ludovic Piètre-Cambacédès, Yann Deflesselle, and Marc Bouissou. Security modeling with BDMP: from theory to implementation. In *Proceedings of the 6th IEEE International Conference on Network and Information Systems Security*, pages 1–8, La Rochelle, France, May 2011. IEEE.

[PDR12] Nayot Poolsappasit, Rinku Dewri, and Indrajit Ray. Dynamic Security Risk Management Using Bayesian Attack Graphs. *IEEE Transactions on Dependable and Secure Computing*, 9(1):61–74, Jan–Feb 2012.

[Pea86] Judea Pearl. Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241–288, 1986.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[PGR08] Sandip C. Patel, James H. Graham, and Patricia A. S. Ralston. Quantitatively assessing the vulnerability of critical information systems: A new method for evaluating security enhancements. *International Journal of Information Management*, 28(6):483–491, December 2008.

[PJM08] Holger Peine, Marek Jawurek, and Stefan Mandel. Security Goal Indicator Trees: A Model of Software Features that Supports Efficient Security Inspection. In *Proceedings of the 2008 11th IEEE High Assurance Systems Engineering Symposium*, pages 9–18, Washington, DC, USA, 2008. IEEE Computer Society.

[PK11] Marc Pouly and Jürg Kohlas. *Generic Inference: A Unifying Theory for Automated Reasoning.* John Wiley & Sons, Inc., 2011.

[Pla93] David A. Plaisted. *Equational Reasoning and Term Rewriting Systems*, pages 274–364. Oxford University Press, Inc., New York, NY, USA, 1993.

[PLC$^+$08] Gee-Yong Park, Cheol Kwon Lee, Jong Gyun Choi, Dong Hoon Choi, Young Jun Lee, and Kee-Choon Kwon. Cyber Security Analysis by Attack Trees for a Reactor Protection System. In *Proceedings of the Korean Nuclear Society, Fall Meeting*, Pyeong Chang, Korea, October 2008.

[PML10] Srdjan Pudar, Govindarasu Manimaran, and Chen-Ching Liu. PENET: a practical method and tool for integrated modeling of security attacks and countermeasures. *Computers & Security*, 28(8):754–771, May 2010.

[Pos12] Simona Posea. Renewal Periods for Cryptographic Keys. Master's thesis, Eindhoven University of Technology, Department of Mathematics and Computer Science, Eindhoven, The Netherlands, August 2012.

[Pot04] Michael D. Potter. *Set Theory and its Philosophy: a Critical Introduction.* Oxford University Press, Inc., 2004.

[Pou08] Marc Pouly. *A Generic Framework for Local Computation.* PhD thesis, Department of Informatics, University of Fribourg, 2008.

[PR07] Nayot Poolsapassit and Indrajit Ray. Investigating Computer Attacks Using Attack Trees. In *Proceedings of the IFIP International Conference on Digital Forensics*, volume 242 of *IFIP*, pages 331–343. Springer, 2007.

[PS98] Cynthia Phillips and Laura Painton Swiler. A Gaph-Based System for Network-Vulnerability Analysis. In *Proceedings of the 1998 New Security Paradigms Workshop*, pages 71–79, Charlottesville, VA, USA, September 1998. ACM.

[Pum99] David Pumfrey. *The Principled Design of Computer System Safety Analyses*. PhD thesis, Department of Computer Science, University of York, York, UK, September 1999.

[QL04] Xinzhou Qin and Wenke Lee. Attack plan recognition and prediction using causal networks. In *Proceedings of the 20th Annual Computer Security Applications Conference*, pages 370–379. IEEE Computer Society, December 2004.

[Ras06] Eric Rasmusen. *Games and Information: An Introduction to Game Theory*. John Wiley & Sons, Inc., 2006.

[RHCM12] Guifré Ruiz, Elisa Heymann, Eduardo César, and Barton P. Miller. Automating Threat Modeling through the Software Development Life-Cycle. `http://research.cs.wisc.edu/mist/papers/Guifre-sep2012.pdf`, 2012. Accessed July 2, 2013.

[RKT10a] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. ACT: Attack Countermeasure Trees for Information Assurance Analysis. In *Proceedings of the INFOCOM IEEE Conference on Computer Communications Workshops*, pages 1–2, San Diego, CA, USA, March 2010. IEEE.

