Formal modeling and quantitative analysis of security using attack- defense trees

Wojciech Widel

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

INSA RENNES

THESE DE DOCTORAT DE

L'INSA RENNES
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématique et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Wojciech WIDEŁ

Formal modeling and quantitative analysis of security using attack-defense trees

Thèse présentée et soutenue à Rennes, le 3 décembre, 2019
Unité de recherche : Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA), UMR 6074
Thèse N° : 19ISAR 28 / D19 - 28

Rapporteurs avant soutenance :

Mathias Ekstedt	Professeur, KTH Royal Institute of Technology
Sjouke Mauw	Professeur, University of Luxembourg

Composition du jury :

Président :	Sophie Pinchinat	Professeur, Université de Rennes 1
Examinateurs :	Gildas Avoine	Professeur, INSA Rennes
	Mathias Ekstedt	Professeur, KTH Royal Institute of Technology
	Barbara Fila	Maître de conférences, INSA Rennes
	Heiko Mantel	Professeur, Technische Universität Darmstadt
	Sjouke Mauw	Professeur, University of Luxembourg
	Saša Radomirović	Senior Lecturer, University of Dundee
Directeur de thèse :	Gildas Avoine	Professeur, INSA Rennes

Acknowledgements

I would like to thank my supervisor Barbara Fila, especially for the help that I could count on whenever faced with problems related to everyday life. I am also grateful for the guidance provided by my official supervisor, Gildas Avoine.

I am thankful to all members of the Embedded Security and Cryptography team at IRISA Rennes who made an effort of speaking English with me, even if they preferred French. This includes my officemates. Of them, additional thanks go to Florent, for showing me a way of getting deeper into cybersecurity, and for guiding me on that way whenever requested.

I am very grateful to all my friends in Rennes for making my life there good. For the time spent together, many thanks go to Adela, Gautier, Katharina, Sam and Lucy, Solène and Vaishnavi.

The support provided continously by Ola throughout these years was by far the most important and precious thing for me. Thanks for being there! Together with my parents and brothers, all of their help and their faith in my capabilities, you made it possible for me to complete this thesis.

Contents

List of publications

Some of the results presented in this thesis have been already published, in the following articles.

[WAFP19] Wojciech Wideł, Maxime Audinot, Barbara Fila and Sophie Pinchinat. Beyond 2014: Formal methods for attack tree-based security modeling. In *ACM Computing Surveys*, 52(4), pages 75:1–75:36, 2019.

[FW19b] Barbara Fila and Wojciech Wideł. Efficient Attack–Defense Tree Analysis using Pareto Attribute Domains. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019*, pages 200–2015. IEEE, 2019.

[FW19a] Barbara Fila and Wojciech Wideł. Attack–defense trees for abusing optical power meters: A case study and the `OSEAD` tool experience report. In *Graphical Models for Security — 6th International Workshop, GraMSec 2019, Proceedings*, volume 11720 of *Lecture Notes in Computer Science*, pages 95–125. Springer, 2019.

[KW18] Barbara Kordy and Wojciech Wideł. On quantitative analysis of attack–defense trees with repeated labels. In *Principles of Security and Trust — 7th International Conference, POST 2018, Proceedings*, volume 10804 of *Lecture Notes in Computer Science*, pages 325–346. Springer, 2018.

[KW17] Barbara Kordy and Wojciech Wideł. How well can I secure my system? In *Integrated Formal Methods — 13th International Conference, IFM 2017, Proceedings*, volume 10510 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2017.

Résumé

Contexte

Le seul système totalement sécurisé est le système vide, qui n'offre aucune fonctionnalité. Tout autre système offrant un réel service, qu'il s'agisse d'un guichet automatique, d'un serveur Web ou d'une centrale nucléaire, sera toujours vulnérable aux attaques. Ces attaques peuvent viser la disponibilité du système (par exemple, attaques par déni de service), l'intégrité des données liées au système (par exemple, modification des dossiers financiers d'un client bancaire) ou la confidentialité des renseignements liés au système (par exemple, accès au dossier médical du patient). Pour atteindre leurs objectifs malveillants, les attaquants, externes ou internes au système, peuvent employer diverses approches, notamment des moyens numériques, des attaques physiques et des techniques d'ingénierie sociale reposant sur la manipulation psychologique. Tous ces aspects doivent être pris en compte lors de l'analyse de la sécurité d'un système. La sécurisation d'un système contre les attaques est d'autant plus difficile que la sécurité parfaite nécessite des ressources illimitées, en termes de moyens financiers et de temps, qui ne sont malheureusement jamais disponibles. C'est dans ce contexte que l'évaluation des risques joue un rôle majeur.

Le risque peut être défini de manière informelle comme la probabilité d'un incident et ses conséquences pour un actif [RSS15]. Pour un actif qui serait la disponibilité d'un service sur un serveur Web, un incident pourrait être une attaque par déni de service : la conséquence serait que le serveur Web devient indisponible pour ses utilisateurs légitimes. Pour faire face aux risques liés à un système, un processus d'identification, d'analyse et de gestion proactive des risques, appelé gestion des risques, est mené. L'évaluation des risques fait partie du processus de gestion des risques. Bien qu'il n'existe pas de définition unique de l'évaluation des risques, elle comprend trois phases selon la norme ISO 3100 [ISO18] : *l'identification des risques*, *l'analyse des risques* et *l'évaluation des risques*.

L'objectif de la phase d'identification des risques est, sans surprise, d'obtenir une liste exhaustive des risques possibles liés au système et à ses actifs. Il s'agit d'identifier les vulnérabilités présentes dans le système, les manières dont elles peuvent être exploitées pour provoquer un incident (*menaces*), et les causes possibles de leur exploitation (*sources de menaces*). Une fois les risques identifiés, l'analyse des risques peut commencer. Son but est d'estimer la probabilité et de déterminer les conséquences des menaces identifiées.

De nombreux facteurs influent sur la probabilité d'une menace.

Quand la source de la menace est un attaquant, selon leur point de vue, «l'objectif principal est de poursuivre des attaques plus faciles et moins coûteuses à mener et qui ont une plus grande probabilité de réussir plutôt que d'échouer», selon le CISO AppSec Guide de l'OWASP[1]. C'est-à-dire que, dans ce cas particulier, lors de l'analyse du risque et de l'évaluation de la probabilité d'une attaque, il faudrait au moins tenir compte de la difficulté de l'attaque, du coût de son exécution et de la probabilité de son succès, exprimés de manière qualitative ou quantitative. Une fois les probabilités et les conséquences des menaces estimées, les risques sont évalués : ils peuvent être comparés à certains critères d'acceptation des risques supposés, certains risques peuvent être regroupés en un seul risque et leur tolérabilité peut être évaluée.

De nombreuses techniques peuvent être employées pour mener à bien le processus d'évaluation des risques et divers outils peuvent rendre la tâche plus facile à accomplir. Lors de l'évaluation des risques, on peut, par exemple, suivre les directives d'évaluation des risques (en constante évolution) créées par des organismes officiels pour traiter des systèmes spécifiques, ou utiliser des méthodologies générales, telles que EBIOS, CRAM, ITSG-04 ou MAGERIT, pour n'en citer que quelques-unes. Ces quatre méthodologies sont décrites et comparées dans le rapport de l'OTAN intitulé *Improving Common Security Risk Analysis* [TR-08]. Les auteurs du rapport affirment également que, dans certains cas, les méthodes basées sur les arbres d'attaque offrent une alternative viable à des méthodes aussi complexes.

Les arbres d'attaque peuvent être utilisés à la fois pour identifier et analyser les risques. Leur fonction fondamentale consiste à traduire les objectifs de l'attaquant en actions simples menant à la réalisation de ces objectifs. Le processus même de leur création pourrait fournir des informations précieuses au cours des délibérations sur l'évaluation des risques et permettre de mieux comprendre le système étudié et les menaces auxquelles il est confronté. Mais leurs applications potentielles ne se limitent pas à ces fonctions. Si l'on peut attribuer aux actions susmentionnées des informations quantitatives ou qualitatives, reflétant par exemple l'investissement monétaire nécessaire à leur exécution, les arbres peuvent être analysés à l'aide de méthodes bien documentées, fournissant des résultats utiles pour évaluer la probabilité d'attaques particulières. Il est également possible d'inclure des contre-mesures dans les modèles arborescents d'attaque, ce qui rend ces modèles étendus utiles dans la phase de gestion des risques qui suit l'évaluation des risques, à savoir le traitement des risques. Le traitement des risques consiste en des activités visant à déterminer et à sélectionner les moyens de faire face aux risques, y compris, entre autres, l'évitement des risques, la réduction des risques, le transfert des risques.

[1]Disponible à l'adresse `https://www.owasp.org/index.php/CISO_AppSec_Guide:_Criteria_for_Managing_Application_Security_Risks`.

Introduction informelle aux arbres d'attaque et aux arbres d'attaque et de défense

Arbres d'attaque Les arbres d'attaque [Sch99] sont un formalisme graphique bien établi et couramment utilisé pour la modélisation de la sécurité. Inspirés des arbres de défaillance [HRVG81], utilisés dans l'analyse de fiabilité des systèmes, et des arbres logiques de menaces [Wei91], ils fournissent une représentation lisible et structurée des attaques possibles contre un système à protéger. Leur structure hiérarchique révèle les caractéristiques communes des attaques et permet une évaluation quantitative des éléments, mettant ainsi en évidence les vulnérabilités les plus graves sur lesquelles il faut se concentrer lors de la mise en œuvre de contre-mesures. Formellement, les arbres d'attaque sont des arbres avec une racine et des nœuds étiquetés. Les étiquettes des nœuds représentent les buts de l'attaquant, avec l'étiquette de la racine correspondant au but principal de l'attaquant. Cet objectif, souvent de haut niveau et abstrait, est alors récursivement raffiné en sous-objectifs représentés par les étiquettes des nœuds restants. Le modèle de base des arbres d'attaque admet deux types de raffinements : le raffinement conjonctif `AND` et le raffinement disjonctif `OR`. Pour atteindre l'objectif d'un nœud `AND`, il faut atteindre les sous-objectifs de tous ses enfants, alors que pour atteindre l'objectif d'un nœud `OR` il suffit d'atteindre au moins un but de ses nœuds enfants. Un autre raffinement souvent considéré est le raffinement conjonctif séquentiel (`SAND`). De même que dans le cas du raffinement conjonctif, l'atteinte d'un but d'un nœud `SAND` nécessite l'atteinte des sous-buts de tous ses enfants, mais dans un ordre spécifique.

Arbres d'attaque et de défense Les arbres d'attaque et de défense [KMRS14] améliorent la puissance expressive des arbres d'attaque en permettant de représenter explicitement les objectifs du défenseur dans le modèle. Dans un scénario représenté par un arbre d'attaque et de défense, le but d'un acteur (attaquant ou défenseur) peut être contré par le but de l'autre acteur. C'est-à-dire, chacun des nœuds, y compris un nœud non raffiné, peut avoir parmi ses enfants un nœud de l'autre acteur, qui représente un moyen de contrer le but du nœud parent. L'objectif d'un nœud ayant une contre-mesure parmi ses enfants est atteinte si les conditions issues du raffinement du nœud sont atteintes (dans le cas où le nœud est raffiné) et si l'objectif du nœud de contre-mesure n'est pas atteint. On peut noter qu'exiger que chaque nœud ait au plus une contre-mesure parmi ses enfants n'est pas restrictive : s'il est possible de décrire l'objectif d'un nœud de plusieurs possibilités, elles peuvent toutes être regroupées sous un nœud parent commun raffiné de façon disjonctive, qui devient alors l'unique contre-mesure du nœud.

Selon la terminologie introduite dans [KMRS14], l'acteur principal d'un arbre d'attaque et de défense est appelé le *proponent* et l'autre acteur est l'*opponent*. L'objectif du proponent est d'atteindre l'objectif fondamental, alors que l'opponent tente de le rendre impossible. Les étiquettes des nœuds qui ne sont pas raffinées sont appelées actions de

base. Elles représentent les actions que les acteurs exécutent pour atteindre les objectifs des nœuds raffinés.

Notation graphique Lors de la représentation graphique des arbres d'attaque et de défense, nous utilisons les conventions standard. Les nœuds de l'attaquant sont représentés par des ellipses rouges et les nœuds du défenseur par des rectangles verts. Les nœuds `AND` diffèrent des nœuds `OR` en ce que les bords qui les relient à leurs enfants sont reliés par un arc. Les contre-mesures sont attachées aux nœuds qu'elles contrent par une ligne pointillée. Les nœuds de contre-mesure et les fils d'un nœud sont représentés sous le nœud.

Exemple fil rouge Un arbre d'attaque et de défense utilisé comme exemple récurrent tout au long de cette thèse est illustré en figure 1. Le scénario modélisé avec cet arbre est expliqué dans l'exemple 1.

Exemple 1. *Dans le scénario représenté par l'arbre d'attaque et de défense de la figure 1, le proponent est l'attaquant et l'opponent est le défenseur. L'attaquant veut voler l'argent du compte du défenseur. Pour atteindre cet objectif, l'agresseur peut utiliser des moyens physiques, c'est-à-dire apprendre le NIP de la victime, voler sa carte, puis retirer de l'argent à un guichet automatique. Pour apprendre le NIP, l'agresseur pourrait forcer la victime à le révéler ou l'intercepter lorsqu'elle entre le NIP. La victime pourrait prévenir ce dernier en recouvrant le clavier de sa main. Cependant, la couverture du clavier échoue si l'attaquant surveille le clavier avec une micro-caméra cachée installée à un endroit approprié. Au lieu d'attaquer d'un point de vue physique, l'attaquant peut voler de l'argent en exploitant les services bancaires en ligne. Pour ce faire, il pourrait apprendre le nom d'utilisateur et le mot de passe de la victime. Ces deux objectifs peuvent être atteints en créant un faux site Web de banque et en utilisant des techniques d'hameçonnage pour amener le titulaire du compte à entrer ses informations d'identification. Il pourrait aussi essayer de deviner quel est le mot de passe et le nom d'utilisateur. L'utilisation d'un mot de passe solide permettrait au titulaire du compte de contrer une telle attaque par devinette. Une fois que l'attaquant obtient les identifiants, il peut les utiliser pour se connecter à la banque en ligne et exécuter un transfert. Pour prévenir une telle attaque, les dispositions de transfert pourraient être en outre sécurisés par une authentification bifactorielle à l'aide de SMS. Cette mesure de sécurité pourrait être contrée en volant le téléphone de la victime.*

L'arbre de la figure 1 est un exemple jouet, pratique pour illustrer les notions introduites plus loin dans la thèse. Dans le chapitre 7, nous construisons un arbre d'attaque et de défense plus grand et réaliste, d'après un arbre d'attaque analysé par le Département de l'énergie des États-Unis dans [Nat15].

Actions de base répétées Il n'est pas rare que dans un arbre d'attaque et de défense certains nœuds portent la même étiquette. Dans un tel cas, il y a deux façons de les interpréter.

- Les nœuds représentent la même instance unique du but - par exemple, les deux nœuds étiquetés avec l'action de phishing dans l'arbre de la figure 1 peuvent se référer à la même action, comme décrit dans l'exemple 1. En d'autres termes, le simple fait de créer un faux site Web de banque et d'inciter la victime à entrer ses informations d'identification permet d'obtenir à la fois un nom d'utilisateur et un mot de passe.
- Chacun des nœuds est traité comme une instance distincte du but. Par exemple, dans le scénario modélisé à l'aide de l'arbre de la figure 1, on pourrait utiliser deux techniques d'hameçonnage différentes, chacune conçue spécialement pour atteindre exactement l'un des objectifs suivants : obtenir un nom d'utilisateur et un mot de passe. Dans un tel cas, alors que les deux nœuds pourraient encore être appelées hameçonnage, elles représenteraient des cas distincts d'une attaque d'hameçonnage.

La présence de différents nœuds étiquetés de la même manière est naturelle. Certains buts et actions peuvent contribuer à de multiples façons d'attaquer ou de défendre un système, et certains de ces moyens peuvent exiger qu'une action soit effectuée plusieurs fois. Il est facile de contrôler les étiquettes si un arbre est petit et s'il est construit manuellement, et de le construire avec une interprétation fixe des étiquettes répétées en tête. Des problèmes peuvent survenir si un arbre est le résultat d'une procédure automatique, ou s'il est, par exemple, composé d'arbres plus petits créés par différents analystes analysant des sous-scénarios du scénario, ou ayant des connaissances sur des sous-systèmes particuliers du système à l'étude.

Dans ce travail, nous suppons la première des deux manières d'interprétation données ci-dessus. Il y a au moins deux raisons à ce choix. Nous croyons que cette méthode correspond à la lecture intuitive des arbres d'attaque et de défense, c'est-à-dire que lorsqu'une personne reçoit un arbre, elle est plus susceptible de considérer que les étiquettes répétées représentent le même événement, et non des instances distinctes de celui-ci. De façon plus importante et plus formelle, cette interprétation est plus libérale que l'autre. Elle permet de modéliser à la fois des objectifs et des actions contribuant à des objectifs multiples, en utilisant le même label, tout en gardant la possibilité de modéliser différentes instances d'une même action ou d'un même but, en utilisant des étiquettes légèrement différentes. Si l'autre interprétation était utilisée, il serait impossible de modéliser la possibilité d'une seule action contribuant à des attaques multiples.

Selon [BK17], nous appelons une action de base qui sert d'étiquette pour au moins deux nœuds un clone ou une action basique clonée. Les nœuds représentant des instances distinctes de la même action ou objectif sont supposés avoir des étiquettes différentes. Dans ce réglage, il est pratique d'utiliser des graphes orientés acycliques, où les nœuds

portant la même étiquette sont fusionnés en un seul nœud, au lieu d'arbres. Une telle approche conduit à une meilleure lisibilité des modèles et peut être exploitée pour l'accélération des calculs effectués sur les arbres. Utiliser des graphes orientés acycliques au lieu d'arbres est une mesure standard dans le domaine de l'analyse des arbres de défaillance [RS15], où les sous-arbres enracinés dans les nœuds portant le même label sont appelés *sous-arbres partagés*, et les analogues des clones sont des *événements de base partagés*, prise aussi parfois dans le cas d'arbres d'attaque, par exemple dans [AHPS14]. Notre définition des arbres d'attaque et de défense basée sur des graphes acycliques orientés sera donnée au Chapitre 2. La représentation graphique de l'arbre de la figure 1 redessinée en graphe orienté acyclique est donnée à la figure 2. La seule différence entre les deux représentations est que dans ce dernier cas, les nœuds étiquetés avec l'action de phishing sont fusionnés en un seul nœud.

Questions de recherche et contributions

L'objectif principal des travaux de recherche dont cette thèse est issue est d'identifier et de lever les limites de l'utilité des arbres de défense contre les attaques dans le processus d'évaluation des risques. Parmi les limites que nous avons pu identifier, mentionnons les suivantes.

1. De nombreuses méthodes d'analyse des arbres d'attaque et de défense qui pourraient être utiles pour estimer la probabilité d'attaques sont soit développées dans l'hypothèse explicite que les arbres ne contiennent pas de clones, soit d'une manière qui les rend impropres aux arbres avec clones.

2. Les méthodes d'analyse axées sur un certain nombre de paramètres à la fois, par exemple pour déterminer les attaques qui sont optimales en termes de coût et de probabilité de réussite, ne sont généralement pas efficaces dans le cas de grands modèles et/ou peuvent être appliquées à un nombre limité de paramètres.

3. Les approches pour une sélection optimale (dans un sens bien défini) des contre-mesures en des scénarios de sécurité modélisés à l'aide d'arbres d'attaque et de défense sont soit formulés en termes d'une analyse par simulation, c'est-à-dire qu'elles permettent de sélectionner des contre-mesures dans le cadre d'un comportement fixe de l'attaquant, ou bien elles ne peuvent être appliquées qu'aux arbres satisfaisant certaines restrictions structurelles.

4. L'accès des analystes aux derniers développements dans le domaine des arbres d'attaque et de défense est très limité. De nouvelles techniques d'analyse sont créées chaque année et il est difficile d'avoir une vue d'ensemble claire du domaine, même pour les chercheurs travaillant dans ce domaine. De plus, très peu d'outils

mettant en œuvre les techniques d'analyse les plus récentes sont accessibles et tous les outils existants ne sont pas maintenus.

Contributions Pour aborder la première de ces limitations, nous avons analysé l'une des méthodes fondamentales d'analyse des arbres de défense contre les attaques, à savoir la procédure ascendante pour calculer les paramètres liés aux attaques. Elle peut être utilisée, par exemple, pour obtenir efficacement des valeurs telles que le coût minimal ou la probabilité maximale de succès d'une attaque. Nous avons pu déterminer les causes du dysfonctionnement de la procédure ascendante en présence de clones. Cela nous a permis de développer des méthodes alternatives pour calculer ces paramètres et de construire des algorithmes efficaces pour déterminer les attaques optimales du point de vue de l'attaquant. Nous avons pu adapter ces nouvelles méthodes pour l'objectif de l'analyse multiparamétrique de scénarios de sécurité modélisés à l'aide d'arbres, c'est-à-dire en abordant dans une certaine mesure le deuxième des quatre points soulevés ci-dessus.

Nous avons également abordé le problème de l'exploitation de modèles d'arbres d'attaque et de défense pour une sélection optimale des contre-mesures dans les scénarios de sécurité. Nous avons développé une méthode pour extraire des modèles les comportements possibles d'un attaquant rationnel, ainsi que des moyens de contrer de tels comportements par le défenseur. Ces informations peuvent être utilisées comme données d'entrée pour des méthodes d'optimisation standard, permettant ainsi de déterminer, par exemple, un ensemble de contre-mesures dont la mise en œuvre correspond à un budget donné et maximise l'investissement nécessaire de l'attaquant pour atteindre son but. Enfin, nous nous sommes efforcés d'accroître l'accessibilité du grand public aux développements récents dans le domaine de l'analyse des arbres d'attaque. Tout d'abord, nous avons passé en revue les articles de recherche pertinents publiés au cours des années 2014-2018. Nous avons évalué les points forts et les points faibles des méthodologies présentées, étudié les relations entre elles et décrit nos conclusions. Deuxièmement, nous avons développé un support d'outil pour les méthodes d'analyse présentées dans cette thèse. L'outil `OSEAD` (Optimal Strategies Extractor for Attack-Defense Trees) est un logiciel facile à utiliser et disponible gratuitement qui vise à soutenir les analystes dans leur travail.

Le processus d'évaluation des risques est une tâche quelque peu délicate, dont les résultats ne sont généralement accessibles à personne d'autre que les parties intéressées. C'est peut-être la cause de l'impossibilité de trouver des modèles réalistes basés sur des arbres d'attaque. Pour valider les méthodes décrites dans cette thèse, nous avons donc créé un arbre d'attaque et de défense réaliste basé sur un scénario de sécurité considéré dans [Nat15]. Nous avons mené une étude de cas sur le scénario modélisé avec l'arbre, en utilisant certaines des méthodes décrites dans cette thèse. Nous espérons que le modèle lui-même pourra être utile à d'autres chercheurs comme banc d'essai pour leurs idées.

Structure de la thèse

Dans le chapitre 2, nous fournissons le contexte formel nécessaire à la compréhension complète des autres parties de la thèse.

Pour situer les résultats de nos recherches dans le contexte de l'analyse des arbres d'attaque, nous décrivons certains des travaux existants qui sont étroitement liés aux nôtres dans le chapitre 3, basé sur notre enquête [WAFP19].

Le problème de l'analyse quantitative de la sécurité à l'aide d'arbres de défense contre les attaques contenant des clones est étudié en profondeur au chapitre 4. Les fondements du cadre décrit dans ce chapitre ont été posés dans [KW18]. La plupart des résultats sont nouveaux et n'ont pas encore été préparés pour publication.

Le chapitre 5, basé sur [FW19b], est consacré à l'analyse multiparamétrique de la sécurité.

La sélection optimale des contre-mesures dans des scénarios modélisés avec des arbres est le point central du chapitre 6. Les idées qui sous-tendent l'approche décrite dans le chapitre, ainsi que certains résultats préliminaires, ont été présentés dans [KW17]. Le reste du chapitre porte sur des développements récents qui n'ont pas encore été publiés.

Enfin, au chapitre 7, nous décrivons l'outil `OSEAD` et l'utilisons pour réaliser une étude de cas d'un scénario de sécurité lié au secteur énergétique. L'étude a été publiée dans [FW19a].

Nous concluons au chapitre 8.

Chapter 1

Introduction

1.1 Context

The only system that is guaranteed to be fully secure is the empty system, which does not provide any functionality. Any other system offering an actual service, be it an automated teller machine, a web server or a nuclear power plant, will always be vulnerable to attacks. These attacks may target the system's availability (e.g., denial-of-service attacks), the integrity of system-related data (e.g., modification of financial records of a bank client), or the confidentiality of system-related information (e.g., gaining access to a patient's medical record). To achieve their malicious goals, attackers, who might be external to the system or insiders, can employ various approaches, including digital means, physical attacks, and social engineering techniques relying on psychological manipulation. All these aspects should be taken into account when analyzing security of a system. The task of securing a system against attacks is made even more difficult by the fact that perfect security requires unlimited resources, in terms of financial means and time, which are of course never available. This is where the *risk assessment* comes into play.

Risk can be informally defined as *the likelihood of an incident and its consequences for an asset* [RSS15]. For the asset being the availability of a web server services, an incident might be a denial-of-service attack, the consequence of which is the web server becoming unavailable to its intended users. To tackle the risks related to the system of interest, the process of risks identification, analysis and proactive management, called *risk management*, is conducted. Risk assessment is a part of the risk management process. While no single definition of risk assessment exists, according to the ISO 3100 standard [ISO18] it consists of three phases: *risk identification*, *risk analysis* and *risk evaluation*.

The goal of the risk identification phase is, not surprisingly, to obtain an exhaustive list of possible risks related to the system and its assets. It involves identifying vulnerabilities present in the system, the ways in which they can be exploited to cause an incident (*threats*), and the possible causes for their exploitation (*threat sources*). With the risks identified, the risk analysis can begin. Its aim is to estimate the likelihood and to determine the consequences of identified threats. Numerous factors impact the likeli-

hood of a threat. When the threat source is an attacker, then from their perspective "the main goal is to pursue attacks that are easier and cheaper to conduct and have the highest probability to succeed rather than otherwise," according to OWASP CISO AppSec Guide[1]. That is, in this particular case, when conducting a risk analysis and assessing the likelihood of an attack, one could take at least the attack's difficulty, the cost of its execution and the probability of its success into account, expressed either qualitatively or quantitatively. Once the likelihoods and the consequences of threats are estimated, the risks are evaluated: they might be compared against some assumed risk acceptance criteria, some of the risks can be aggregated into one risk, and their tolerability can be assessed.

Numerous techniques can be employed for conducting the risk assessment process and various tools can make the task easier to handle. When performing risk assessment, one can, for instance, follow (ever evolving) risk assessment guidelines created by official bodies for dealing with specific systems, or use general-purpose methodologies, such as EBIOS, CRAM, ITSG-04 or MAGERIT, to name a few. These four methodologies are described and compared in the NATO's *Improving Common Security Risk Analysis* report [TR-08]. The authors of the report state also that in some cases methods based on *attack trees* offer a viable alternative to such complex methodologies.

Attack trees can be used for both identifying and analyzing risks. Their fundamental function lies in translating the attacker goals into simple actions leading to realization of these goals. The very process of their creation might provide valuable insights during the risk assessment deliberations and offer a better understanding of the system under consideration and the threats that the system is facing. But their potential applications are not limited to these functions. If the above mentioned actions can be assigned quantitative or qualitative information, reflecting for instance the monetary investment necessary for their execution, trees can be analyzed using well-studied methods, providing results helpful in assessing the likelihood of particular attacks. It is also possible to include countermeasures against attacks in the attack tree-based models, which makes such extended models useful in the phase of risk management that follows the risk assessment, namely, *risk treatment*. Risk treatment consists of activities aimed at determining and selecting ways of dealing with risks, including risk avoidance, risk reduction, risk transfer and others.

[1] Available at `https://www.owasp.org/index.php/CISO_AppSec_Guide:_Criteria_for_Managing_Application_Security_Risks`.

1.2 Informal introduction to attack trees and attack–defense trees

Attack trees *Attack trees* [Sch99] are a well-established and commonly used graphical formalism for security modeling. Inspired by fault trees [HRVG81], which are used in system reliability analysis, and threat logic trees [Wei91], they provide readable and structured representation of possible attacks against a system to protect. Their hierarchical structure reveals common features of the attacks and enables quantitative evaluation of security, thus highlighting the most severe vulnerabilities to focus on while implementing countermeasures. Formally, attack trees are *rooted trees* with *labeled nodes*. The labels of the nodes represent *goals* of the attacker, with the label of the root node corresponding to the attacker's main goal. This, often high-level and abstract, goal is recursively *refined* into subgoals represented by the labels of the remaining nodes. The basic model of attack trees admits two types of refinements: *conjunctive refinement* `AND` and *disjunctive refinement* `OR`. To achieve a goal of an `AND` node one needs to achieve the subgoals of all of its children, whereas to achieve the goal of an `OR` node it is enough to achieve any of the goals of its child nodes. Another often considered refinement is the *sequential conjunctive* refinement (`SAND`). Similarly as in the case of the conjunctive refinement, achieving a goal of a `SAND` node requires achieving the subgoals of all of its children, but in a specific order.

Attack–defense trees *Attack–defense trees* [KMRS14] enhance the expressive power of attack trees by allowing for explicitly depicting goals of a defender in the model. In a scenario represented by an attack–defense tree, a goal of an actor (attacker or defender) can be *countered* by a goal of the other actor. That is, each of the nodes, including the non-refined ones, can have among its children a single node of the other actor, which represents a way of countering the parent node's goal. The goal of a node having a countermeasure among its children is achieved if the achievement conditions following from the node's refinement are satisfied (if the node is refined) *and* the goal of the countermeasure node is *not* achieved. Note that the requirement of every node having *at most one* countermeasure among its children is not limiting at all: should it be possible to counter a goal of a node in many different ways, all of these ways can be gathered under a common disjunctively refined parent, which can then be the single unique countermeasure of the node.

According to the terminology introduced in [KMRS14], the root actor in an attack–defense tree is called the *proponent* and the other actor is the *opponent*. The aim of the proponent is to achieve the root goal, whereas the opponent tries to make this impossible. The labels of the nodes that are non-refined are called *basic actions*. They represent actions that the actors execute to achieve the goals of the refined nodes.

Graphical notation When depicting attack–defense trees graphically, we use the standard conventions. The nodes of the attacker are represented with red ellipses, and the nodes of the defender with green rectangles. The `AND` nodes differ from the `OR` nodes in that the edges connecting them with their children are joined with an arc. The countermeasures are attached to the nodes they are countering via a dotted line. Both countermeasure and child nodes of a node are depicted *below* the node.

Running example An attack–defense tree used as a running example in this thesis is depicted in Figure 1. The scenario modeled with this tree is explained in Example 1.

Example 1. *In the scenario represented by the attack–defense tree from Figure 1, the proponent is the attacker and the opponent is the defender. The attacker wants to steal money from the defender's account. To achieve this goal, the attacker can use physical means, i.e., learn the victim's PIN, steal their card, and then withdraw cash from an ATM. To learn the PIN, the attacker could force the victim to reveal it or eavesdrop on the victim when they enter the PIN. The victim could prevent the latter by covering the keypad with hand. However, covering the keypad fails if the attacker monitors the keypad with a hidden micro–camera installed at an appropriate spot.*

Instead of attacking from a physical angle, the attacker can steal money by exploiting online banking services. In order to do so, they could learn the victim's user name and password. Both of these goals can be achieved by creating a fake bank website and using phishing techniques for tricking the account holder into entering their credentials. The attacker could also try to guess what the password and the user name are. Using very strong password would allow the account holder to counter such a guessing attack. Once the attacker obtains the credentials, they can use them for logging into the online banking services and execute a transfer. To prevent such an attack, transfer dispositions might be additionally secured with two-factor authentication using mobile phone text messages. This security measure could be counterattacked by stealing the victim's phone.

The tree in Figure 1 is a toy example, convenient for illustrating notions introduced further in the thesis. In Chapter 7, we construct a bigger, realistic attack–defense tree, based on an attack tree analyzed by the U.S. Department of Energy in [Nat15].

Repeated basic actions It is not rare that in an attack–defense tree some nodes bear the same label. In such a case, there are two ways of interpreting them.

– The nodes represent the same single instance of the goal – e.g., both of the nodes labeled with the *phishing* action in the tree from Figure 1 might refer to the same action, as described in Example 1. That is, the single action of setting up a fake bank's website and luring the victim into entering their credentials achieves both the *get user name* and *get password* goals.

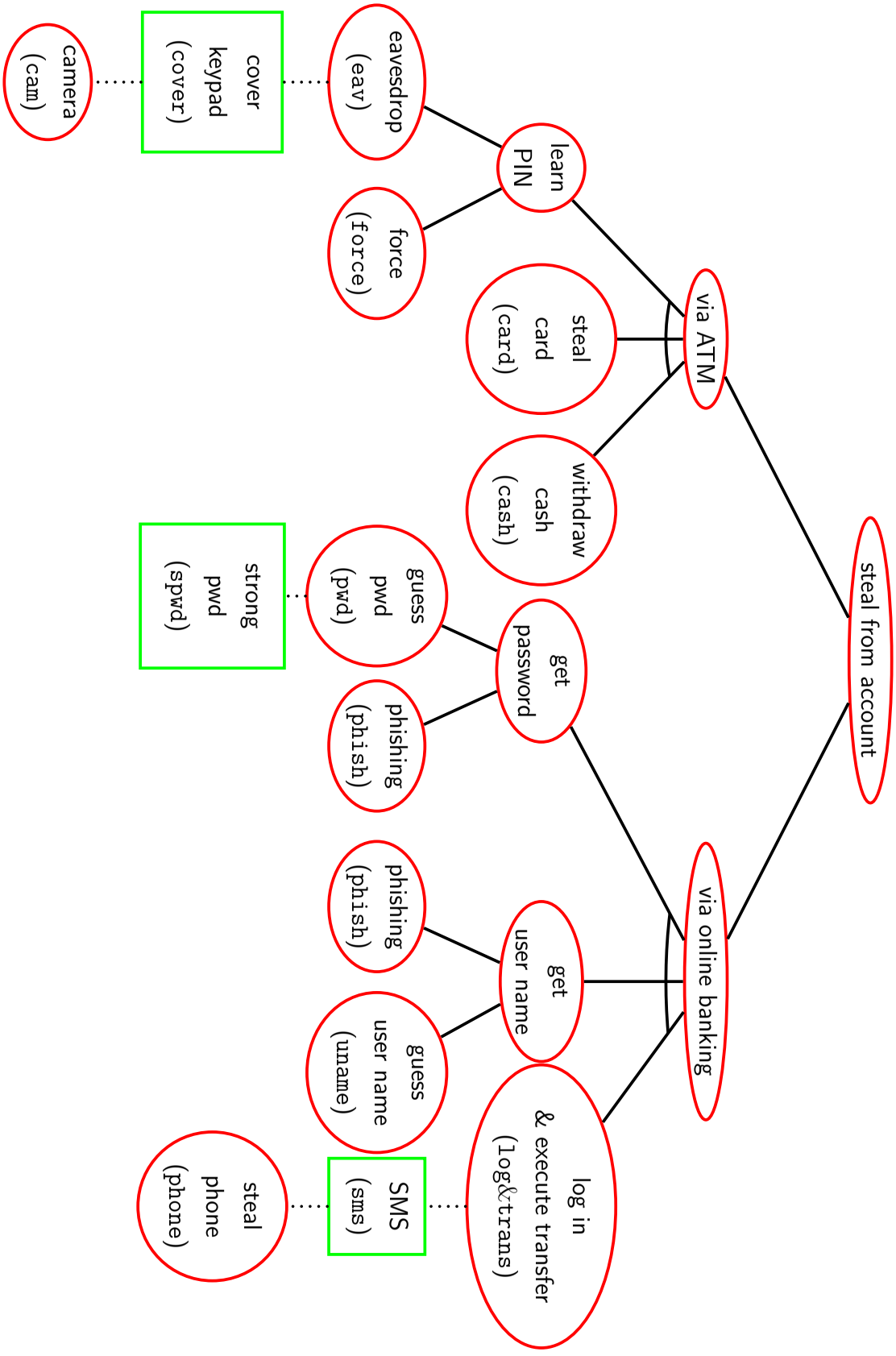


Figure 1: Attack–defense tree for stealing money from somebody's account

– Each of the nodes is treated as a distinct instance of the goal. For instance, in the scenario modeled with the tree from Figure 1 one could employ two different phishing techniques, each tailored specifically for achieving exactly one of the goals *get user name* and *get password*. In such a case, while both nodes could still be labeled *phishing*, they would represent distinct instances of a phishing attack.

The presence of different nodes being labeled in the same way is natural. Some goals and actions might contribute to multiple ways of attacking or defending a system, and some of these ways might require an action to be performed a number of times. It is easy to control the labels if a tree is small and if it is constructed manually, and to construct it with a fixed interpretation of repeated labels in mind. Problems might arise if a tree is a result of an automatic procedure, or if it is, for instance, composed from smaller trees created by different analysts analyzing subscenarios of the scenario, or having knowledge about particular subsystems of the system under consideration.

In this work we assume the first of the two ways of interpretation given above. There are at least two reasons for this choice. We believe that this way corresponds to the intuitive reading of attack–defense trees, that is, we believe that when a person is given a tree, they are more likely to consider the repeated labels to stand for the same event, and not for distinct instances of it. More importantly and more formally, this interpretation is more liberal than the other one. It allows for modeling both goals and actions contributing to multiple goals, by using the same label, while keeping the possibility of modeling different instances of the same action or goal, by using slightly different labels. Should the other interpretation be used, it would be impossible to model the possibility of a single action contributing to multiple attacks.

Following [BK17], we call a basic action that serves as a label for at least two nodes a *clone* or a *cloned basic action*. Nodes representing distinct instances of the same action or goal are assumed to have different labels. In this setting, it is convenient to use directed acyclic graphs, where nodes bearing the same label are merged into a single node, instead of trees. Such approach leads to a better readability of models and can be exploited for speeding up computations performed on trees. Using directed acyclic graphs instead of trees is a standard measure in the field of fault trees analysis [RS15] (where subtrees rooted in nodes bearing the same label are called *shared subtrees*, and the analogue of clones are *shared basic events*), taken also sometimes in the case of attack trees, e.g., in [AHPS14]. Our definition of attack–defense trees based on directed acyclic graphs will be given in Chapter 2. The graphical representation of the tree from Figure 1 redrawn as a directed acyclic graph is given in Figure 2. The only difference between the two representations is that in the latter the nodes labeled with the phishing action are merged into a single node.