[RKT10b] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Cyber security analysis using attack countermeasure trees. In *Proceedings of the 6th Annual Workshop on Cyber Security and Information Intelligence Research*, pages 28:1–28:4, New York, NY, USA, 2010. ACM.

[RKT12a] 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.

[RKT12b] Arpan Roy, Dong Seong Kim, and Kishor S. Trivedi. Scalable optimal countermeasure selection using implicit enumeration on attack countermeasure trees. In Robert S. Swarz, Philip Koopman, and Michel Cukier, editors, *Proceedings of the 42nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 299–310, Boston, MA, USA, June 2012. IEEE.

[Rø06] Lillian Røstad. An extended misuse case notation: Including vulnerabilities and the insider threat. In *Proceedings of the 12th International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 33–43, Luxembourg, Luxembourg, June 2006.

[Roy10] Arpan Roy. Attack Countermeasure Trees: A Non-state-space Approach Towards Analyzing Security and Finding Optimal Countermeasure Sets. Master's thesis, Duke University, Department of Electrical and Computer Engineering, USA, 2010.

[RP05] Indrajit Ray and Nayot Poolsapassit. Using Attack Trees to Identify Malicious Attacks from Authorized Insiders. In Sabrina di Vimercati, Paul Syverson, and Dieter Gollmann, editors, *Proceedings of the 10th European*

Symposium on Research in Computer Security, volume 3679 of *LNCS*, pages 231–246. Springer, 2005.

[RSF+09] Martin Rehák, Eugen Staab, Volker Fusenig, Michal Pěchouček, Martin Grill, Jan Stiborek, Karel Bartoš, and Thomas Engel. Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems. In Engin Kirda, Somesh Jha, and Davide Balzarotti, editors, *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, volume 5758 of *LNCS*, pages 61–80. Springer, 2009.

[RSK+12] Andreas Reinhardt, Daniel Seither, André König, Ralf Steinmetz, and Matthias Hollick. Protecting IEEE 802.11s Wireless Mesh Networks Against Insider Attacks. In *Proceedings of the 37th IEEE Conference on Local Computer Networks*, pages 224–227. IEEE, 2012.

[RVOC08] Kamil Reddy, Hein S. Venter, Martin S. Olivier, and Iain Currie. Towards Privacy Taxonomy-Based Attack Tree Analysis for the Protection of Consumer Information Privacy. In Larry Korba, Steve Marsh, and Reihaneh Safavi-Naini, editors, *Proceedings of the 6th Annual Conference on Privacy, Security and Trust*, pages 56–64. IEEE, October 2008.

[RWT13] RWTH Aachen, Research Group Computer Science 2. Automated Program Verification Environment (AProVE). http://aprove.informatik.rwth-aachen.de/, 2001–2013. Accessed July 2, 2013.

[Sam11] K. C. Sameer. Attack Generation From System Models. Master's thesis, Technical University of Denmark, Denmark, 2011.

[Sch24] Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92(3):305–316, 1924.

[Sch99] Bruce Schneier. Attack Trees. *Dr. Dobb's Journal of Software Tools*, 24(12):21–29, 1999.

[Sch04a] Stuart Edward Schechter. *Computer Security Strength and Risk - A Quantitative Approach*. PhD thesis, Harvard University, Cambridge, MA, USA, May 2004.

[Sch04b] Bruce Schneier. *Secrets and lies*. John Wiley & Sons, Inc., Indianapolis, USA, 2004.

[Scu10] Marco Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 35(3):1–22, July 2010.

[SDHH98] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian Approach to Filtering Junk E-Mail. In *Proceedings of the 1998 AAAI Workshop on Learning for Text Categorization*, pages 55–62. AAAI Press, 1998.

[SDP08] Vineet Saini, Qiang Duan, and Vamsi Paruchuri. Threat Modeling Using Attack Trees. *Journal of Computing in Small Colleges*, 23(4):124–131, 2008.

[SEJ08] Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. Combining defense graphs and enterprise architecture models for security analysis. In *Proceedings of the 12th IEEE International Conference on Enterprise Distributed Object Computing*, pages 349–355, München, Germany, September 2008. IEEE Computer Society.

[SEJ09] Teodor Sommestad, Mathias Ekstedt, and Pontus Johnson. Cyber Security Risks Assessment with Bayesian Defense Graphs and Architectural Models. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences*, pages 941 941–950, Hawaii, USA, January 2009. IEEE.

[SEN09] Teodor Sommestad, Mathias Ekstedt, and Lars Nordström. Modeling security of power communication systems using defense graphs and influence diagrams. *IEEE Transactions on Power Delivery*, 24(4):1801–1808, October 2009.

[She92] Prakash P. Shenoy. Valuation-Based Systems: A Framework for Managing Uncertainty in Expert Systems. In Lotfi A. Zadeh and Janusz Kacprzyk, editors, *Fuzzy Logic for the Management of Uncertainty*, pages 83–104. John Wiley & Sons, Inc., 1992.

[She04] Oleg Sheyner. *Scenario Graphs and Attack Graphs*. PhD thesis, Carnegie Mellon University (CMU), Pittsburgh, PA, 2004.

[SHI10a] SHIELDS. FP7 project, grant agreement 215995. http://www.shields-project.eu/, 2008–2010. Accessed July 2, 2013.

[SHI10b] SHIELDS. FP7 project, grant agreement 215995, Detecting known security vulnerabilities from within design and development tools. http://www.shields-project.eu/, 2008–2010. Accessed July 2, 2013.

[SHI10c] SHIELDS. FP7 project, grant agreement 215995, GOAT. http://www.ida.liu.se/divisions/adit/security/goat/, 2008–2010. Accessed July 2, 2013.

[SHI10d] SHIELDS. FP7 project, grant agreement 215995, Final SHIELDS approach guide - Deliverable D1.4. http://www.shields-project.eu/files/docs/D1.4%20Final%20SHIELDS%20Approach%20Guide.pdf, 2010. Accessed July 2, 2013.

[SHJ+02] Oleg Sheyner, Joshua W. Haines, Somesh Jha, Richard Lippmann, and Jeannette M. Wing. Automated Generation and Analysis of Attack Graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273–284, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[Sin07] Guttorm Sindre. Mal-Activity Diagrams for Capturing Attacks on Business Processes. In *Proceedings of the 13th International Working Conference on Requirements Engineering: Foundation for Software Quality*, volume 4542 of *LNCS*, pages 355–366, Trondheim, Norway, June 2007. Springer.

[SL05] Guttorm Sindre and Andreas L.Opdahl. Eliciting security requirements with misuse cases. *Requirements Engineering*, 10(1):34–44, 2005.

[SMdO+12] Nahid Shahmehri, Amel Mammar, Edgardo Montes de Oca, David Byers, Ana Cavalli, Shanai Ardi, and Willy Jimenez. An advanced approach for modeling and detecting software vulnerabilities. *Information and Software Technology*, 54(9):997–1013, 2012.

[SO00] Guttorm Sindre and Andreas L. Opdahl. Eliciting Security Requirements by Misuse Cases. In *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems*, pages 120–131, Sydney, Australia, November 2000. IEEE Computer Society.

[SO01] Guttorm Sindre and Andreas L. Opdahl. Templates for misuse case description. In *Proceedings of the 7th International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 125–136, Interlaken, Switzerland, June 2001.

[SOB02] Guttorm Sindre, Andreas L. Opdahl, and Gøran F. Brevik. Generalization/specialization as a structuring mechanism for misuse cases. In *Proceedings of the 2nd Symposium on Requirements Engineering for Information Security*, Raleigh, NC, USA, October 2002.

[SPEC01] Laura Painton Swiler, Cynthia Phillips, David Ellis, and Stefan Chakerian. Computer-attack graph generation tool. In *Proceedings of the DARPA Information Survivability Conference and Exposition II*, volume 2, pages 307–321, Anaheim, CA, USA, June 2001. IEEE Computer Society.

[SS02] Jan Steffan and Markus Schumacher. Collaborative attack modeling. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 253–259, Madrid, Spain, March 2002. ACM.

[SS04] Frank Swiderski and Window Snyder. *Threat modeling*. Microsoft Press, Redmond, USA, 2004.

[SS12] Husam Suleiman and Davor Svetinovic. Evaluating the effectiveness of the security quality requirements engineering (SQUARE) method: a case study using smart grid advanced metering infrastructure. *Requirements Engineering*, :1–29, 2012.