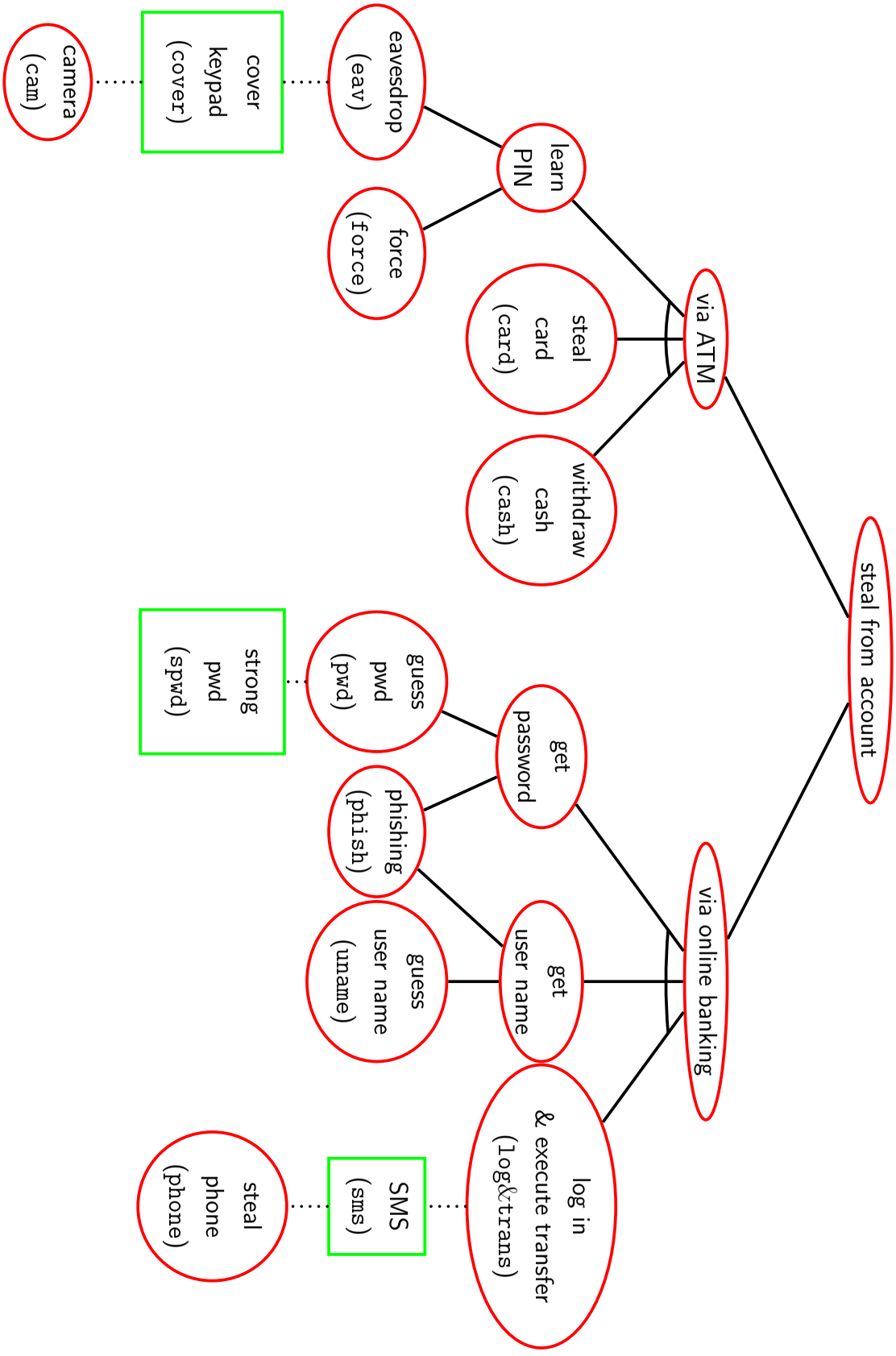


Figure 2: Attack–defense tree for stealing money from somebody's account represented using a directed acyclic graph

1.3 Research questions and our contributions

Research questions The main goal of the research that this thesis is a byproduct of was to identify and to lift limitations on the usefulness of attack–defense trees in the risk assessment process. Among the limitations that we were able to identify are the following.

1. Many methods for analysis of attack–defense trees that could be useful for estimating likelihood of attacks are either developed under the explicit assumption of trees not containing clones, or in a way that makes them not suitable for trees with clones.

2. Analysis methods focusing on a number of parameters at a time, e.g., for determining attacks that are optimal w.r.t.[2] both cost and success probability, are generally not efficient in the case of big models and/or can be applied to a limited number of parameters.

3. Approaches for optimal (in a well defined sense) selection of countermeasures in security scenarios modeled with attack–defense trees are either formulated in terms of a "what-if" analysis, that is, they allow for selection of countermeasures under fixed behavior of the attacker, or else they can be applied only to trees satisfying some structural restrictions.

4. The access of risk analysts to the latest developments in the field of attack–defense trees is very limited. New analysis techniques are being created yearly, and it is difficult to have a clear overview of the field, even for the researchers working in the domain. Furthermore, very few tools implementing most recent analysis techniques are accessible, and not all of the existing tools are maintained.

In the light of the above limitations, we have posed and tried to answer the following research questions.

1. How to determine optimal attacks efficiently in the presence of clones?

2. How to determine efficiently attacks optimal w.r.t. to multiple parameters, possibly in the presence of clones?

3. How to determine efficiently sets of optimal countermeasures, possibly in the presence of clones?

Contributions Trying two answer the first two of the above questions, we have analyzed one of the fundamental methods for analysis of attack–defense trees, namely, the

[2]To be read as *with respect to*.

bottom-up procedure for computing attack-related parameters. It can be used, for instance, for efficiently obtaining values such as the minimal cost or the maximal success probability of an attack. We were able to determine causes of the bottom-up procedure malfunctioning in the presence of clones. This allowed us for developing alternative methods for computing such parameters, and for constructing efficient algorithms for determining attacks optimal from the point of view of the attacker. We were able to adapt these new methods for the purpose of multi-parameter analysis of security scenarios modeled with trees.

We have also tackled the problem of exploiting attack–defense trees models for optimal selection of countermeasures in security scenarios, thus partially answering the third of the research question posed above. We have developed a method for extracting possible behaviors of a rational attacker from models, as well as ways of countering such behaviors by the defender. This information can be used as input for standard optimization methods, thus allowing for determining, e.g., a set of countermeasures the implementation of which fits a given budget and maximizes the necessary investment of the attacker into achieving their goal.

Finally, we have made efforts to raise accessibility of the recent developments in the field of attack tree analysis to general public. First, we have surveyed relevant research articles published in the years 2014-2018[3]. We have assessed the strong and weak points of the methodologies presented within, studied relations between them and described our findings. Second, we have developed a tool support for the analysis methods presented in this thesis. The `OSEAD` tool (*Optimal Strategies Extractor for Attack–Defense Trees*) is an easy-to-use and freely available software that aims at supporting risk analysts in their work.

The risk assessment process is a somewhat delicate task, a one the results of which are generally not made accessible to anyone beyond the parties of interest. This might be the cause for realistic attack tree-based models being almost impossible to find. To validate the methods described in this thesis, we have thus created a realistic attack–defense tree model, based on a security scenario considered in [Nat15]. We conducted a case study of the scenario modeled with the tree, using some of the methods described in this thesis. We hope that the model itself might be useful for other researchers as a testing ground for their ideas.

1.4 Thesis structure

In Chapter 2, we provide the formal background necessary for full understanding of the remaining parts of the thesis.

To place the results of our research in the context of the field of attack tree analysis,

[3]The previously published articles have been extensively surveyed before, and compared taking different criteria into account, see, e.g., [KPS14, HKCH17, NPMK18].

we describe some of the existing works that are closely related to ours in Chapter 3, which is based on our survey [WAFP19].

The problem of quantitative analysis of security using attack–defense trees containing clones is studied in depth in Chapter 4. The foundations of the framework described in this chapter have been layed in [KW18]. Most of the results are new, and have not been prepared for publication yet.

Chapter 5, based on [FW19b], is devoted to the multi-parameter analysis of security.

The optimal selection of countermeasures in scenarios modeled with trees is the focal point of Chapter 6. The ideas underlying the approach described in the chapter, as well as some preliminary results, have been presented in [KW17]. The remainder of the chapter consists of recent developments that have not been published yet.

Finally, in Chapter 7, we describe the `OSEAD` tool and use it for conducting a case study of a security scenario related to the energy sector. The study has been published as [FW19a].

We conclude in Chapter 8.

Chapter 2

Preliminaries

Reliable methods for modeling and analysis of security necessarily require firm formal foundations. In Section 2.1– 2.4, we recall and illustrate with examples some of the notions and concepts underlying security analysis based on attack–defense trees. Definition of attack–defense trees based on directed acyclic graphs is given in Section 2.5. The last part of this chapter, Section 2.6, is devoted to the so-called attribute domains and the bottom-up procedure, which is a standard tool for analysis of models based on `AND/OR` trees, including attack trees, attack–defense trees, fault trees and many others.

2.1 Elements of set theory and abstract algebra

We use $\mathbb{N}$ for the set of natural numbers, including zero, and $\mathbb{R}$ for the set of real numbers. For $n \in \mathbb{N}$ and $r \in \mathbb{R}$, the set of natural numbers greater than or equal to n and the set of real numbers greater than or equal to r are denoted by $\mathbb{N}_{\geqslant n}$ and $\mathbb{R}_{\geqslant r}$, respectively.

The number of elements of a finite set X is denoted by $|X|$. We use 2^X for the set of all subsets of X (the *powerset* of X). For a subset Y of X, we write $Y \subseteq X$ if $Y \in 2^X$, and $Y \subset X$ if $Y \in 2^X \backslash \{X\}$. A subset R of the Cartesian product $X \times X$ is called a *binary relation over* X. For better readability, we sometimes write xRy instead of $(x, y) \in R$. Binary relations that will be of particular interest for us are partials orders.

Definition 1 (Partial order)**.** *A binary relation $\preceq$ over a set X is called a partial order on X if*

- *it is reflexive, i.e., $x \preceq x$ for every $x \in X$,*
- *it is antisymmetric, i.e., if $x \preceq y$ and $y \preceq x$ for some $x, y \in X$, then $x = y$,*
- *it is transitive, i.e., if $x \preceq y$ and $y \preceq z$ for some $x, y, z \in X$, then $x \preceq z$.*

Definition 2 (Partially ordered set)**.** *A partially ordered set is a pair $(X, \preceq)$, where X is a set and $\preceq$ is a partial order on X.*

For a partially ordered set $(X, \preceq)$, we use $x \prec y$ to denote the fact that $x \preceq y$ and $x \neq y$. An element x of X is a *minimal* (respectively, *maximal*) *element* w.r.t. the order $\preceq$ if there is no $y \in X$ such that $y \prec x$ (respectively, $x \prec y$).

Example 2. *For every set X, the pair $(2^X, \subseteq)$ is a partially ordered set. The empty set $\emptyset$ is the unique minimal element w.r.t. the relation of inclusion $\subseteq$, and the unique maximal element w.r.t. this order is the set X.*

Recall that a *function f from a set X to a set Y* is defined by a subset G_f of the Cartesian product $X \times Y$ such that for every $x \in X$ there is exactly one $y \in Y$ satisfying $(x, y) \in G_f$. The set G_f is called the *graph of the function f*. The notation $f \colon X \to Y$ is used to denote the fact that f is a function from X to Y. If $(x, y) \in G_f$, then y is called *the image of x by f* and denoted by $f(x)$. If the set Y is obvious from the context or irrelevant, the function f is said to be a *function on X*. A *binary operation on a set X* is a function $f \colon X \times X \to X$. For f being a binary operation on X, we sometimes write xfy instead of $f(x, y)$.

A special example of the partial order is the canonical partial order on idempotent semirings.

Definition 3 (Semiring)**.** *Let X be a set and let $\oplus$ and $\otimes$ be binary operations on X. The triple $(X, \oplus, \otimes)$ is a semiring if*

- *both $\oplus$ and $\otimes$ are associative, i.e., $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ and $(x \otimes y) \otimes z = x \otimes (y \otimes z)$, for every $x, y, z \in X$,*
- *the operation $\oplus$ is commutative, i.e., $x \oplus y = y \oplus x$, for every $x, y \in X$,*
- *X contains neutral element for $\oplus$, i.e., an element $\mathsf{e}_\oplus$ satisfying $x \oplus \mathsf{e}_\oplus = x$, for every $x \in X$,*
- *X contains neutral element for $\otimes$, i.e., an element $\mathsf{e}_\otimes$ satisfying $x \otimes \mathsf{e}_\otimes = x$ and $\mathsf{e}_\otimes \otimes x = x$, for every $x \in X$,*
- *the neutral element $\mathsf{e}_\oplus$ for $\oplus$ is equal to the absorbing element $\mathsf{a}_\otimes$ for $\otimes$, i.e., $x \otimes \mathsf{e}_\oplus = \mathsf{e}_\oplus$, for every $x \in X$,*
- *the operation $\otimes$ distributes over $\oplus$, i.e., $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ and $(y \oplus z) \otimes x = (y \otimes x) \oplus (z \otimes x)$ for every $x, y, z \in X$.*

A semiring $(X, \oplus, \otimes)$ is *commutative* if the operation $\otimes$ is commutative. If the operation $\oplus$ is idempotent, that is, if for every $x \in X$ the equality $x \oplus x = x$ holds, then the semiring $(X, \oplus, \otimes)$ is an *idempotent semiring*. Every idempotent semiring admits a partial order defined as follows.

Definition 4 (Canonical partial order on idempotent semiring)**.** *Let $(X, \oplus, \otimes)$ be an idempotent semiring. The canonical partial order on $(X, \oplus, \otimes)$ is the order defined for $x, y \in X$ by $x \preceq y$ if and only if $x \oplus y = y$.*

We illustrate the notion of canonical partial order with the following two examples.

Example 3. *For a set X, the triple $(2^X, \cup, \cap)$ is a commutative idempotent semiring. The neutral elements for the union and intersection of sets are the empty set $\emptyset$ and the set X, respectively. The empty set is also the absorbing element for the intersection. The canonical partial order on this semiring is the inclusion relation, defined for $Y, Z \in 2^X$ by $Y \preceq Z$ if and only if $Y \cup Z = Z$.*

Example 4. *The commutative idempotent semiring $([0,1], \max, \cdot)$, where $\cdot$ is the multiplication operator, belongs to the class of so-called Viterbi semirings. The neutral elements for the operation of taking maximum and the multiplication are 0 and 1, respectively. The former is also the absorbing element for the multiplication. The canonical partial order on this semiring is the* less than or equal to *relation $\leqslant$, defined for $x, y \in [0,1]$ by $x \preceq y$ if and only if $\max(x,y) = y$.*

If every two elements of a set X are comparable under a partial order $\preceq$, that is, if $x \preceq y$ or $y \preceq x$ holds for every two elements $x, y \in X$, then $\preceq$ is called *total order*. If $\preceq$ is a total order, then the pair $(X, \preceq)$ is a *totally ordered set*.

Another relation that will be of use for us is the equivalence relation.

Definition 5 (Equivalence relation)**.** *A binary relation $\equiv$ over a set X is called an equivalence relation on X if*

- *it is reflexive, i.e., $x \equiv x$ for every $x \in X$,*
- *it is symmetric, i.e., if $x \equiv y$ implies that $y \equiv x$, for every $x, y \in X$,*
- *it is transitive, i.e., if $x \equiv y$ and $y \equiv z$ for some $x, y, z \in X$, then $x \equiv z$.*

For a function $f\colon X \to Y$, we use $f|_Z$ to denote the *restriction of f* to the subset $Z \subseteq X$ of X, i.e., $g = f|_Z$ if $Z \subseteq X$, $g\colon Z \to Y$ and $g(x) = f(x)$, for $x \in Z$.

A function f is a *Boolean function* if $f\colon \{0,1\}^n \to \{0,1\}$, for some $n \in \mathbb{N}_{\geqslant 1}$.

Definition 6. *Let f be a Boolean function on $\{0,1\}^n$, with $n \in \mathbb{N}_{\geqslant 1}$, and let $k \in \{1, \dots n\}$. The function f is positive (respectively, negative) in the k-th variable if for every $(x_1, \dots, x_{k-1}, x_{k+1}, \dots, x_n) \in \{0,1\}^{n-1}$ the inequality*

$$f(x_1, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n) \leqslant f(x_1, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n)$$

(respectively,

$$f(x_1, \dots, x_{k-1}, 0, x_{k+1}, \dots, x_n) \geqslant f(x_1, \dots, x_{k-1}, 1, x_{k+1}, \dots, x_n))$$

holds.

Finally, an *unranked function* is defined as follows.

Definition 7 (Unranked function)**.** *An unranked function on a set X is a family of functions $(f_n)_{n=1}^{+\infty}$ such that $f_n\colon X^n \to X$, for $n \in \mathbb{N}_{\geqslant 1}$.*

Throughout the thesis, we naturally treat binary, associative operations as unranked functions. In such a case, we assume that when provided with a single argument, the function returns the argument itself. The following example illustrates the notion of unranked function.

Example 5. *The families $(f_n)_{n=1}^{+\infty}$ and $(g_n)_{n=1}^{+\infty}$ with f_n and g_n being defined for $n \in \mathbb{N}_{\geqslant 1}$ and $x_1, \ldots, x_n \in \mathbb{R}$ as*

$$f_n(x_1, \ldots, x_n) := \sum_{i=1}^{n} x_i,$$

$$g_n(x_1, \ldots, x_n) := \prod_{i=1}^{n} x_i,$$

are unranked functions on $\mathbb{R}$. To represent $(f_n)_{n=1}^{+\infty}$ and $(g_n)_{n=1}^{+\infty}$ in a simple manner, we would use the (binary and associative) operators $+$ and $\cdot$, respectively.

Another concept which will be of use in our considerations is that of a multiset.

Definition 8 (Multiset)**.** *Multiset is a pair (X, m), where X is the underlying set of the multiset and $m\colon X \to \mathbb{N}$ is the multiplicity function.*

The multiplicity function describes the number of occurrences of particular elements of the underlying set in the multiset. For a set X, we denote by $\mathfrak{M}(X)$ the set of all multisets whose underlying set is X. If $M_1 = (X, m_1)$ and $M_2 = (X, m_2)$ are multisets belonging to $\mathfrak{M}(X)$, then their *sum* is defined as $M_1 \biguplus M_2 := (X, m_1 + m_2)$, with $(m_1 + m_2)(x)$ defined as $m_1(x) + m_2(x)$ for $x \in X$. For simplicity, we use the $\{|\cdot|\}$ notation to denote multisets, and when defining a multiset (X, m), we specify the function m by explicitly listing each of the elements x of X the corresponding number $m(x)$ of times. For example, we write $\{|a, a, b|\}$ for the multiset $(\{a, b, c\}, m(a) = 2, m(b) = 1, m(c) = 0) \in \mathfrak{M}(\{a, b, c\})$.

Example 6. *Let $M_1 = \{|a, a, b|\}$ and $M_2 = \{|a, b, c|\}$ be multisets belonging to $\mathfrak{M}(\{a, b, c\})$. Their sum is $M_1 \biguplus M_2 = \{|a, a, a, b, b, c|\}$.*

We finish this section with a simple and yet useful lemma.

Lemma 1. *Let $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$, for $k \in \mathbb{N}_{\geqslant 1}$, be sets such that $A_i \cap B_j = \emptyset$, for $i, j \in \{1, \ldots, k\}$, $i \neq j$, satisfying*

$$\bigcup_{j=1}^{k} A_j \subseteq \bigcup_{j=1}^{k} B_j.$$

If $A_i \neq B_i$, for some $i \in \{1, \ldots, k\}$, then $A_i \subset B_i$.

Proof. Let $i \in \{1, \ldots, k\}$ be such that $A_i \neq B_i$. Since $A_i \cap B_j = \emptyset$ for $j \in \{1, \ldots, k\}$, $j \neq i$, and every element of A_i belongs to $\bigcup_{j=1}^{k} B_j$, it follows that every element of A_i belongs to B_i. □

2.2 Elements of term rewriting

For technical reasons, it will be sometimes useful to transform algebraic expressions over semirings into a specific form. Formally, we will apply to such expressions *term rewriting rules*, which will iteratively reduce the expressions to the desired form. This section, based on [BN98], is devoted to introducing notions necessary for defining the above mentioned reductions.

An *abstract reduction system* is a pair $(A, \to)$, where A is a set and $\to$ is a binary relation on A, called *reduction*. The *reflexive transitive closure* of the reduction $\to$, denoted $\xrightarrow{*}$, is defined as

$$\begin{aligned}\xrightarrow{*} := {} & \to \\ & \cup \{(x,x) \colon x \in A\} \\ & \cup \{(x,y) \colon \text{ there is } n \in \mathbb{N}_{\geqslant 1} \text{ and } x_1, \ldots, x_n \in A \text{ such that} \\ & \quad x \to x_1, x_n \to y \text{ and } x_i \to x_{i+1}, \text{ for } i \in \{1, \ldots, n-1\}\}.\end{aligned}$$

Intuitively, if $x \xrightarrow{*} y$, then x can be reduced to y in a finite number of steps using the reduction $\to$. An element $x \in A$ is *reducible*, if there is $y \in A$, $y \neq x$, such that $x \to y$. If $x \in A$ is not reducible, then it is said to be *in normal form*. An element $y \in A$ is a *normal form of* x if $x \xrightarrow{*} y$ and y is in normal form.

Example 7 (Example 2.1.2 in [BN98])**.** *Let* $A = \mathbb{N}\backslash\{0,1\}$ *and* $\to = \{(m,n) \colon m > n \text{ and } n \text{ divides } m\}$*. The elements of* A *that are not reducible are the prime numbers. An element* $p \in A$ *is a normal form of* $m \in A$ *if and only if* p *is a prime factor of* m*.*

Among the properties of reduction systems that will be of interest for us are local confluence and termination.

Definition 9. *Let* $(A, \to)$ *be an abstract reduction system. The reduction* $\to$ *is*

- *locally confluent, if for every* $x, y_1, y_2 \in A$ *satisfying* $x \to y_1$ *and* $x \to y_2$ *there is* $z \in A$ *such that* $y_1 \xrightarrow{*} z$ *and* $y_2 \xrightarrow{*} z$*,*
- *terminating, if there is no infinite sequence* $x_1, x_2, \ldots$ *of elements of* A *such that* $x_i \to x_{i+1}$*, for* $i \in \mathbb{N}_{\geqslant 1}$*.*

Example 8. *Let* $(A, \to)$ *be the reduction system considered in Example 7. The reduction* $\to$ *is terminating, since every sequence of reductions using* $\to$ *ends with a prime number that cannot be reduced further. The reduction is not locally confluent. Indeed, if* $n = p_1 \cdot p_2$ *is a product of two distinct prime numbers, then* $n \to p_1$ *and* $n \to p_2$*, but neither of the two primes is reducible.*

Note that a reduction system in which every element can be reduced to exactly one element is trivially locally confluent. This is the case, since if for every $x \in A$ there is

exactly one $y \in A$ such that $x \to y$, then the only possible choice of y_1 and y_2 from the definition of local confluence is $y_1 = y, y_2 = y$, and the second part of the definition is satisfied by $z = y$.

Reduction systems that are both locally confluent and terminating have the following property.

Lemma 2 (Reformulation of Theorem 2.1.9 and Lemma 2.7.2 from [BN98])**.** *Let $(A, \to)$ be an abstract reduction system. If $\to$ is locally confluent and terminating, then every element of A has a unique normal form.*

2.3 Elements of graph theory

In this section, we recall basic notions necessary for defining attack–defense trees using directed graphs. The content of this section is based mainly on [BM08].

Definition 10 (Directed graph)**.** *A directed graph is an ordered pair $D = (V, A)$ consisting of a set V of* nodes *and a set A, disjoint from V, of* arcs*, together with an* incidence function *ψ_D that associates with each arc of D an ordered pair of (not necessarily distinct) nodes of D.*

Let $D = (V, A)$ be a directed graph. If a is an arc in A and $\psi_D(a) = (w, v)$, then a is said to *join* w and v. In this work, we consider directed graphs with no parallel arcs, i.e., directed graphs having injective incidence functions. For such graphs, we identify arcs with their images by the incidence function. In other words, we assume that $A \subseteq V \times V$.

Let (V, A) be a directed graph. If the pair (w, v) is an arc in A, then w is called a *child* of v and v is a *parent* of w. A *path* in (V, A) is a sequence of nodes of V in which each node is a child of its successor in the sequence.

Definition 11 (Directed acyclic graph)**.** *A directed acyclic graph (DAG) is a directed graph (V, A) in which none of the nodes appears more than once in any of the paths in (V, A).*

When depicting DAGs graphically, we place children of a node below that node, and connect each of them with the parent using a line segment.

Example 9. *Let $V = 2^{\{0,1\}}$ and let $A = \{(X, Y) \colon X \subset Y\}$. In the DAG (V, A), depicted in Figure 3, the sequence $(\emptyset, \{0\}, \{0, 1\})$ is a path, since the node $\emptyset$ is a child of the node $\{0\}$, and the latter is a child of the node $\{0, 1\}$.*

If a DAG (V, A) contains a unique node that has no parents, then this node is called the *root* of (V, A). DAG containing a root node is called *rooted*.

Example 10. *The DAG depicted in Figure 3 is rooted. Its root is the node $\{0, 1\}$.*

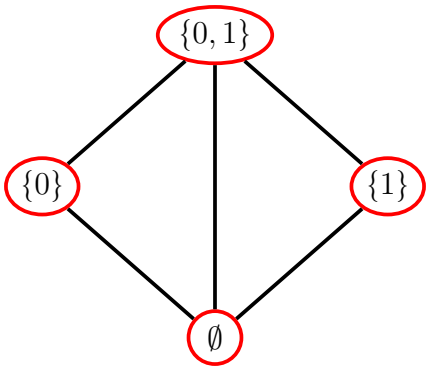


Figure 3: DAG $(2^{\{0,1\}}, \{(X, Y) \colon X \subset Y\})$.

A directed graph (V', A') is a *subgraph* of a directed graph (V, A) if $V' \subseteq V$ and $A' \subseteq A$. If the set A' consists of all the arcs of A whose both nodes belong to V', then (V', A') is a subgraph of (V, A) *induced by* V'. If both (V, A) and (V', A') are DAGs, we use the word *subdag* for (V', A'), instead of subgraph.

Example 11. *Consider again the DAG (V, A) depicted in Figure 3. Let*

$$V' = \{\emptyset, \{0\}, \{0, 1\}\},$$
$$A' = \{(\emptyset, \{0\}), (\{0\}, \{0, 1\})\} \text{ and}$$
$$A'' = \{(\emptyset, \{0\}), (\{0\}, \{0, 1\}), (\emptyset, \{0, 1\})\}.$$

The pair (V', A') is a subdag of (V, A), and (V', A'') is a subdag of (V, A) induced by V'.

If for every two nodes u, v of a directed graph (V, A) there is a sequence of nodes $v_1, v_2, \ldots, v_k$ such that $v_1 = u$, $v_k = v$ and for every $i \in \{1, \ldots, k-1\}$ either (v_i, v_{i+1}) or (v_{i+1}, v_i) is an arc in A, then the graph is said to be *connected*. A maximal, w.r.t. to the inclusion of both nodes and arcs sets, connected subgraph of a directed graph (V, A) is called a *component* of (V, A). Note that the only component of a connected directed graph is the graph itself, and that every rooted DAG is connected.

2.4 Elements of formal language theory

It is standard to represent attack–defense trees as typed ground terms over a specific signature. In this section, we briefly recall notions necessary for the understanding of this representation. An interested reader is referred to [Koz97] for more details.

An *alphabet* is any finite set. The elements of an alphabet Σ are called *symbols*. A *string* over Σ is any finite-length sequence of elements of Σ. The length of a string s is the number of symbols in s. The unique string of length zero over Σ is called the *empty string* and is denoted by ϵ.

Example 12. *Let $\Sigma = \{a, b\}$. Both $s_1 = aaa$ and $s_2 = abab$ are strings over Σ. The length of s_1 is three, and the length if s_2 is four.*

The set of all strings over an alphabet Σ is denoted by Σ^*. A *language* over Σ is any subset of Σ^*. Some languages can be concisely described with a finite set of *production rules*. Production rules specify how strings in a language can be transformed into other strings in this language.

Example 13. *Let $\Sigma = \{a, b\}$ and let $L = \{\epsilon, aa, aaaa, aaaaaa, \ldots\}$ be the language of strings containing an even number of the letter a. The language L can be described using the production rules*

$$s ::= \epsilon \mid saa.$$

They can be read as follows: every string s in L is either the empty string ϵ or a concatenation of a string in L with two letters a.

An *algebraic signature* is an alphabet consisting of *function symbols* in which each symbol is assigned a natural number, called its *arity*.

Definition 12 (Algebraic signature)**.** *An algebraic signature is a pair (Σ, ar) such that Σ is an alphabet consisting of function symbols and $ar\colon \Sigma \to \mathbb{N}$ is a function assigning a natural number to each of the symbols.*

If (Σ, ar) is an algebraic signature, then an element of Σ is called *constant*, *unary*, *binary*, *trinary* or *n-ary* if its arity is $0, 1, 2, 3$ or n, respectively. An expression built from the function symbols of Σ that respects the arities of symbols is called a *ground term* over the signature.

Definition 13 (Ground term over a signature)**.** *The set T_Σ of ground terms over a signature (Σ, ar) is defined recursively as follows. Any constant function symbol $c \in \Sigma$ is in T_Σ. If $t_1, \ldots, t_n \in T_\Sigma$ and f is an n-ary function symbol of Σ, then $f(t_1, \ldots, t_n) \in T_\Sigma$.*

The following example illustrates the notion of ground terms over a signature.

Example 14. *Consider the alphabet $\Sigma = \{x, y, \vee, \wedge\}$, with x and y being constant symbols, and $\vee$ and $\wedge$ being unranked functions, i.e., families $(\vee_n)_{n\in\mathbb{N}_{\geqslant 1}}$, $(\wedge_n)_{n\in\mathbb{N}_{\geqslant 1}}$, with the arity function defined as $ar(\vee_n) = n$ and $ar(\wedge_n) = n$, for $n \in \mathbb{N}_{\geqslant 1}$. The set of ground terms over the signature (Σ, ar) is*

$$T_\Sigma = \{x, y, \vee(x, x), \vee(y, y), \vee(x, y), \wedge(x, x), \wedge(y, y), \wedge(x, y), \ldots\},$$

and it can be seen as the set of representations of all propositional formulæ involving variables x and y and logical conjunction and disjunction.

On the top of a signature a type system can be defined, assigning types (called *sorts*) to symbols. This is usually achieved by generalizing the arity function in the following manner.

Definition 14 (Many-sorted algebraic signature)**.** *A many-sorted algebraic signature is a triple (S, Σ, ar), where S is a set of sorts, Σ is an alphabet consisting of function symbols and ar is a function assigning to each of the symbols its arity of the form $s_1 \times \ldots \times s_n \to s_{n+1}$, for $s_1, \ldots, s_{n+1} \in S$.*

Intuitively, the arity function defined as in Definition 14 specifies for a function symbol $f \in \Sigma$ the number of its arguments, the sorts of the arguments, and the sort of the image by f. For the constant symbols, i.e., when $n = 0$, the arity function describes their sorts.

Since the sets of ground terms over a signature are special strings over an alphabet, they can sometimes be specified using appropriate production rules, as illustrated in the next example.

Example 15. *Let T_Σ be the language produced by the grammar*

$$t^{\mathrm{s}} ::= x^{\mathrm{s}} \mid y^{\mathrm{s}} \mid \vee^{\mathrm{s}}(t^{\mathrm{s}}, \ldots, t^{\mathrm{s}}) \mid \wedge^{\mathrm{s}}(t^{\mathrm{s}}, \ldots, t^{\mathrm{s}}) \mid \wedge\neg^{\mathrm{s}}(t^{\mathrm{s}}, t^{\bar{\mathrm{s}}}),$$

for $\mathrm{s} \in \{\mathrm{s}_1, \mathrm{s}_2\}$ and $\overline{\mathrm{s}_1} = \mathrm{s}_2$, $\overline{\mathrm{s}_2} = \mathrm{s}_1$. The language T_Σ is the set of ground terms over many-sorted algebraic signature

$$(\{\mathrm{s}_1, \mathrm{s}_2\}, \{x^{\mathrm{s}_1}, x^{\mathrm{s}_2}, y^{\mathrm{s}_1}, y^{\mathrm{s}_2}, \vee^{\mathrm{s}_1}, \vee^{\mathrm{s}_2}, \wedge^{\mathrm{s}_1}, \wedge^{\mathrm{s}_2}, \wedge\neg^{\mathrm{s}_1}, \wedge\neg^{\mathrm{s}_2}, \}, ar),$$

with the arity function ar defined as

$$\begin{aligned} ar(x^{\mathrm{s}}) &= \mathrm{s}, \\ ar(y^{\mathrm{s}}) &= \mathrm{s}, \\ ar(\vee_n^{\mathrm{s}}) &= \mathrm{s}^n \to \mathrm{s}, \\ ar(\wedge_n^{\mathrm{s}}) &= \mathrm{s}^n \to \mathrm{s}, \\ ar(\wedge\neg^{\mathrm{s}}) &= \mathrm{s} \times \bar{\mathrm{s}} \to \mathrm{s}, \end{aligned}$$

for $\mathrm{s} \in \{\mathrm{s}_1, \mathrm{s}_2\}$ and $n \in \mathbb{N}_{\geqslant 1}$.

2.5 Attack–defense trees

Various definitions of attack(–defense) trees can be found in the literature, each of them being either graph-based [AHPS14, KW17] or term-based [KMRS14, AN15, GHL+16]. We use the following definition based on DAGs.

Definition 15 (Attack–defense tree)**.** *An attack–defense tree is a tuple $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$, where*

- *(V, A) is a rooted DAG,*
- *L is a set of labels representing the attacker's and the defender's goals,*
- *$\lambda \colon V \to \mathrm{L}$ is an injective function assigning labels to the nodes,*

– actor: $V \to \{\mathtt{a}, \mathtt{d}\}$ *is a function assigning actors to the nodes, in such a way that every node has at most one child assigned to the other actor,*

– ref: $V \to \{\mathtt{OR}, \mathtt{AND}, \mathtt{N}\}$ *describes refinements of nodes. We use* OR *for disjunctively and* AND *for conjunctively refined nodes,* N *stands for the non-refined nodes, i.e., nodes labeled with basic actions,*

– *for every node* $v \in V$, $\mathrm{ref}(v) = \mathtt{N}$ *if and only if* v *has no child assigned to the same actor as* v*,*

– *for every node* $v \in V$*, the set of children of* v *is totally ordered*[1]*, and if* v *has a child belonging to the other actor, then this child is the maximal element of this set*[2]*.*

From now on, whenever we use the word "tree", we mean attack–defense tree. The *root* of a tree T, denoted $\mathrm{root}(T)$, is the root of its underlying DAG. The actor assigned to the root of a tree is called *proponent*, and the other one is called *opponent*. For a tree T, we use p_T to mark the components of T assigned to the proponent, and o_T for those assigned to the opponent, i.e., p_T stands for $\mathrm{actor}(\mathrm{root}(T))$ and o_T stands for the other actor. The labels of the non-refined nodes are *basic actions*. For $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$, we denote by $\mathbb{B}^{\mathrm{s}_T}$ the set of basic actions of the corresponding actor in T, and we set $\mathbb{B}_T := \mathbb{B}^{\mathrm{p}_T} \cup \mathbb{B}^{\mathrm{o}_T}$. The universe of all basic actions is denoted with $\mathbb{B}$. Note that the fact that the labeling function from Definition 15 is injective implies that the sets $\mathbb{B}^{\mathrm{p}_T}$ and $\mathbb{B}^{\mathrm{o}_T}$ are disjoint. We use $\mathbb{T}$ for the set of all attack–defense trees.

Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree. For $v \in V$, we use

$$\mathrm{children}_T(v) := \{w \in V \colon wv \in A,\ \mathrm{actor}(w) = \mathrm{actor}(v)\}$$

to denote the set of children of v that are assigned the same actor as v. Whenever a function acts over children of v, the order of its arguments follows the total order of the set of children, beginning with the minimal element. If v has a child belonging to the other actor, this child is denoted by $\bar{v}$. If all of the nodes of a tree belong to the same actor, then the tree is an *attack tree*. Finally, for $v \in V$, we use $T(v)$ to denote the *maximal subdag of* T *rooted at* v , i.e., a subdag of T induced by all the nodes w such that there is a path from w to $\mathrm{root}(T)$ passing by v.

While labels of refined nodes are important when creating a tree, they might not be necessary for its analysis. Indeed, they are disregarded in most of the formal approaches to the attack–defense trees analysis, e.g., in [AN15, GHL+16, KW17]. Similarly, it is often irrelevant for the analysis who the proponent is, i.e., whether the root actor is

[1] In the case of graph-based definitions of attack trees, the condition of children being ordered is often formulated by defining a function that maps nodes to *lists* of their children, see, e.g., [AHPS14, KRS15]. In the term-based definitions the order is explicit in the form of the term.

[2] The choice of this particular child being the maximal element w.r.t. the order is dictated by the fact that such a child is listed as the last one in the standard term-based notation.

the attacker or the defender. This is also true for the methods presented in this thesis. Therefore, for the purpose of concise representation of trees, we employ the standard term-based notation, which relies only on the labels of the non-refined nodes and on the refinement operators of the refined ones, and distinguishes the actors with respect to the root goal of the tree.

Definition 16 (Attack–defense term)**.** *An attack–defense term over a set of basic actions* $\mathbb{B}$ *is a typed term conforming with the grammar*

$$t^{\mathrm{s}} ::= \mathtt{b}^{\mathrm{s}} \mid \mathtt{OR}^{\mathrm{s}}(t^{\mathrm{s}}, \ldots, t^{\mathrm{s}}) \mid \mathtt{AND}^{\mathrm{s}}(t^{\mathrm{s}}, \ldots, t^{\mathrm{s}}) \mid \mathtt{C}^{\mathrm{s}}(t^{\mathrm{s}}, t^{\bar{\mathrm{s}}}), \tag{1}$$

where $\mathtt{b} \in \mathbb{B}$*,* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$ *and* $\bar{\mathrm{p}} := \mathrm{o}$*,* $\bar{\mathrm{o}} := \mathrm{p}$*.*

With the following definition, we formalize the procedure for creating attack–defense terms corresponding to trees, sketched graphically in [KMRS14].