[SSSW98] Chris Salter, O. Sami Saydjari, Bruce Schneier, and Jim Wallner. Toward a Secure System Engineering Methodolgy. In *Proceedings of the 1998 New Security Paradigms Workshop*, pages 2–10, Charlottesville, VA, USA, September 1998. ACM.

[SVD+02] Michael Stamatelatos, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick III, and Jan Railsback. Fault Tree Handbook with Aerospace Applications (NASA). `http://www.hq.nasa.gov/office/codeq/doctree/fthb.pdf`, 2002. Accessed July 2, 2013.

[SWX11] Michael Sanford, Daniel Woodraska, and Dianxiang Xu. Security Analysis of FileZilla Server Using Threat Models. In *Proceedings of the 23rd International Conference on Software Engineering and Knowledge Engineering*, pages 678–682. Knowledge Systems Institute Graduate School, 2011.

[TA10] Eedee Tanu and Johnnes Arreymbi. An examination of the security implications of the supervisory control and data acquisition (SCADA) system in a mobile networked environment: An augmented vulnerability tree approach. In *Proceedings of the 5th Annual Conference on Advances in Computing and Technology*, pages 228–242. University of East London, School of Computing, Information Technology and Engineering, 2010.

[TDL13] TDL. Trust in Digital Life research community. http://www.trustindigitallife.eu/, 2008–2013. Accessed July 2, 2013.

[TJR10] Inger Anne Tøndel, Jostein Jensen, and Lillian Røstad. Combining Misuse Cases with Attack Trees and Security Activity Models. In *Proceedings of the 5th International Conference on Availability, Reliability and Security*, pages 438–445, Los Alamitos, CA, USA, February 2010. IEEE Computer Society.

[TLFH01] Terry Tidwell, Ryan Larson, Kenneth Fitch, and John Hale. Modeling Internet Attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security*, pages 54–59, West Point, NY, USA, June 2001.

[TLM07] Chee-Wooi Ten, Chen-Ching Liu, and Govindarasu Manimaran. Vulnerability Assessment of Cybersecurity for SCADA Systems Using Attack Trees. In *Proceedings of the IEEE Power Engineering Society General Meeting*, pages 1–8, Tampa, FL, USA, June 2007. IEEE.

[TML10] Chee-Wooi Ten, Govindarasu Manimaran, and Chen-Ching Liu. Cybersecurity for Critical Infrastructures: Attack and Defense Modeling. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 40(4):853–865, July 2010.

[TRE16] TREsPASS. Technology-supported Risk Estimation by Predictive Assessment of Socio-technical Security, FP7 project, grant agreement 318003. http://www.trespass-project.eu/, 2012–2016. Accessed July 2, 2013.

[TS09] Kishor S. Trivedi and Robin Sahner. SHARPE at the age of twenty two. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):52–57, March 2009.

[US 88] US Department of Defense (DoD). Standard Practice For System Safety. Technical Report MIL-STD-882D, US Department of Defense (DoD), June 1988.

[US 10] US Nuclear Regulatory Commission (NRC). Cyber Security Programs For Nuclear Facilities. http://pbadupws.nrc.gov/docs/ML0903/ML090340159.pdf, 2010. Accessed July 2, 2013.

[VGRH81] William E. Vesely, Francine F. Goldberg, Norman H. Roberts, and David F. Haasl. Fault Tree Handbook. Technical Report NUREG-0492, US Regulatory Commission, 1981.

[VIK11] VIKING. FP7 project, grant agreement 225643. http://www.vikingproject.eu, 2008–2011. Accessed July 2, 2013.

[VJ03] Stilianos Vidalis and Andy Jones. Using vulnerability trees for decision making in threat assessment. Technical Report CS-03-02, School of Computing, University of Glamorgan, Pontypridd, Wales, UK, 2003.

[vL04] Axel van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering*, pages 148 – 157. IEEE Computer Society, May 2004.

[vLBLJ03] Axel van Lamsweerde, Simon Brohez, Renaud De Landtsheer, and David Janssens. From System Goals to Intruder Anti-Goals: Attack Generation and Resolution for Security Requirements Engineering. In *Proceedings of the 2nd International Workshop on Requirements for High Assurance Systems*, pages 49–56, 2003.

[vLL00] Axel van Lamsweerde and Emmanuel Letier. Handling Obstacles in Goal-Oriented Requirements Engineering. *IEEE Transactions on Software Engineering*, 26(10):978–1005, October 2000.

[Wal90] John Q. Walker II. A Node-positioning Algorithm for General Trees. *Software: Practice and Experience*, 20(7):685–705, July 1990.

[Wat61] H. A. Watson. *Launch Control Safety Study*, volume 1. Bell Labs, Murray Hill, NJ, 1961.

[Wei91] Jonathan D. Weiss. A system security engineering process. In *Proceedings of the 14th National Computer Security Conference*, pages 572–581, 1991.

[WFM$^+$03] Yu-Sung Wu, Bingrui Foo, Blake Matheny, Tyler Olsen, and Saurabh Bagchi. ADEPTS: Adaptive Intrusion Containment and Response using Attack Graphs in an E-commerce Environment. Technical Report 2003-33, Purdue University, School of Electrical and Computer Engineering, December 2003.

[WFM$^+$05] Yu-Sung Wu, Bingrui Foo, Yu-Chun Mao, Saurabh Bagchi, and Eugene Spafford. Automated Aaptive Intrusion Containment in Systems of Interacting Services. Technical Report Paper 68, Purdue University, School of Electrical and Computer Engineering, West Lafayette, IN, USA, 2005.

[WFMB03] Yu-Sung Wu, Bingrui Foo, Yongguo Mei, and Saurabh Bagchi. Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 234–244. IEEE Computer Society, 2003.

[WIL$^+$08] Lingyu Wang, Tania Islam, Tao Long, Anoop Singhal, and Sushil Jajodia. An Attack Graph-Based Probabilistic Security Metric. In *Proceedings of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 5094 of *LNCS*, pages 283–296, London, UK, July 2008. Springer.

[Win90] Jeannette M. Wing. A Specifier's Introduction to Formal Methods. *IEEE Computer*, 23(9):8–23, September 1990.

[WJ10] Jan Willemson and Aivo Jürgenson. Serial Model for Attack Tree Computations. In D. Lee and S. Hong, editors, *Proceedings of the 13th International Conference Information Security and Cryptology*, volume 5984 of *LNCS*, pages 118–128. Springer, 2010.

[WLI07] Leevar Williams, Richard Lippmann, and Kyle W. Ingols. An interactive attack graph cascade and reachability display. In *Proceedings of the 2007 Workshop on Visualization for Computer Security*, pages 221–236, Sacramento, CA, USA, October 2007. Springer.

[WLR11] Mathew Warren, Shona Leitch, and Ian Rosewall. Attack vectors against social networking systems : the Facebook example . In *Proceedings of the 9th Australian Information Security Management Conference.* SECAU - Security Research Centre, 2011.

[WLZ06] Hui Wang, Shufen Liu, and Xinjia Zhang. An improved model of attack probability prediction system. *Wuhan University Journal of Natural Sciences*, 11(6):1498–1502, 2006.

[WNJ06] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, November 2006.

[WPWP10a] Jie Wang, Raphael C.-W. Phan, John N. Whitley, and David J. Parish. Augmented Attack Tree Modeling of Distributed Denial of Services and Tree Based Attack Detection Method. In *Proceedings of the 10th IEEE International Conference on Computer and Information Technology*, pages 1009–1014, Bradford, UK, June 2010. IEEE Computer Society.

[WPWP10b] Jie Wang, Raphael C.-W. Phan, John N. Whitley, and David J. Parish. Augmented attack tree modeling of SQL injection attacks. In *Proceedings of the 2nd IEEE International Conference on Information Management and Engineering*, volume 6, pages 182–186, Chengdu, China, April 2010. IEEE.

[WPWP10c] Jie Wang, Raphael C.-W. Phan, John N. Whitley, and David J. Parish. Quality of detectability (QoD) and QoD-aware AAT-based attack detection. In *Proceedings of the 2010 International Conference for Internet Technology and Secured Transactions*, pages 131–136, London, UK, November 2010. IEEE.

[WPWP11] John N. Whitley, Raphael C.-W. Phan, Jie Wang, and David J. Parish. Attribution of attack trees. *Computers & Electrical Engineering*, 37(4):624–628, 2011.

[WSJ07a] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Measuring the Overall Security of Network Configurations Using Attack Graphs. In Steve Barker and Gail-Joon Ahn, editors, *Proceedings of the 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4602 of *LNCS*, pages 98–112. Springer, 2007.