Definition 17 (Attack–defense term corresponding to an attack–defense tree)**.** *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree and let* $v \in V$ *be a node such that* $\mathrm{actor}(v) = \mathrm{s}_T$*,* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$*,* $\mathrm{ref}(v) = \mathtt{OP}$ *and* $\mathrm{children}_T(v) = \{v_1, \ldots, v_k\}$*, with the children being ordered according to their indices. Let* $t(T, v)$ *be the function defined recursively as follows*

$$t(T,v) := \begin{cases} \lambda(v)^{\mathrm{s}}, & \textit{if } \mathtt{OP} = \mathtt{N} \textit{ and } \bar{v} \textit{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}\left(\lambda(v), t(T, \bar{v})\right), & \textit{if } \mathtt{OP} = \mathtt{N} \textit{ and } \bar{v} \textit{ exists,} \\ \mathtt{OP}^{\mathrm{s}}\left(t(T, v_1), \ldots, t(T, v_k)\right), & \textit{if } \mathtt{OP} \neq \mathtt{N} \textit{ and } \bar{v} \textit{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}\left(\mathtt{OP}^{\mathrm{s}}(t(T, v_1), \ldots, t(T, v_k)), t(T, \bar{v})\right), & \textit{otherwise.} \end{cases}$$

The attack–defense term corresponding to T*, denoted* $t(T)$*, is then defined as* $t(T, \mathrm{root}(T))$*.*

In the remainder of this thesis, when using attack–defense terms, we skip types of the basic actions. For example, we would use $\mathtt{C}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{b}_2)$ instead of $\mathtt{C}^{\mathrm{p}}(\mathtt{b}_1^{\mathrm{p}}, \mathtt{b}_2^{\mathrm{o}})$. Since for a given tree T the sets $\mathbb{B}^{\mathrm{p}_T}$ and $\mathbb{B}^{\mathrm{o}_T}$ are disjoint, this does not introduce any ambiguity.

Example 16. *Using the abbreviations of basic actions in tree* T *from Figure 2, one obtains the corresponding attack–defense term*

$$\begin{aligned} t(T) = \mathtt{OR}^{\mathrm{p}}\Bigg(& \mathtt{AND}^{\mathrm{p}}\Bigg(\mathtt{OR}^{\mathrm{p}}\Big(\mathtt{C}^{\mathrm{p}}\big(\mathtt{eav}, \mathtt{C}^{\mathrm{o}}(\mathtt{cover}, \mathtt{cam})\big), \mathtt{force}\Big), \\ & \qquad \mathtt{card}, \\ & \qquad \mathtt{cash}\Bigg), \\ & \mathtt{AND}^{\mathrm{p}}\Bigg(\mathtt{OR}^{\mathrm{p}}\Big(\mathtt{C}^{\mathrm{p}}(\mathtt{pwd}, \mathtt{spwd}), \mathtt{phish}\Big), \\ & \qquad \mathtt{OR}^{\mathrm{p}}\Big(\mathtt{phish}, \mathtt{uname}\Big), \\ & \qquad \mathtt{C}^{\mathrm{p}}\Big(\mathtt{log\&trans}, \mathtt{C}^{\mathrm{o}}(\mathtt{sms}, \mathtt{phone})\Big)\Bigg)\Bigg). \end{aligned}$$

When introducing an attack–defense tree, we either use the corresponding attack–defense term or the graphical representation. In the former case, the order of children of particular nodes follows the order in which they appear in the term. Thus, the underlying attack–defense tree can be easily reconstructed, with the exception of the actors assigned to the nodes (attacker/defender) and the labels of refined nodes. In the latter, we assume that the children of a node are placed from left to right, following the corresponding total order.

2.6 Attribute domains for attack–defense trees

Among the existing approaches to analysis of attack–defense trees there are methods that can be formulated using the notion of *attribute domains* (even if originally they were not). In this section, we recall the notion of attribute domains and some of the ways in which they can be exploited for the purpose of analysis of attack–defense trees. Most of the notions and definitions used in this section are well-established [MO05, KMRS14, KW18], but we adapt them to the DAG-based formalization of attack–defense trees.

Intuitively, an attribute of an attack–defense tree is a piece of information regarding the scenario modeled with the tree. Attributes can represent quantitative aspects of the scenario, such as *minimal cost* of executing an attack or *maximal damage* caused by an attack. As it will be extensively illustrated in Section 3.1, they can also correspond to other scenario-related information, e.g., the ways in which goals and subgoals of the actors can be achieved.

Numerous methods for evaluation of attributes on attack–defense trees exist, and most of them involve a bottom-up procedure: some of them as the sole method of evaluation, some of them as a subprocedure. The idea behind the bottom-up procedure is to assign attribute values to the basic actions and to propagate them up to the root of the tree using appropriate operations at the intermediate nodes. The notions of an attribute and the bottom-up evaluation are formalized using attribute domains.

Definition 18 (Attribute domain)**.** *Let α be an attribute of attack–defense trees. An attribute domain for α is a tuple $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$, where*

- *D_α is a set of values that the attribute can attain,*
- *$\mathtt{OR}^{\mathrm{s}}_\alpha$ and $\mathtt{AND}^{\mathrm{s}}_\alpha$ are unranked functions on D_α, for $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$,*
- *$\mathtt{C}^{\mathrm{s}}_\alpha$ is a binary function on D_α, for $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$.*

In practice, α appearing in the above definition is usually a shorthand for an intuitive description of the attribute, such as `cost` for the *minimal cost for the proponent* attribute. To analyze an attack–defense tree using attribute domains, one assigns values of the attribute to the basic actions of the actors and then combines them using the domain's

operations. In the next example, the domain for the *minimal cost for the proponent* attribute is presented. The choice of its operations will be explained once a way in which they can be exploited is introduced. Further examples of attribute domains are given in Table 1.

Example 17. *The standard attribute domain for the* minimal cost for the proponent *attribute is* $A_{\mathtt{cost}} = (\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \min, +, +, \min, +, \min)$.

A function $\beta\colon \mathbb{B} \to D_\alpha$ is called a *basic assignment for attribute* α. A standard way of combining the values of the basic assignment to obtain the value of the attribute corresponding to the modeled scenario is the following.

Table 1: Selected attribute domains for attack–defense trees, where $x \star y := x \wedge \neg y$ for $x, y \in \{0, 1\}$

Attribute	α	D_α	$\mathtt{OR}^{\mathrm{p}}_\alpha$	$\mathtt{AND}^{\mathrm{p}}_\alpha$	$\mathtt{OR}^{\mathrm{o}}_\alpha$	$\mathtt{AND}^{\mathrm{o}}_\alpha$	$\mathtt{C}^{\mathrm{p}}_\alpha$	$\mathtt{C}^{\mathrm{o}}_\alpha$
Minimal cost for the proponent	`cost`	$\mathbb{R}_{\geqslant 0} \cup \{+\infty\}$	min	$+$	$+$	min	$+$	min
Maximal damage done by the proponent	`dmg`	$\mathbb{R}_{\geqslant 0} \cup \{-\infty\}$	max	$+$	$+$	max	$+$	max
Minimal skill level of the proponent	`skill`	$\mathbb{N} \cup \{0, +\infty\}$	min	max	max	min	max	min
Maximal probability for the proponent	`prob`	$[0, 1]$	max	$\cdot$	$\cdot$	max	$\cdot$	max
Minimal time for the proponent	`time`	$\mathbb{N} \cup \{0, +\infty\}$	min	$+$	$+$	min	$+$	min
Satisfiability for the proponent	`satp`	$\{0, 1\}$	$\vee$	$\wedge$	$\wedge$	$\vee$	$\wedge$	$\vee$
Satisfiability	`sat`	$\{0, 1\}$	$\vee$	$\wedge$	$\vee$	$\wedge$	$\star$	$\star$

Definition 19 (Bottom-up evaluation of attributes)**.** *Let* α *be an attribute of attack–defense trees, and let* $A_\alpha = (D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ *be its attribute domain. Given an attack–defense tree* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$*, a basic assignment* β *for* α*, and a node* $v \in V$*, such that* $\mathrm{actor}(v) = \mathrm{s}_T$*,* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$*, and* $\mathrm{ref}(v) = \mathtt{OP}$*, the* value of α at v

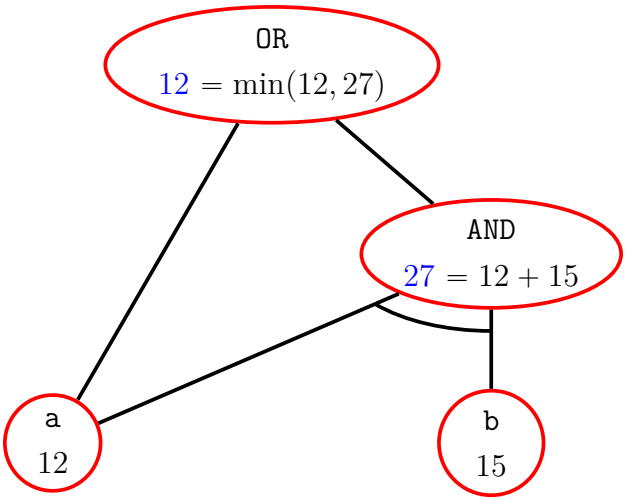


Figure 4: Bottom-up evaluation of the *minimal cost for the proponent* attribute on an attack tree. Values assigned to the basic actions are given in black, values computed at the intermediate nodes – in dark blue.

under β, *denoted by* $\alpha_B(T, \beta, v)$, *is defined recursively as*

$$\alpha_B(T,\beta,v) := \begin{cases} \beta(\lambda(v)), & \textit{if } \texttt{OP} = \texttt{N} \textit{ and } \bar{v} \textit{ does not exist,} \\ \mathtt{C}^{\mathtt{s}}_{\alpha}\left(\beta(\lambda(v)), \alpha_B(T,\beta,\bar{v})\right), & \textit{if } \texttt{OP} = \texttt{N} \textit{ and } \bar{v} \textit{ exists,} \\ (\mathtt{OP}^{\mathtt{s}}_{\alpha})_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v'), & \textit{if } \texttt{OP} \neq \texttt{N} \textit{ and } \bar{v} \textit{ does not exist,} \\ \mathtt{C}^{\mathtt{s}}_{\alpha}\left((\mathtt{OP}^{\mathtt{s}}_{\alpha})_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v'), \alpha_B(T,\beta,\bar{v})\right), & \textit{otherwise.} \end{cases}$$

The value of attribute α for T under β obtained via the bottom–up procedure*, denoted by* $\alpha_B(T, \beta)$*, is then defined as* $\alpha_B(T, \beta, \mathrm{root}(T))$*. In the notation* $\alpha_B(T, \beta)$*, the subscript* B *refers to the "bottom-up" computation.*

An extensive overview of attribute domains and their classification can be found in [KMS12]. The article [BKMS12] contains a case study and guidelines for practical application of the bottom-up procedure. Numerous examples of attributes of attack trees and attack trees extended with additional sequential refinement have been given in [JKM+15] and [HMT17].

The following two examples illustrate the bottom-up evaluation of the `cost` attribute in attack and attack–defense trees.

Example 18. *In Figure 4 a bottom–up evaluation of the* minimal cost for the proponent *attribute, whose domain* $A_{\mathtt{cost}} = (\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \min, +, +, \min, +, \min)$ *has been given in Example 17, is depicted. Since the* `OR` *nodes of the proponent correspond to a choice, the minimal cost is computed at these nodes using the operation of taking the minimum. The addition is used at the* `AND` *nodes of the proponent, as the achievement of a goal of an* `AND` *node requires achieving goals of all of its children.*

Example 19. *In Figure 5 a bottom–up evaluation of the* minimal cost for the proponent *attribute on an attack–defense tree is depicted. Recall that the domain for* minimal cost for the proponent *is* $A_{\mathtt{cost}} = (\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \min, +, +, \min, +, \min)$. *Similar intuition as the one provided in Example 18 supports the choice of operations for the nodes of the opponent: to counter the goal of an opponent's* AND *node, it is sufficient for the proponent to counter any of its child nodes (the proponent has a choice, thus the* min *operation), and to counter the goal of an opponent's* OR *node, all of the children of the node need to be countered (addition).*

The choice of the operations to be performed when the bottom-up evaluation traverses countermeasures is closely related to the values assigned to the basic actions of the opponent. For $\mathtt{C}^{\mathrm{p}}_{\mathtt{cost}} = +$ *and* $\mathtt{C}^{\mathrm{o}}_{\mathtt{cost}} = \min$, *the reasonable values for the actions of the opponent are* 0, *modeling the opponent not executing the action, and* $+\infty$, *modeling the action being executed by the opponent. Note that* $+\infty$ *is both the neutral element for taking the minimum and the absorbing element for the addition, while* 0 *is the neutral element for the addition. In consequence, the values assigned to the actions* not executed *by the opponent do not influence the bottom-up evaluation of* minimal cost for the proponent. *The actions* executed *and the goals achieved by the opponent do, since they either absorb the results of the bottom-up evaluation, yielding* $+\infty$, *modeling the impossibility for the proponent being successful (as it is the case for the* OR *node of the defender in the tree from Figure 5), or else they force the values of countermeasures attached to them to be taken into account (via the* min *operator; as it is the case for the node labeled* $\mathtt{d}_1$ *and the* AND *node of the defender in the tree from Figure 5).*

The discussion from the above paragraph justifies further the choice of the addition being performed at the opponent's OR *nodes. If each of the basic actions of the opponent is assigned either* 0 *or* $+\infty$, *then the value computed at the nodes the goals of which the proponent does not have to counter will be* 0. *Thus, the result of addition at an* OR *node of the opponent corresponds to the cost of countering all the goals of the child nodes of the node that have been achieved by the opponent; the remaining goals are ignored.*

The result of the bottom-up evaluation of minimal cost for the proponent *on the tree in Figure 5 can be therefore interpreted as follows: if the opponent executes actions* $\mathtt{d}_1, \mathtt{d}_2$ *and* $\mathtt{d}_3$, *then the minimal cost of achieving the root goal by the proponent is* 22. *It corresponds to the execution of both actions* a *and* c.

Excluding the *satisfiability* attribute, the attribute domains presented in Table 1 have the following feature in common. Each of them is of the form $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, where $(D_\alpha, \oplus, \otimes)$ is a commutative idempotent semiring. We shall say that such domains are induced by semirings.

Definition 20 (Attribute domain induced by a semiring)**.** *An attribute domain* A_α *is induced by a semiring if* $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ *and* $(D_\alpha, \oplus, \otimes)$ *is a commutative idempotent semiring.*

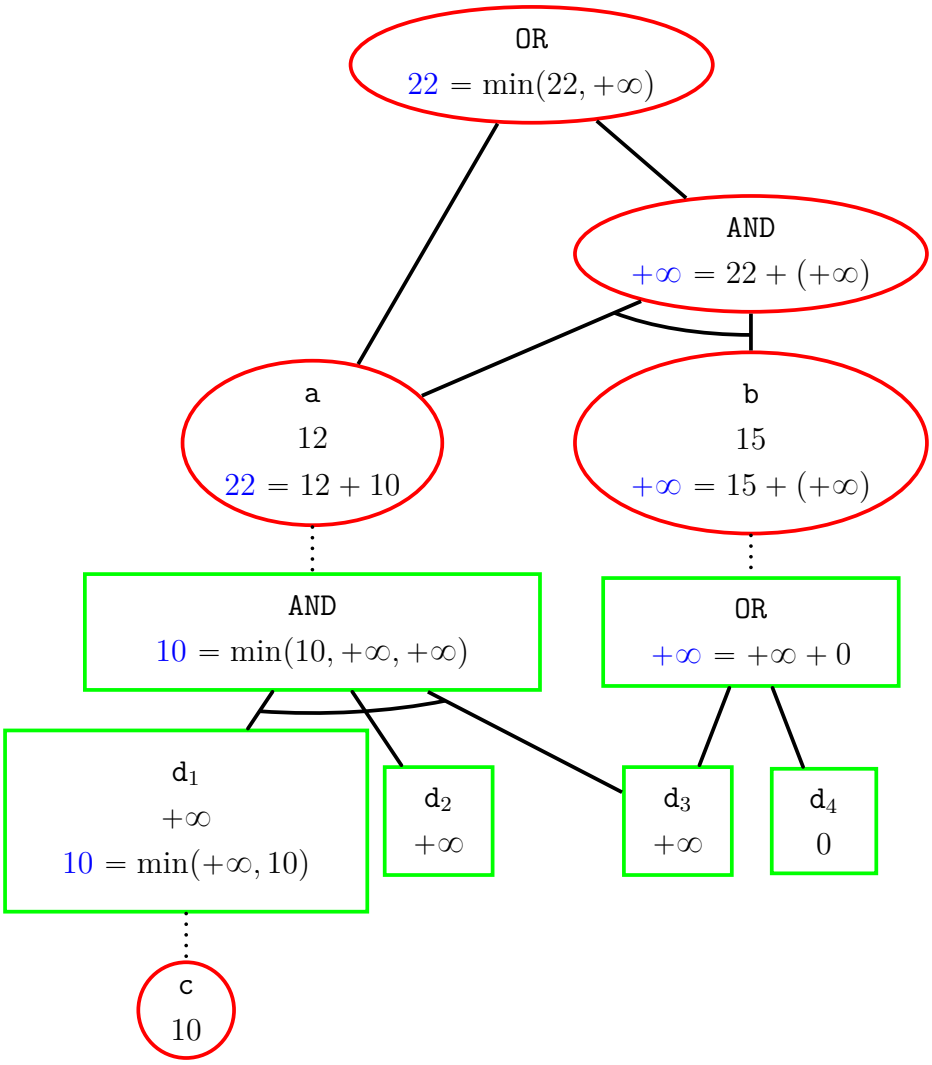


Figure 5: Bottom-up evaluation of the *minimal cost for the proponent* attribute on an attack–defense tree. Values assigned to the basic actions are given in black, values computed at the intermediate nodes – in dark blue

The reasoning behind the choice of the operations for the *minimal cost for the proponent* attribute domain, given in Example 19, can be generalized for attribute domains induced by semirings.

Remark 1. *For a number of attribute domains of the form* $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$*, with* $(D_\alpha, \oplus, \otimes)$ *being a commutative idempotent semiring, under the assumption that a given basic action is executed by the opponent, the value assigned to it is* $\mathtt{a}_\otimes$ *(*$= \mathtt{e}_\oplus$*), whereas the value assigned to the opponent's actions assumed not to be executed is* $\mathtt{e}_\otimes$*. In consequence, the actions not executed by the opponent do not influence the bottom-up evaluation of the attribute, while the executed actions (unless countered by the proponent) absorb the results of the computation corresponding to a given subtree of the tree.*

Example 19 and Remark 1 highlight the particular applicability of the bottom-up evaluation for the so called "what-if" analysis. Being able to compute, say, the *minimal*

cost of a successful attack under the given behavior of the defender can be exploited for selecting an optimal set of countermeasures to be implemented for increasing security of a system.