[WSJ07b] Lingyu Wang, Anoop Singhal, and Sushil Jajodia. Toward measuring network security using attack graphs. In *Proceedings of the 2007 ACM Workshop on Quality of Protection*, pages 49–54, New York, NY, USA, 2007. ACM.

[WW11] Lv Wen-ping and Li Wei-min. Space Based Information System Security Risk Evaluation Based on Improved Attack Trees. In *Proceedings of the 3rd International Conference on Multimedia Information Networking and Security*, pages 480–483. IEEE, November 2011.

[WWPP11] Jie Wang, John N. Whitley, Raphael C.-W. Phan, and David J. Parish. Unified Parametrizable Attack Tree. *International Journal for Information Security Research*, 1(1):20–26, 2011.

[WYSJ06] Lingyu Wang, Chao Yao, Anoop Singhal, and Sushil Jajodia. Interactive Analysis of Attack Graphs Using Relational Queries. In Ernesto Damiani and Peng Liu, editors, *Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, volume 4127 of *LNCS*, pages 119–132. Springer, 2006.

[XLO$^+$10] Peng Xie, Jason H. Li, Xinming Ou, Peng Liu, and Renato Levy. Using Bayesian networks for cyber security analysis. In *Proceedings of the 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 211–220. IEEE, Jun–Jul 2010.

[XN06] Dianxiang Xu and Kendall E. Nygard. Threat-driven modeling and verification of secure software using aspect-oriented Petri nets. *IEEE Transactions on Software Engineering*, 32(4):265–278, 2006.

[xp12] The xine project. xine multimedia engine. http://www.xine-project.org/home, 2002–2012. Accessed July 2, 2013.

[Yag06] Ronald R. Yager. OWA trees and their role in security modeling using attack trees. *Information Sciences*, 176(20):2933–2959, 2006.

[Zad78] Lotfi A. Zadeh. Fuzzy Sets as a Basis for a Theory of Possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

[Zan91] Hans Zantema. Termination of term rewriting, from many-sorted to one-sorted. Technical Report RUU-CS-91-18, Department of Information and Computing Sciences, Utrecht University, 1991.

[Zan00] Hans Zantema. Termination of Term Rewriting. Technical Report UU-CS-2000-04, Department of Information and Computing Sciences, Utrecht University, 2000.

[ZF11] Anita N. Zakrzewska and Erik M. Ferragut. Modeling cyber conflicts using an extended Petri Net formalism. In *Proceedings of the 2011 IEEE Symposium on Computational Intelligence in Cyber Security*, pages 60–67. IEEE, April 2011.

[ZKSY09] Saman A. Zonouz, Himanshu Khurana, William H. Sanders, and Timothy M. Yardley. RRE: A game-theoretic intrusion Response and Recovery Engine. In *Proceedings of the 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 439–448. IEEE, July 2009.

[Zon11] Saman Aliari Zonouz. *Game-theoretic Intrusion Response and Recovery.* PhD thesis, University of Illinois at Urbana-Champaign, USA, 2011.

[ZSR$^+$11] Saman Aliari Zonouz, Aashish Sharma, HariGovind V. Ramasamy, Zbigniew T. Kalbarczyk, Birgit Pfitzmann, Kevin McAuliffe, Ravishankar K. Iyer, William H. Sanders, and Eric Cope. Managing business health in the presence of malicious attacks. In *Proceedings of the 41st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops*, pages 9–14. IEEE Computer Society, June 2011.

[ZY12] Chengli Zhao and Zhiheng Yu. Quantitative Analysis of Survivability Based on Intrusion Scenarios. *Advances in Electronic Engineering, Communication and Management Vol. 2*, 140:701–705, 2012.

Author's Publications

[10KMMS] Barbara Kordy, Sjouke Mauw, Matthijs Melissen, and Patrick Schweitzer. Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent. In Tansu Alpcan, Levente Buttyán, and John S. Baras, editors, *Proceedings of the 1st Conference on Decision and Game Theory for Security*, volume 6442 of *LNCS*, pages 245–256. Springer, 2010.

[10KMMSTec] Barbara Kordy, Sjouke Mauw, Matthijs Melissen, and Patrick Schweitzer. Attack–Defense Trees and Two-Player Binary Zero-Sum Extensive Form Games Are Equivalent – Technical report with proofs. *Computing Research Repository*, abs/1006.2732:1–15, 2010. http://arxiv.org/abs/1006.2732.