Attribute domains can also be used for formalizing the intuition behind the notions of refinements and goal achievement. This is usually done using the *satisfiability* attribute domain $A_{\mathtt{sat}} = (\{0,1\}, \vee, \wedge, \vee, \wedge, \star, \star)$, where $x \star y = x \wedge \neg y$, for $x, y \in \{0,1\}$. Under the basic assignment that assigns 1 to each of the actions assumed to be executed by the actors and 0 to the remaining actions, the result of the bottom-up evaluation of `sat` models the root goal of a tree being or not being achieved. In the following definition, we use $\mathbb{1}_X$ for the indicator function of a set $X \subseteq \mathbb{B}$, i.e., a function that assigns one to each of the elements of X, and zero to each of the remaining elements of $\mathbb{B}$.

Definition 21 (Goal achievement)**.** *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree, let $v \in V$ be one of its nodes, and let $X \subseteq \mathbb{B}_T$ be a set of basic actions in T. With* $\mathrm{achieved}_{\mathrm{T}}(v, X)$ *being a shorthand for* $\mathtt{sat}_B(T, \mathbb{1}_X, v)$*, we say that the goal of v is achieved by X in T if* $\mathrm{achieved}_{\mathrm{T}}(v, X) = 1$*.*

Example 20. *Let T be the attack–defense tree from Figure 6 (considered also on Figure 5). Assuming that the attacker executes only the actions* `a` *and* `c` *and the defender executes only* $\mathtt{d}_1, \mathtt{d}_2$ *and* $\mathtt{d}_3$*, which is modeled by assigning* 1 *to each of these actions and* 0 *to the remaining basic actions, the defender fails to counter the action* `a` *(the value computed at the* `AND` *node countering* `a` *is* 0*), and so the attacker achieves the goal of the root node (the value computed at the root node is* 1*). In other words, for the set $X = \{\mathtt{a}, \mathtt{c}, \mathtt{d}_1, \mathtt{d}_2, \mathtt{d}_3\}$ the equality* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), X) = 1$ *holds.*

A careful reader will notice that nothing stops a potential user of attack–defense trees from, e.g., setting contradictory goals as labels of children of an `AND` node in a tree. In such a case, results of any analysis performed on the tree cannot be relied upon. This considers in particular the value of $\mathrm{achieved}_{\mathrm{T}}(\cdot, \cdot)$. In the following, *we assume that the basic actions are independent*, as it is classically done, e.g., in [AN15, GHL+16, AN17]. That is, the only dependency between the basic actions that we allow for, is that an action might be a countermeasure against another action.

Remark 2. *The notion of achievement from Definition 21 is closely related to the notion of* propositional semantics *for attack–defense trees [KMRS14]. For β being a function assigning to every basic action $\mathtt{b} \in \mathbb{B}$ the propositional variable $\beta(\mathtt{b}) = x_{\mathtt{b}}$, the propositional semantics of an attack–defense tree T is the Boolean function $\mathcal{P}(T)$ obtained by the bottom-up propagation of these variables using the operators of the satisfiability domain. Thus, for $X \subseteq \mathbb{B}_T$, the value of* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), X)$ *is equal to the value of $\mathcal{P}(T)$, when the variables corresponding to the basic actions in X are assigned* 1*, and the remaining actions are assigned* 0*.*

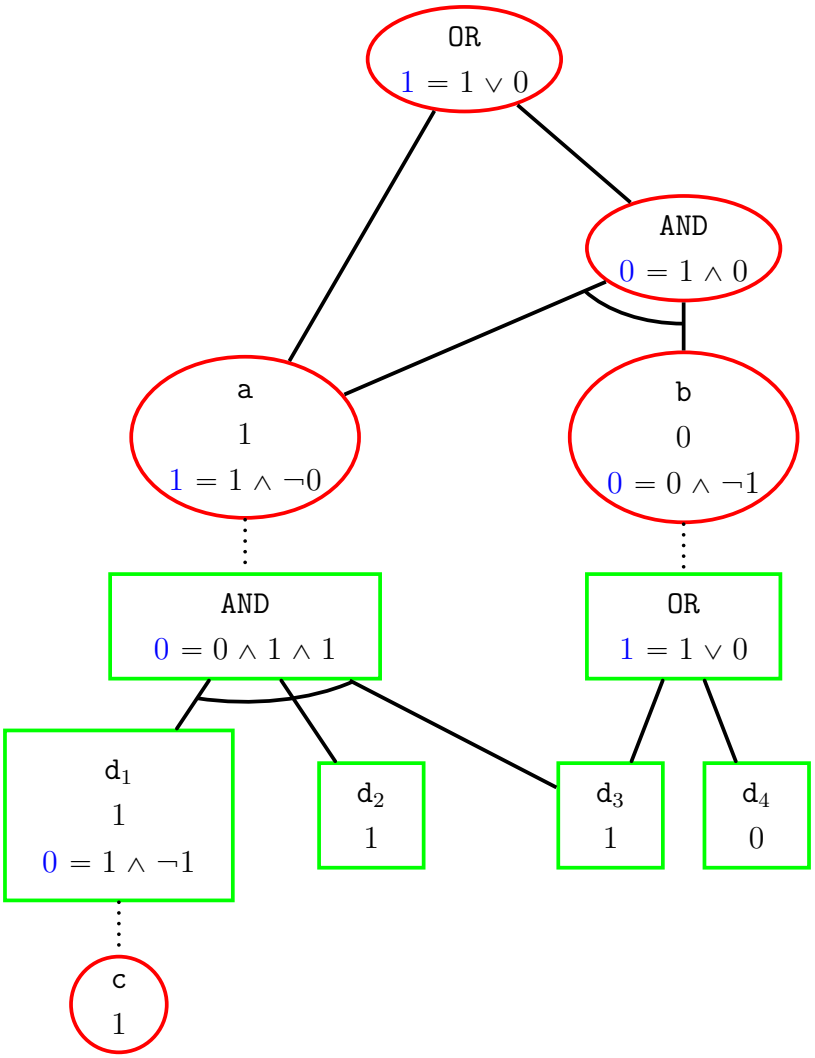


Figure 6: Bottom-up evaluation of *satisfiability* attribute. Values assigned to the basic actions are given in black, values computed at the intermediate nodes using the attribute domain's operations – in dark blue.

Remark 3. *The propositional semantics $\mathcal{P}(T)$ of a tree T, sketched in Remark 2, has been proven in [KPS11] to be a Boolean function positive (respectively, negative) in the variables corresponding to the basic actions of the proponent (respectively, in the variables corresponding to basic actions of the opponent).*

More generally, for a node v of T and a set $X \subseteq \mathbb{B}_T$, the value of $\mathrm{achieved}_{\mathrm{T}}(v, X)$ *is obtained by evaluating a Boolean function that is positive in the variables corresponding to the basic actions of* $\mathrm{actor}(v)$*, and negative in the remaining variables.*

The formalization of the notion of achievement provided in Definition 21 is standard, in the sense that it is widely used, even though under various names, or sometimes under no name at all. For instance, for an attack–defense tree T, a set $P \subseteq \mathbb{B}^{\mathrm{P}_T}$ and a set $O \subseteq \mathbb{B}^{\mathrm{O}_T}$, the authors of [GHL+16] call the value of $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O)$ the "standard boolean semantics" of T. The same expression is used also in [HJL+17], to define the final states of automata that the authors transform attack–defense trees into.

Similarly, the authors of [AN15] call the pair

$$\begin{aligned}&(\min\{\max\{\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) \colon O \subseteq \mathbb{B}^{\mathrm{o}_T}\} \colon P \subseteq \mathbb{B}^{\mathrm{p}_T}\},\\ &\max\{\min\{\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) \colon O \subseteq \mathbb{B}^{\mathrm{o}_T}\} \colon P \subseteq \mathbb{B}^{\mathrm{p}_T}\})\end{aligned} \tag{2}$$

the "boolean semantics evaluation of an attack–defense tree T". Since the authors of [AN15] allow the presence of basic actions that are assumed to be always executed (representing, for example, countermeasures already present in the system), denoting the set of such basic actions in T with X and employing Remark 2 and 3, we note that the pair (2) is equal to

$$(\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), X), \mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O)).$$

As the bottom-up computation simply propagates the values assigned to the basic actions up to the root of the tree, it involves a number of evaluations of the attribute domain's operations that is linear in the size of the tree. Thus, it is generally very fast. On the downside, it may provide unreliable results in the presence of clones. This fact can be easily illustrated with the tree $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{a}, \mathtt{OR}^{\mathrm{p}}(\mathtt{a}, \mathtt{b}))$. Under the basic assignment $\beta(\mathtt{a}) = 5, \beta(\mathtt{b}) = 10$ of the *minimal cost for the proponent* attribute, the result of the bottom up evaluation is $\mathtt{cost}_B(T, \beta) = 5 + \min(5, 10) = 10$. However, to achieve the goal of the root node it is sufficient to execute the basic action $\mathtt{a}$ once, at the cost of 5.

In Chapter 4 we study in detail conditions ensuring that the bottom-up evaluation of attributes yields correct results in trees containing clones. For the case when these conditions are not satisfied, we devise an alternative method of attributes evaluation. Another, heuristic method for the special case of this problem, i.e., for computing the minimal cost of achieving the root goal in attack trees containing clones, is described in Section 3.2.1.

Chapter 3

State of the art

The field of graphical modeling and quantitative analysis of security using attack trees and attack–defense trees is relatively young, but it is developing fast. Numerous approaches are being adapted for improving the applicability of trees for the real-life situations, in particular for answering the questions raised in Section 1.3. In this chapter, we give a detailed overview of some of the frameworks that are closely related to our work, and a brief description of other approaches. We focus on three main areas, namely, on

- formal semantics for attack–defense trees (Section 3.1), where the objective is to give a rigorous meaning to an attack tree or attack–defense tree model,
- quantitative analysis of security using attack–defense trees (Section 3.2), and
- approaches to the problem of optimal selection of countermeasures in the security scenarios modeled with attack–defense trees (in Section 3.3).

It is of course impossible to cover the whole research field in a single chapter. An interested reader is referred to the survey [KPS14] for an exhaustive state of the art on DAG-based security modeling until the year 2013. Usability aspects, practical applications, and computer tools for graphical security modeling are discussed in [HKCH17]. Further examples of recent developments in the first two of the three areas that we cover in this chapter, as well as their deeper comparison, can be found in the recent survey [WAFP19]. Finally, a detailed overview of approaches to the problem of optimal selection of countermeasures against potential attacks, including some works based on attack trees and attack graphs, is given in [NPMK18].

3.1 Formal semantics for attack–defense trees

Even a small and easily readable attack–defense tree might encode a vast number of possible realizations of the underlying attack–defense scenario, as illustrated by the following example.

Example 21. *Consider an attack tree* $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{b}_2), \ldots, \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{n-1}, \mathtt{b}_n))$. *For* β *being a basic assignment for the* satisfiability *attribute, the value of* $\mathtt{sat}_B(T, \beta,)$ *is equal to*

$$(\beta(\mathtt{b}_1) \vee \beta(\mathtt{b}_2)) \wedge \ldots \wedge (\beta(\mathtt{b}_{n-1}) \vee \beta(\mathtt{b}_n)).$$

A simple proof by induction shows that for n *being an even natural number the number of assignments for which the above formula evaluates to* 1 *is* $3^{n/2}$. *Thus, there are* $3^{n/2}$ *sets of basic actions of the proponent that achieve the goal of the root of* T.

Formal analysis of possible realizations of the modeled scenario (which need not be the basic assignments under which the root goal of the tree is achieved, as it is the case in Example 21) is made possible by formally specifying what is considered to be such realization. This is achieved by defining a formal semantics for attack–defense trees. Transforming attack–defense trees into objects modeling realizations of the underlying scenario, such as propositional formulæ or automata, helps addressing a wide range of problems, including enumerating all ways in which the root goal of the tree can be achieved [KMRS14], checking whether two structurally different trees represent the same security scenario [MO05, KMRS14, HMT17], comparing whether one tree contains more information than another one [MO05, KMRS14, HMT17], identifying paths in the analyzed system that correspond to potential attacks [APK18], and verifying the quality of the tree refinements [APK17].

In this section, we recall some of the existing semantics for attack–defense trees. Our goal is to illustrate possible approaches to the problem of interpretation of attack–defense trees and to highlight their advantages and disadvantages.

3.1.1 Multiset semantics

One of the first semantics introduced for attack–defense trees is the *multiset semantics*. Formalized for attack trees by Mauw and Oostdijk in [MO05], generalized for attack–defense trees in [KMRS14] by Kordy et al., and used for the purpose of threat analysis of ATMs in [FFG+16], it interprets attack–defense trees as sets of pairs of multisets.

The definition of the multiset semantics for attack–defense trees employs the operation defined for sets of pairs of multisets of basic actions $X_1, \ldots, X_k \subseteq \mathfrak{M}(\mathbb{B}) \times \mathfrak{M}(\mathbb{B})$ as

$$\bigcirc_{i=1}^{k} X_i := \{(\biguplus_{i=1}^{k} P_i, \biguplus_{i=1}^{k} O_i) \mid (P_i, O_i) \in X_i\}. \tag{3}$$

Definition 22 (Multiset semantics)**.** *Let* T *be an attack–defense tree and let* $\mathcal{M}$ *be the attribute specified by the attribute domain* $A_{\mathcal{M}} = (\mathfrak{M}(\mathbb{B}) \times \mathfrak{M}(\mathbb{B}), \cup, \circledcirc, \circledcirc, \cup, \circledcirc, \cup)$. *Let* β *be the basic assignment of* $\mathcal{M}$ *defined as*

$$\beta(\mathtt{b}) = \begin{cases} \{(\{|\mathtt{b}|\}, \emptyset)\}, & \textit{if } \mathrm{actor}(\mathtt{b}) = \mathrm{p}_T, \\ \{(\emptyset, \{|\mathtt{b}|\})\}, & \textit{otherwise.} \end{cases}$$

The multiset semantics of T, denoted $\mathcal{M}(T)$, is the result $\mathcal{M}_B(T, \beta)$ of the bottom-up evaluation of $\mathcal{M}$ on T under the basic assignment β.

Each of the elements of the multiset semantics is of the form (P, O), where P is a multiset of basic actions of the proponent, and O is a multiset of basic actions of the opponent. Intuitively, as stated in [KMRS14], "A bundle [pair] (P, O) [belonging to the multiset semantics of a tree] encodes how the proponent can achieve his goal [goal of the root node]: the proponent must perform all actions present in P while the opponent must not perform any of the actions in O." The multiset semantics is created by propagating sets of pairs of multisets of basic actions of the actors up to the root of the tree, combining them along the way using appropriate operations. Since the achievement of a goal of an `OR` node requires that at least one goal of its child nodes is achieved, the sets' union is performed; as achieving the goal of an `AND` node is possible only by achieving goals of all of its children, the different ways of achieving these goals are combined using the $\circledcirc$ operation. Similar reasoning as the one given in Example 19 motivates the choice of the remaining operations.

Example 22. *The multiset semantics $\mathcal{M}(T)$ of the tree T in Figure 2 is*

$$\begin{aligned}\mathcal{M}(T) = \big\{&(\{|\texttt{force}, \texttt{card}, \texttt{cash}|\}, \emptyset),\\ &(\{|\texttt{cam}, \texttt{eav}, \texttt{card}, \texttt{cash}|\}, \emptyset),\\ &(\{|\texttt{eav}, \texttt{card}, \texttt{cash}|\}, \{|\texttt{cover}|\}),\\ &(\{|\texttt{phish}, \texttt{phish}, \texttt{log\&trans}|\}, \{|\texttt{sms}|\}),\\ &(\{|\texttt{phish}, \texttt{uname}, \texttt{log\&trans}|\}, \{|\texttt{sms}|\}),\\ &(\{|\texttt{phish}, \texttt{pwd}, \texttt{log\&trans}|\}, \{|\texttt{spwd}, \texttt{sms}|\}),\\ &(\{|\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}|\}, \{|\texttt{spwd}, \texttt{sms}|\}),\\ &(\{|\texttt{phish}, \texttt{phish}, \texttt{phone}, \texttt{log\&trans}|\}, \emptyset),\\ &(\{|\texttt{phish}, \texttt{uname}, \texttt{phone}, \texttt{log\&trans}|\}, \emptyset),\\ &(\{|\texttt{phish}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}|\}, \{|\texttt{spwd}|\}),\\ &(\{|\texttt{uname}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}|\}, \{|\texttt{spwd}|\})\big\}.\end{aligned}$$

As illustrated in the above example by the pair $(\{|\texttt{phish}, \texttt{phish}, \texttt{log\&trans}|\}, \{|\texttt{sms}|\})$, the multiset semantics does not interpret repeated basic actions as clones. The meaning of the pair $(\{|\texttt{phish}, \texttt{phish}, \texttt{log\&trans}|\}, \{|\texttt{sms}|\})$ is the following: if the opponent does not perform `sms` action (transfer dispositions are not secured with two-factor authentication using mobile phone text messages), then the proponent can steal money from the opponent's account by executing the `phish` action *twice*, and by performing the `log&trans` action. Thus, the multiset semantics could be employed for analysis of attack–defense trees if repeated basic actions are not interpreted as clones (cf. the second of the two interpretations of repeated basic actions given on page 14).

A modification of the multiset semantics that could be a viable option for analysis of attack–defense trees under the clones interpretation of repeated basic actions has been proposed in [BK17]. We present it in the next section.

3.1.2 Set semantics

The set semantics for attack–defense trees has been first defined by Bossuat and Kordy in [BK17], for the purpose of interpretation of repeated basic actions as clones. It is a simple adaptation of the multiset semantics, where multisets are replaced with sets. Its definition employs the operation defined for sets of pairs of sets of basic actions $X_1, \ldots, X_k \subseteq 2^{\mathbb{B}} \times 2^{\mathbb{B}}$ as

$$\bigodot_{i=1}^{k} X_i := \{(\bigcup_{i=1}^{k} P_i, \bigcup_{i=1}^{k} O_i) \mid (P_i, O_i) \in X_i\}. \tag{4}$$

Definition 23 (Set semantics)**.** *Let T be an attack–defense tree and let $\mathcal{S}$ be the attribute specified by the attribute domain $A_{\mathcal{S}} = (2^{2^{\mathbb{B}} \times 2^{\mathbb{B}}}, \cup, \odot, \odot, \cup, \odot, \cup)$. Let β be a basic assignment of $\mathcal{S}$ defined as*

$$\beta(\mathtt{b}) = \begin{cases} \{(\{\mathtt{b}\}, \emptyset)\} & \text{if } \mathtt{b} \in \mathbb{B}^{p_T}, \\ \{(\emptyset, \{\mathtt{b}\})\} & \text{otherwise.} \end{cases}$$

The set semantics of T, denoted $\mathcal{S}(T)$, is the result $\mathcal{S}_B(T, \beta)$ of the bottom-up evaluation of $\mathcal{S}$ on T under the basic assignment β.

The intuition behind the set semantics is similar to the one behind the multiset semantics: the presence of a pair (P, O) in the set semantics $\mathcal{S}(T)$ means that if the proponent executes all the actions from P and the opponent executes none of the actions from O, then the root goal of T is achieved. The choice of the operations performed when creating the set semantics is dictated by the same reasoning as in the case of the multiset semantics.

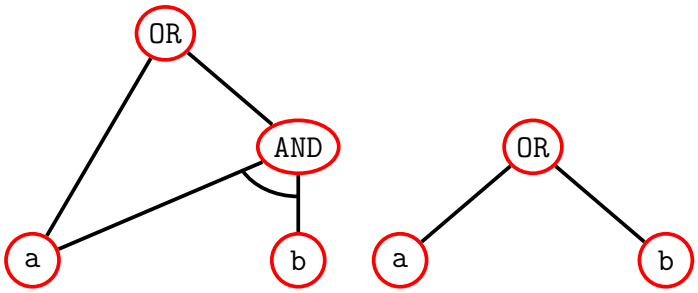
Example 23. *The set semantics $\mathcal{S}(T)$ of the tree T in Figure 2 is*

$$\begin{aligned}\mathcal{S}(T) = \big\{&(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset),\\ &(\{\texttt{cam}, \texttt{eav}, \texttt{card}, \texttt{cash}\}, \emptyset),\\ &(\{\texttt{eav}, \texttt{card}, \texttt{cash}\}, \{\texttt{cover}\}),\\ &(\{\texttt{phish}, \texttt{log\&trans}\}, \{\texttt{sms}\}),\\ &(\{\texttt{phish}, \texttt{uname}, \texttt{log\&trans}\}, \{\texttt{sms}\}),\\ &(\{\texttt{phish}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{spwd}, \texttt{sms}\}),\\ &(\{\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{spwd}, \texttt{sms}\}),\\ &(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset),\\ &(\{\texttt{phish}, \texttt{uname}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset),\\ &(\{\texttt{phish}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{spwd}\}),\\ &(\{\texttt{uname}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{spwd}\})\big\}.\end{aligned}$$

Note that the pair $(\{|\texttt{phish}, \texttt{phish}, \texttt{log\&trans}|\}, \{|\texttt{sms}|\})$, that belongs to the multiset semantics of the tree from Example 23 became the pair $(\{\texttt{phish}, \texttt{log\&trans}\}, \{\texttt{sms}\})$ under the set semantics interpretation. That is, employing sets instead of multisets results in no repetitions of basic actions in the elements of the set semantics. While the set semantics seems fit for analyzing attack–defense trees containing clones, we note that it should not be interpreted in the same way as the multiset semantics. What we mean by this, is that, while for a pair $(P, O) \in \mathcal{M}(T)$, "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 [emphasis added]," the "must" and "must not" no longer applies in the case of a pair (P, O) belonging to the set semantics $\mathcal{S}(T)$. This fact can be illustrated with the two trees $T_1 = \texttt{OR}^{\text{p}}(\texttt{a}, \texttt{AND}^{\text{p}}(\texttt{a}, \texttt{b}))$ and $T_2 = \texttt{OR}^{\text{p}}(\texttt{a}, \texttt{b})$, depicted in Figure 7. While the notion of achievement is not explicitly formalized in [BK17], its informal description is equivalent with the one we provided in Section 1.2. Following this description, execution of both actions a and b in both T_1 and T_2 results in the root goal being achieved, and in none of the two trees both actions are necessary; executing only the action a suffices. However, the set semantics of the trees T_1 and T_2 are

$$\begin{aligned}\mathcal{S}(T_1) &= \{(\{\texttt{a}\}, \emptyset), (\{\texttt{a}, \texttt{b}\}, \emptyset)\},\\ \mathcal{S}(T_2) &= \{(\{\texttt{a}\}, \emptyset), (\{\texttt{b}\}, \emptyset)\},\end{aligned}$$

i.e., the pair $(\{\texttt{a}, \texttt{b}\}, \emptyset)$ belongs to the set semantics of T_1, but not to the set semantics of T_2. While one could argue that $(\{\texttt{a}, \texttt{b}\}, \emptyset)$ is indeed one of the possible realizations of the scenario modeled with the tree T_1, as it represents a way of achieving the goal of the AND node, we believe that the information provided by this pair is redundant. This is the case, because the fact that $(\{\texttt{a}, \texttt{b}\}, \emptyset)$ achieves the root goal follows immediately from the fact that the pair $(\{\texttt{a}\}, \emptyset)$ achieves it.



(a) $T_1 = \mathtt{OR}^{\mathrm{p}}(\mathtt{a}, \mathtt{AND}^{\mathrm{p}}(\mathtt{a}, \mathtt{b}))$ (b) $T_2 = \mathtt{OR}^{\mathrm{p}}(\mathtt{a}, \mathtt{b})$

Figure 7: Two attack trees in which execution of both actions a and b achieves the goal of the root node

To provide an intuitive grasp on the contents of the set semantics, we study its properties in Section 4.2. The established properties allow for using the results of the *evaluation of attributes on the set semantics* as a reference point for the results obtained via other methods, as discussed in detail in Chapter 4.

3.1.3 SP semantics

Neither the multiset nor the set semantics interpretation of a tree provides information on the order in which actions should be executed by the proponent so that the root goal can be achieved. The problem of ordering actions that compose an attack has been apparent for attack trees since their introduction in 1999. It is also visible in the attack–defense tree in Figure 2: while it does not matter whether the attacker first learns the victim's password or the user name, they need to learn both pieces of information before being able to log in to the online banking system.

In the attack tree literature, the problem of ordering actions in attack trees has been addressed in two ways: either AND is implicitly interpreted as an ordered operator, or an extra sequential refinement, that we call SAND and depict with an arrow, is added to capture that some actions must be executed in a specific order. While some works focused on the problem of ordering actions composing an attack existed before [JW09, PB10], it was not until the publication of [AHPS14, KRS15] and [JKM+15] that a formal semantics for attack trees containing SAND refinement[1] has been given. In [AHPS14, KRS15] and [JKM+15], basic actions are assigned mathematical objects (*cumulative distribution functions*, *priced timed automata* and *series-parallel graphs*, respectively), and the object corresponding to the whole tree is obtained from such an assignment using a bottom-up evaluation. Here, we focus on the *SP semantics* of Jhawar et al., introduced in [JKM+15], as it is closely related to the multiset semantics.

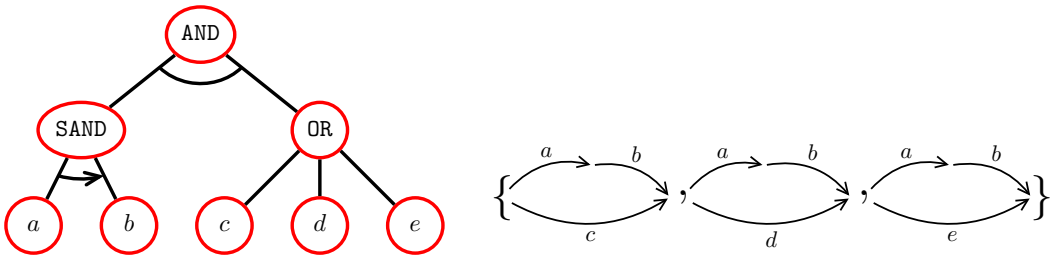
The objective of [JKM+15] is to provide mathematical foundations of attack trees extended with the SAND refinement, called SAND attack trees. To do so, the authors

[1] Called SEQ in [AHPS14].

introduce a formal semantics for SAND attack trees, based on *series-parallel graphs* (SP graphs), and extend the bottom-up method for quantitative analysis from classical attack trees formalized in [MO05] to SAND attack trees.

SAND attack trees considered in [JKM+15] use three types of refinements: OR, AND, and SAND. They thus allow to distinguish between actions that can be executed in parallel (connected with AND) from those that need to be executed sequentially (connected with SAND). To formally interpret SAND attack trees, Jhawar et al. use SP graphs. SP graphs are oriented, edge-labeled graphs that contain two distinct nodes – a *source* with no incoming edges, and a *sink* with no outgoing edges – and that can be built in a recursive way from smaller SP graphs, using their *parallel* and *sequential compositions*. The parallel composition glues two SP graphs by identifying their sinks and their sources, respectively. The sequential composition attaches the second SP graph to the first one, by identifying the sink of the first one with the source of the second one.

The semantics developed in [JKM+15], called the *SP semantics*, interprets an SAND attack tree as a set of SP graphs whose edges are labeled with the basic actions of the attacker. The semantics is created in a bottom-up manner, similarly to the multiset and set semantics. Each of the nodes labeled with basic actions, i.e., each of the leaves of the tree, is interpreted as an SP graph consisting of a single edge, labeled with that action. The parallel and sequential compositions are used to interpret the AND and SAND refinements, respectively. OR refinements are simply interpreted as the union of the sets of SP graphs corresponding to their children. Each SP graph belonging to the set of SP graphs interpreting a tree corresponds to a way of achieving the goal of the root of the tree. An example of a SAND attack tree and its SP semantics is given in Figure 8.



(a) A SAND attack tree (b) The SP semantics of the tree from Figure 8a

Figure 8: The SP interpretation of an SAND attack tree

The SP semantics is a conservative extension of the multiset semantics for classical AND/OR attack trees of [MO05]. The SP semantics equips the multisets of the multiset semantics with a partial order encoding which of the actions need to be performed sequentially.

3.1.4 Path semantics

The goal of the work presented in [APK17] by Audinot et al. is to verify the *correctness* of an `OR/AND/SAND` attack tree with respect to the analyzed system represented as a *transition system*. In this paper, the authors introduce a novel way of labeling the attack tree nodes and a new semantics for attack trees which is based on paths in the underlying transition system. This allows them to define four correctness properties describing how well the children of an attack tree node refine the node's goal, in the context of a given system. The paper establishes the theoretical complexity of checking the introduced correctness properties.

Audinot et al. use transition systems to model real-life systems. A transition system [Kel76] is an operational state-transition model with non–deterministic transitions. In [APK17], the states of the transition system are labeled with *propositions* that express possible configurations of the real–life system, and the transitions correspond to the actions of the attacker. Attack trees considered in this work make use of the same set of propositions as the underlying transition system. Each node of an attack tree is labeled with a so called *goal*, expressed with the help of two propositions: the *initial configuration* representing the situation before the node's attack starts (preconditions), and the *final configuration*, describing the situation to be reached (postconditions). These pre– and postconditions characterize the states of the transition system from which the attacker can start and where they can end their attack. The nodes' goals are not necessarily independent.

Contrary to the existing formalizations of attack trees, the semantics of the trees considered by Audinot et al. relies on *paths* in the underlying transition system and not on the collection of the attacker's actions. The semantics of a node is defined as a set of paths in the transition system linking a state where the initial configuration of the node's goal is satisfied with a state where the final configuration is valid. The semantics of a disjunctive (`OR`), conjunctive (`AND`), and sequential (`SAND`) composition of nodes is defined using respectively the union, the parallel composition, and the concatenation of the paths belonging to the semantics of its components. For instance, a conjunctive composition of several goals is realized if there is a path that can be decomposed into (possibly overlapping) paths that realize each of these goals. Such a view disallows any kind of parallelism in the execution model.

Table 2: Complexity of correctness checking of [APK17]

	meet	under-match	over-match	match
`OR`	P	P	P	P
`SAND`	P	P	P	P
`AND`	NP-c	co-NP-c	co-NP	co-NP

The correctness of an attack tree refinement is then defined by comparing the se-

mantics of a parent node with the semantics of its refinement, i.e., the semantics of the combination of its children using the parent node's operator. The following four correctness properties are introduced: *meet* – when the intersection between the node's semantics and the semantics of its refinement is non-empty; *under-match* – when the semantics of the refinement is included in the semantics of the parent node; *over-match* – when the semantics of the node is included in the semantics of its refinement; and *match* when the semantics of a node is equal to the semantics of its refinement. The complexity of verifying the four correctness properties is summarized in Table 2. The verification procedures have been implemented in the ATSyRA Studio tool [ats18].

The authors of [APSW18] follow-up on the work initiated in [APK17] by providing tight bounds for the complexity of deciding the non-emptiness of the path semantics of an attack tree. The non-emptiness problem is shown to be NP-complete for arbitrary attack trees, and NL-complete for attack trees without `AND` refinements.

3.1.5 Sequence semantics

A common approach in the attack trees literature is to provide a reader with an intuitive explanation of the rules of goals' achievement, and then to proceed directly with defining a semantics for trees (as, e.g., in the works described in Section 3.1.1 – 3.1.3 or in [KRS15]). The achievement rules are rarely formalized independently of a semantics; rather, the starting point is a semantics, and it is assumed that the semantics does describe the ways in which the root goal of a tree can be achieved. In the case of semantics involving basic actions of the attacker, e.g., the multiset semantics or the SP semantics, further relations between the root goal and the attacks present in the semantics are almost never studied. Such relations are the main focus of Mantel and Probst in [MP19]. The authors of [MP19] aim at introducing a framework in which numerous connections between the attacks and the root goal of an attack tree can be formally specified. Relying on a description of the system under consideration and formalization of the attacker's goals using propositional formulæ, the authors provide means for defining various criteria for attacks to be successful in scenarios modeled with trees.

The trees considered in [MP19] are `SAND` attack trees, in which the leaf nodes represent basic actions of the attacker. Any finite, non-empty sequence of the attacker's actions is an attack; attacks relevant to a given tree are gathered in its semantics, which we will call *sequence semantics*[2]. The sequence semantics of a `SAND` attack tree is created using a bottom-up procedure similar to the one used for the SP semantics. The semantics of a leaf node is a singleton consisting of a sequence whose only element is the node's label (attacker's action). Attacks belonging to the sequence semantics of an `OR` node are the attacks that belong to the semantics of at least one of the node's child nodes. The semantics of an `AND` node is obtained by *interleaving* the attacks belonging to the

[2]This will make referring to this particular semantics easier. We note that the authors of [MP19] do not give the semantics that they define any name.

semantics of the node's child nodes. That is, an attack belongs to the sequence semantics of an `AND` node if it can be partitioned into subsequences, in such a way that each of the subsequences belongs to the sequence semantics of one of the child nodes of the node. Finally, the attacks in the semantics of the child nodes are *concatenated* in order to obtain attacks in the semantics of their parent `SAND` node.

To analyze connections between the attacks in the sequence semantics and the goals that the nodes of the tree are labeled with, the authors of [MP19] rely on a description of the system in the context of which the tree is analyzed. The minimal viable description of the system is a *set of states*, with *state* being a function that assigns *values* to the system's *locations*. The attacker's goals are then modeled with propositional formulæ parameterized by locations. Intuitively, a goal modeled with a formula is achieved, if the system is in a state in which this formula is satisfied. Finally, the possible interactions of the attacker with the system (realizations of the security scenario) are modeled as sequences of alternating states and the attacker's actions, starting with a state. In the remainder of this section we will call such sequences *traces*.

Formalization of the attacker's goals using propositional formulæ allows for specifying what does it mean for the goal to be *achieved in a trace* (that is, in a particular realization of the scenario), and for introducing two types of attack occurrences in a trace. Arguing that an occurrence of an attack in a trace and a goal being achieved by a trace constitute the minimal sensible criterion for an attack being successful w.r.t. the attacker's goal, Mantel and Probst specify three degrees of freedom in defining a success criterion. Called *purity*, *persistence*, and *causality*, these degrees allow for making the definition of achieving the root goal more specific. For instance, one could consider an attack occurring in a trace to be successful only if no other actions are executed in between the executions of the actions belonging to the attack (high degree of *purity*), or if, once satisfied in some state in a trace, the root goal remains satisfied in each of the following states (high degree of *persistence*).

The whole framework sketched above relies on a formal description of a system. Such description is the starting point in some of the existing approaches for semi-automatic generation of attack trees (e.g., [VNN14, IPHK15]; see also [WAFP19] for an overview of methods for attack trees generation). A combination of such approaches with the framework developed in [MP19] could be the first step in a meaningful methodology for analysis of security with the help of attack trees, which could be followed, e.g., by application to the created model some of the existing methods for quantitative analysis of trees. We finish this section with noting that the sequence semantics does not interpret repeated basic actions as clones. Furthermore, in contrast to the SP semantics, it does not allow for reasoning about attacks in which some of the actions could be executed in parallel.

3.2 Quantitative analysis of security using attack–defense trees

To fully benefit from the process of security modeling using attack–defense trees, semantic analysis, that, e.g., exhibits possible attacks against a system and highlights its vulnerabilities, should be accompanied by a quantitative analysis of the modeled scenario. One of the common ways of doing this is to employ attribute domains and the bottom-up evaluation of attributes [KMS12, BKMS12, KMRS14, HMT17]. Another way is to transform the tree into another formal object, such as an automaton [GHL+16, HJL+17] or a stochastic two-player game [ANP16], and to perform analysis on the resulting object.

In this section, we provide an overview of some of the attack tree-based methods for quantitative analysis of security. We begin with the works [BLWC17] of Buldas et al. and [AN15] of Aslanyan and Nielson, which are closely related to the analysis framework based on attribute domains. They tackle the same problems that we are interested in: the problem of attributes evaluation in the presence of clones, and the problem of multi-objective quantitative analysis of scenarios modeled with attack–defense trees. We present these works in Section 3.2.1 and 3.2.2, respectively. In the next two sections we give a flavor of the second approach, the one involving transformation of an attack–defense tree into another formal object. An analysis method based on stochastic two-player games is described in Section 3.2.3. In Section 3.2.4 we give an overview of selected works involving stochastic timed automata. Section 3.2.5 is devoted to a narrow description of some of the approaches to the problem of multi-parameter analysis of security using attack–defense trees, and to a comparison between them and the framework that we develop in Chapter 5.

3.2.1 Approximation of the minimal cost of an attack in the presence of clones

The focus of Buldas et al. in [BLWC17] is to provide proofs that for some attack trees no profitable attacks exist. Formally, the problem is addressed by determining whether the cost of a cheapest attack is greater than a given threshold. This is partially achieved by evaluating a lower bound for the cost of a cheapest attack via a combination of a *weight reduction* technique and the bottom-up evaluation of the *minimal cost for the proponent* attribute.

This work considers standard `AND`/`OR` attack trees that might contain repeated basic actions. Attack trees are modeled with *monotone Boolean functions* over propositional variables representing successful executions of particular basic actions by the attacker. An attack in a tree is a minterm of the corresponding formula, i.e., a conjunction of some of the variables that implies the truth of the whole formula. Given a weight function w that assigns non-negative, real values to the propositional variables, the cost of an attack is the

sum of weights of its variables [3]. For a tree Φ, a weight function w, and a profit threshold K, the aim is to determine whether it is profitable for the attacker to execute an attack, i.e., whether the weight of a cheapest attack in Φ, denoted with $w(\Phi)$, does not exceed K. This problem can be formulated in terms of the *weighted monotone satisfiability problem*, which is known to be NP-complete [BLWC17]. To bypass the complexity of this problem, the authors of [BLWC17] propose a method for computing a lower bound for $w(\Phi)$, which is then compared with K. The quality of the obtained lower bound is indicated by the relative error of the method, i.e., by the ratio of the difference between the upper bound and the lower bound to the lower bound.

The lower bound for $w(\Phi)$ is obtained in two steps. First, a weight reduction technique is employed. For every propositional variable x that appears multiple times in the formula Φ, each of its occurrences is replaced with a new variable, and the weight of x is distributed among the new variables, i.e., the sum of weights of the new variables is equal to $w(x)$. Information on how the weights of repeated variables of Φ should be distributed among their occurrences is called a *certificate* for Φ. In the propositional formula obtained after this step, every variable appears exactly once. As we shall prove in Chapter 4, the exact cost of the cheapest attack in this new tree can be obtained via the bottom-up evaluation. This exact cost is computed in the second step of the method. It provides a lower bound for $w(\Phi)$. Furthermore, Buldas et al. prove that if in every subformula of the form $G \wedge F$ of Φ the subformalæ G and F have at most one variable in common, then there exists a certificate for which this lower bound is actually equal to $w(\Phi)$. If the lower bound is greater than the profit K, then it is not profitable for the attacker to conduct an attack.