[10KMRS] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Foundations of Attack–Defense Trees. In Pierpaolo Degano, Sandro Etalle, and Joshua D. Guttman, editors, *Proceedings of the 7th International Workshop on Formal Aspects of Security and Trust*, volume 6561 of *LNCS*, pages 80–95. Springer, 2010.

[10KS] Frank C. Krysiak and Patrick Schweitzer. The optimal size of a permit market. *Journal of Environmental Economics and Management*, 60(2):133 – 143, 2010.

[10SS] Pascal Schweitzer and Patrick Schweitzer. Connecting face hitting sets in planar graphs. *Information Processing Letters*, 111(1):11–15, 2010.

[11KPS] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational Aspects of Attack–Defense Trees. In *Proceedings of the 19th International Conference Security & Intelligent Information Systems*, volume 7053 of *LNCS*, pages 103–116. Springer, 2011.

[12BKMS] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. Attribute Decoration of Attack–Defense Trees. *International Journal of Secure Software Engineering*, 3(2):1–35, 2012.

[12KMRS] Barbara Kordy, Sjouke Mauw, Saša Radomirović, and Patrick Schweitzer. Attack–Defense Trees. *Journal of Logic and Computation*, 2012. To appear. http://logcom.oxfordjournals.org/content/early/2012/06/21/logcom.exs029.

[12KMS] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative Questions on Attack–Defense Trees. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Proceedings of the 15th Annual International*

Conference on Information Security and Cryptology, volume 7839 of *LNCS*, pages 49–64. Springer, 2012.

[12KMSTec] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative Questions on Attack–Defense Trees. *Computing Research Repository*, abs/1210.8092:1–17, 2012. http://arxiv.org/abs/1210.8092.

[12KSADTLib] Barbara Kordy and Patrick Schweitzer. The ADTree Library. http://satoss.uni.lu/projects/atrees/library.php, 2012. Accessed July 1, 2013.

[12KSADTMan] Piotr Kordy and Patrick Schweitzer. The ADTool Manual. http://satoss.uni.lu/software/adtool/manual.pdf, 2012. Accessed July 1, 2013.

[12MS] Tim Muller and Patrick Schweitzer. A Formal Derivation of Composite Trust. In Joaquín García-Alfaro, Frédéric Cuppens, Nora Cuppens-Boulahia, Ali Miri, and Nadia Tawbi, editors, *Proceedings of the 5th International Symposium on Foundations & Practice of Security*, volume 7743 of *LNCS*, pages 132–148. Springer, 2012.

[12SADTSty] Patrick Schweitzer. The ADTree Style File. http://satoss.uni.lu/projects/atrees/library.php, 2012. Accessed July 1, 2013.

[13KKMS] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: Security Analysis with Attack–Defense Trees (Tool Demonstration Paper). In *Proceedings of the 10th International Conference on Quantitative Evaluation of SysTems*, volume 8054 of *LNCS*, pages 173–176. Springer, 2013.

[13KKMSTec] Barbara Kordy, Piotr Kordy, Sjouke Mauw, and Patrick Schweitzer. ADTool: Security Analysis with Attack–Defense Trees (Tool Demonstration Paper — Extended Version). *Computing Research Repository*, abs/1305.6829:1–10, 2013. http://arxiv.org/abs/1305.6829.

[13KPSFor] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. DAG-Based Attack and Defense Modeling: Don't Miss the Forest for the Attack Trees. http://arxiv.org/abs/1303.7397, 2013. Submitted to Computer Science Review.

[13KPSPro] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. A Probabilistic Framework for Security Scenarios with Dependent Actions, 2013. Submitted to the 35th IEEE Symposium on Security and Privacy.

[13MSS] Brendan McKay, Pascal Schweitzer, and Patrick Schweitzer. Competition Numbers, Quasi-Line Graphs and Holes. *SIAM Journal on Discrete Mathematics*, 2013. To appear.

[13MS] Tim Muller and Patrick Schweitzer. On Beta Models with Trust Chains. In *Proceedings of the 7th IFIP WG 11.11 International Conference on Trust Management*, volume 401 of *IFIP Advances in Information and Communication Technology*, pages 49–65. Springer, 2013.

Index

Page references in **bold** indicate pages containing the definition or an extensive explanation of the keywords.