Once a certificate for Φ is known, it is computationally easy to verify it, that is, to check whether the lower bound for the cost of the cheapest attack in Φ that it provides exceeds K. The choice of a certificate that would achieve the best approximation of $w(\Phi)$ remains problematic. It is worth noting that the exact value of $w(\Phi)$ can be obtained using methods presented in Chapter 4, in a time linear in the number of nodes of a tree and exponential in the number of repeated basic actions. A method for extracting an attack the cost of which is equal to $w(\Phi)$ is also described in Chapter 4.

3.2.2 Pareto efficient strategies in attack–defense trees

In [AN15], Aslanyan and Nielson provide a formal approach to the problem of *multi-parameter optimization* in attack–defense trees. Every set of basic actions of the actors (called *strategy* throughout this section) is assigned a vector $v = (v_1, \ldots, v_k)$ of $k \geqslant 1$ values. Some of the values might represent costs associated with execution of the actions of the proponent that belong to a given strategy. Among them there might also be the probability of the root goal being achieved when the strategy is executed (*probability of*

[3] Expressed in our terminology: an attack in an attack tree is a set of basic actions of the attacker that achieves the goal of the tree's root node; the cost of an attack is the sum of costs of the basic action that constitute it.

success). The aim is to determine the strategies that achieve the root goal and *optimize all of the values at once*. Such optimal strategies are defined in terms of *Pareto efficiency*. A strategy is optimal if its corresponding vector v is Pareto efficient in the set of vectors corresponding to all strategies, i.e., if every other vector that offers an improvement w.r.t. v on at least one coordinate entails a worsening on some other coordinate.

The underlying assumption of the whole framework is the independence of basic actions performed by the actors. The main focus is put on the class of trees that do not contain clones, called *linear trees* by the authors. The basic model of attack–defense trees introduced in [KMRS14] is extended with a negation operator, which allows for capturing the situation in which execution of an action by an actor makes it impossible for them to perform some other action. Aslanyan and Nielson use this operator also for defining a specific class of attack–defense trees, called *polarity–consistent trees* (PCTrees), in which multiple occurrences of basic actions are allowed under some constraints.

Each of the basic actions is assigned two probability values: a probability of achieving the goal it represents in the case of attempted execution, and a probability of achieving the goal in the case when the action is not executed (in the Boolean case, where the problem of satisfiability of the root goal is tackled, these values are 1 and 0, respectively). Furthermore, each of the actions is decorated with a vector $c = (c_1, \ldots, c_m)$ of $m \geqslant 0$ real-valued costs. In this setting, two approaches to the problem of determining Pareto optimal strategies that maximize the probability of success and minimize costs are considered. In the first one, called *semantic evaluation*, the probabilities and costs corresponding to all possible strategies (with the cost of a strategy being a coordinate-wise sum of costs of the actions that constitute the strategy) are computed, and only then the Pareto optimal values are selected. This method has the drawback of high complexity, due to the fact that the number of strategies in an attack–defense tree is exponential in the size of the tree. To overcome this difficulty, the authors of [AN15] develop an alternative method, which they call *algorithmic evaluation*. For the case when $m = 0$, this method is a combination of two standard bottom–up procedures, and determines the lowest and the highest values of probability of success in a linear tree, in the time linear in the size of the tree. Boolean version of this problem is solved similarly in the class of PCTrees. In the Boolean variant the result reflects the influence of the actions of the other actor on the actions of the root actor. For instance, the result can highlight the fact that the root goal is always achieved, no matter what the other actor does, or that the other actor can select actions that ensure that the root goal cannot be achieved by the root actor. The algorithmic evaluation method in the case of $m = 1$, that is, in the presence of both probability and a single cost, propagates up to the root of a tree only the Pareto efficient values. In a linear tree, the result obtained at the root coincides with the result of the semantic evaluation, and, again, is obtained in the time linear in the size of the tree. The computation of the set of Pareto optimal solutions for the probability and cost parameters has been automated in the Attack Tree Evaluator tool (ATE) [Asl16b, Asl16a].

It is worth noticing that the complexity of the algorithmic evaluation increases with the growth of the number m of costs associated with the basic actions, i.e., for any fixed m there is an attack–defense tree T of size linear in m and with the number of unique Pareto optimal strategies exponential in the number of nodes of T. For instance, for an even m, the tree

$$T = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{b}_2), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_3, \mathtt{b}_4), \ldots, \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{m-1}, \mathtt{b}_m))$$

has $n = 3/2\,m + 1$ nodes. Set the probability of successful execution of each of the actions to 1, and, for $i \in \{1, \ldots, m\}$, let the cost of execution of the action $\mathtt{b}_i$ be a vector assuming 1 on the ith coordinate and 0 on each of the remaining $m-1$ coordinates. It is not difficult to see that every set of the form $\{\mathtt{b}_{i_1}, \mathtt{b}_{i_2}, \ldots, \mathtt{b}_{i_{m/2}}\}$, where $i_j \in \{2j-1, 2j\}$, is Pareto optimal, and that the value corresponding to such set is unique. The number of such strategies is $2^{m/2} = 2^{(n-1)/3}$.

In the framework of [AN15] the possible behavior of the actors is described by sets of actions that they execute. This description does not take the order of the actions' execution into account. To additionally capture the order of execution of actions, Aslanyan et al. develop a framework based on stochastic two-player games, in [ANP16] (see Section 3.2.3). Contrary to the approaches for multi-parameter optimization in attack–defense trees based on timed automata (see Section 3.2.4), the methods presented in [AN15] do not capture the possibility of a single action being executed multiple times.

The work of [AN15] served as the main motivation for our research on the multi-parameter optimization in attack–defense trees. In Chapter 5, we provide a general framework for Pareto-based analysis of security scenarios modeled with attack–defense trees. Contrary to the work of Aslanyan and Nielson, our framework allows for optimization of parameters belonging to a wide class, including *maximal probability for the proponent* and *minimal cost for the proponent* attributes, and can be employed for analysis of trees containing clones.

3.2.3 Stochastic game interpretation of attack–defense trees

To overcome the limitations of usual static analysis of scenarios modeled with attack–defense trees, Aslanyan et al. propose a more dynamic approach in [ANP16]. The formalism of attack–defense trees is extended with sequential conjunctive and sequential disjunctive nodes, to capture temporal or causal

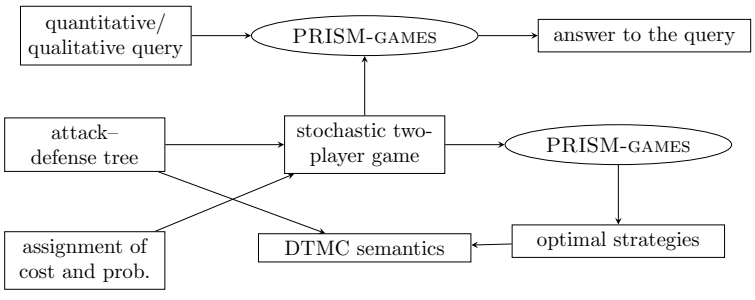


Figure 9: PRISM-GAMES for attack–defense trees by [ANP16]

dependencies between the goals of the actors. With the basic actions being given an assignment of cost of attempted execution and probability of successful execution, the aim is to synthesize strategies for the actors that satisfy given constraints on the two parameters. Intuitively, a strategy provides an actor with information on what actions to perform, as well as in which order or under which circumstances particular actions should be executed. Formally, the strategies are represented as *decision trees*. They are derived from a specific *stochastic two-player game* (STG) [NS03] that the underlying attack–defense tree is transformed into. The whole framework is sketched in Figure 9.

In order to analyze an attack–defense tree, taking the order in which actions are executed into account, the authors of [ANP16] propose a way of transforming the tree into an STG. To explicitly reason about strategies available to the players in the stochastic game, they use probabilistic model checking techniques for stochastic games based on the *probabilistic alternating-time temporal logic with rewards* (rPATL) [CFK+13a]. This allows for expressing and answering questions such as "can the defender ensure that the probability of a successful attack is less than a given threshold?" or "what strategy of the attacker maximizes the probability of a successful attack?". An extension of rPATL [CFK+13b] is employed to synthesize memoryless strategies (or verify their existence) satisfying given constraints on both parameters under consideration, i.e., a bound on the probability of a successful attack and a bound on the expected cost of implementing a strategy by one of the actors. The actual analysis of the game is performed by the PRISM-GAMES tool [KPW16]. Apart from answering the above-mentioned questions, the tool can also present the Pareto optimal strategies (cf. Section 3.2.2; note however, that here the *expected*, and not the *exact*, cost is considered).

Strategies of the actors in an attack–defense tree are intuitively represented using a variant of decision trees. Given a pair of strategies to be implemented by the actors, the possible realizations of the modeled scenario are represented as a *discrete-time Markov chain* (DTMC) [Pri13]. The equivalence between those strategies and the ones originating from the corresponding STG, as well as ways of obtaining the former given the latter, is presented in [ANP16]. Finally, Aslanyan et al. implement a prototype tool that translates an attack–defense tree into a specification of the corresponding STG that is accepted as input by the PRISM-GAMES tool.

The presented framework is developed under the assumption that the sequential nodes present in a tree cannot have non-sequential nodes among their ancestors. For the rest of this section let us refer to a maximal subtree of an attack–defense tree that does not contain sequential nodes as simply *subtree*. We observe that the authors of [ANP16] do not explicitly state the way in which they interpret multiple occurrences of a single basic action in a tree. However, one can deduce from the procedure constructing an STG that multiple nodes labeled with the same basic action and belonging to the same subtree

are interpreted as the same single instance of the action. On the contrary, multiple occurrences originating from different subtrees are interpreted as distinct instances of the action.

From the complexity perspective, the approach of [ANP16] does not manage to escape the state space explosion problem. In the simplest case of an `AND/OR` attack tree with n basic actions, there are $n + 2^n + 3$ states in the resulting stochastic game, which seems to make the framework not usable in practice.

3.2.4 Attack–defense trees analysis with timed automata

The main goal of the framework developed in [GHL+16] is to model temporal behavior of the attacker in an attack–defense tree and to exploit this modeling for the purpose of quantitative analysis of the underlying attack–defense scenario. Gadyatskaya et al. propose a way of encoding the actors and their basic actions as *networks of timed automata* [AD90]. Such a network is then provided as input to the Uppaal model checker [BDL04, LPY97], which allows for extracting strategies of the actors satisfying particular properties, as schematized in Figure 10. The standard model of attack–defense trees with `OR` and `AND` refinements only is considered. The success or failure of the attacker in a tree T, when the attacker has executed set of actions A and the defender the set of actions D, is defined as the value of achieved$_{\mathrm{T}}$(root(T), $A \cup D$).

First, an attack–defense tree is used to derive a directed labeled graph, called by the authors of [GHL+16] an *attack–defense graph*. This graph represents possible realizations of the scenario modeled by the tree, i.e., combinations of all sets of actions executed by the defender with all potential sequences of the actions executed by the attacker. The attack–defense graph is used to define the attacker's profile, which models the capabilities (what are the actions that the attacker can execute and what are the properties of their execution times) and preferences (the probability that a given action is chosen) of the attacker in any situation that can occur in the scenario. Formally, the attacker is modeled as a *timed transition system* [HMP91] equipped with a description of its non-deterministic behavior. The attack–defense graph and the profile of the stochastic attacker are combined to create a stochastic timed transition system that models possible realizations of the scenario.

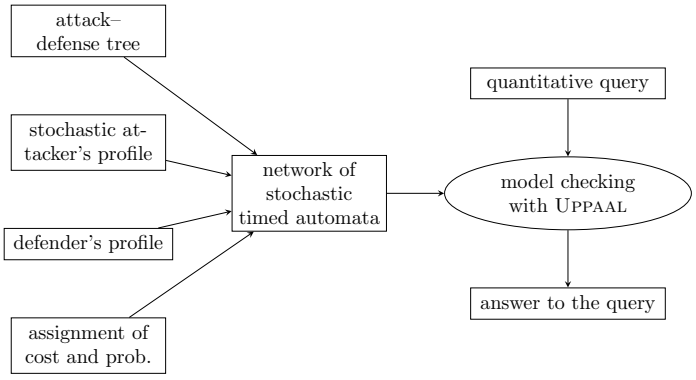


Figure 10: Uppaal-based analysis of attack–defense trees by [GHL+16] and [HJL+17]

Given a set of actions executed by the defender, and taking into account the stochasticity of the attacker, the probability of successful execution of basic actions, and the cost of attempting their execution, Gadyatskaya et al. derive explicit formulæ for the probability of the attacker's success, and the expected cost within a given time bound. This naturally leads to the problem of choosing the attacker's profile that optimizes these values.

The final transition system is encoded using network of *stochastic timed automata*, in a way that ensures that the runs of the network correspond to sequences of transitions in the system. The encoding is performed in a modular manner, i.e., the network consists of an automaton that models the attacker, an automaton modeling the defender, and an automaton for each of the basic actions of the attacker, that models possible outcomes of executing the action. The authors of [GHL+16] implemented the encoding procedure, and the implementation outputs a specification of the network that is accepted as input by the Uppaal model checking engine [BDL04], [LPY97]. Using Uppaal, it is then possible to, e.g., determine the probability of a successful attack or the expected cost of succeeding (for a specific attacker profile) within a given time bound.

The approach from [GHL+16] is expanded upon by Hansen et al., in [HJL+17], with three novelties. First, a dependency between the total cost of execution of an action and the time spent on the execution of the latter is introduced. Instead of being equipped with a real value of cost, as in [GHL+16], every basic action in [HJL+17] is assigned a relative cost of execution per time unit. Second, Hansen et al. formalize a profile of a cost-preserving attacker. The probability of a given action being executed by a cost-preserving attacker depends on the relative cost of the action and the maximal possible time needed for its execution. The lower the impact of the execution of an action on the attacker's budget, the more likely the attacker is to execute the action. Since a cost-preserving attacker might not behave in a way that maximizes the probability of success, a parametrization of such an attacker is proposed. In the case of the parametrized cost-preserving attacker, the probabilities based on the impact of the execution of an action on the attacker's budget are additionally weighted. Finally, a method for selecting a configuration of parameters that minimize the expected cost of an attack in a given tree and under given stochastic defender is proposed. For a given set of configurations of parameters, a number of simulations of the attack–defense scenario is performed for each of the configurations, and the results (costs of success) are subject to analysis of variance. As long as the analysis of the variance detects differences between the sets of results, some of the configurations are being removed, additional simulations are performed for the remaining configurations, and the results are tested again. When no differences are detected, the results of the simulations are assumed to originate from identically distributed random variables. In particular, it is assumed that all of the remaining configurations of the parameters yield the same (optimal) expected cost of the attacker being successful within the given time bound.

In order for the results of the analysis proposed in both [GHL+16] and [HJL+17] to

be meaningful, the underlying attack–defense tree should satisfy some properties, which seem to be implicitly assumed. Computations of the probability of the attacker's success rely on the assumption of mutual independence of all basic actions. Furthermore, if the actions under an `AND` node of the attacker can be executed in parallel, this information is lost in the automata interpretation of the tree, since the final behavior of the attacker is represented as a sequence of actions. Finally, it is assumed that every action of the attacker can be executed an unbounded number of times, until it is completed successfully.

3.2.5 Multi-parameter analysis of security using attack–defense trees

The works presented in Section 3.2.2– 3.2.4 provide ways of analyzing security scenarios modeled with attack–defense trees while taking multiple parameters (or attributes) into account simultaneously. In this section, we briefly describe some of the other possible approaches to this task, and compare them with the framework that we develop in Chapter 5. We limit this section to the works concerned with multi-parameter quantitative evaluation and attack–defense trees possibly containing clones.

One way of addressing the problem of multi-parameter quantitative analysis of security using attack tree-based models is to construct an attribute being a combination of several relevant elementary parameters. An example of such an attribute is the *expected outcome* of the attacker, considered by Jürgenson and Willemson in the context of attack trees in [JW08]. The *expected outcome* represents monetary profit of the attacker, expressed in terms of the *gain* of the attacker in case the attack succeeds, the *costs* of the attack, its *success probability*, as well as the *probability of being caught*, and the related *penalties*. Similarly as [BLWC17], this work uses Boolean functions as the underlying formal model of attack trees. The *expected outcome*'s value is computed for all valuations satisfying the Boolean function representing an attack tree, and the solution with the highest value is retained as the outcome that the attacker can get from performing an attack. Since the logical operators used by Boolean functions are idempotent, repeated basic actions are treated in [JW08] as clones. Due to the necessity of checking all relevant valuations, the complexity of the solution from [JW08] is higher than the complexity of the framework presented in Chapter 5.

In [EDRM06], Edge et al. discuss how to combine the *probability*, *expected cost*, and *impact* parameters into a metrics called *risk*. The individual parameters are propagated using the standard bottom-up approach, and the risk at each node of an attack tree is then computed according to the formula $(probability/cost) \cdot impact$. A simple analysis of the bottom-up propagation rules for *probability* and *cost* used in [EDRM06] implies that they are not suited for trees containing clones.

More recently, several approaches exploiting model checking techniques have been proposed to address the problem of multi-parameter quantitative evaluation on attack

tree-based models. The focus of Aslanyan and Nielson in [AN17] is on attack trees with the *exact cost* [4] and the *probability* parameters. Attack trees are transformed into Markov decision processes with reward structure, and erPCTL[5] queries, such as "what is the *maximum probability* of an attack with the *cost* at most c?" are answered using probabilistic model checking. Compared to the framework of Chapter 5, the approach developed by Aslanyan and Nielson deals with two-parameter evaluation (*exact cost* and *probability*) only, and similarly to [EDRM06], it does not seem to be suited for attack trees containing repeated labels.

In [KRS15], Kumar et al. consider attack trees with basic actions decorated with cost structures modeling *time*, *skills*, *damage*, and *difficulty*. Attack trees are translated into priced timed automata which are then given to the Uppaal Cora model checker where they are queried for quantitative properties of interest expressed with weighted computation tree logic (CTL) queries. The objective is to provide an effective way of computing the necessary resources (e.g., *time*, *skills*) and the corresponding attack paths leading to the achievement of the root goal. This solution allows the authors to deal with two-parameter optimization using an iterative procedure. The method is suitable for attack trees with repeated basic actions, but cannot be applied to attack–defense trees and does not tackle the probability attribute. In his Ph.D. thesis [Kum18], Kumar automatizes this procedure with the help of the ATTOP tool [KSR+18], but does not provide time measurements. Interestingly, for the attack tree considered in [KSR+18], having 12 nodes and no repeated basic actions, the authors state that the ATTOP tool needed more than 6 seconds for computing an attack of minimal time, i.e., for performing the first step of the iterative method for determining Pareto optimal attacks. In the light of the results presented in Section 5.3.2, it thus seems that our solution outperforms the method of [KRS15] (on inputs suitable for both methods).

Model checking of attack–defense trees decorated with the *cost* of attempted execution and the *success probability* is the focus of Aslanyan et al. in [ANP16], as detailed in Section 3.2.3. To capture temporal or causal dependencies between the goals of the actors, sequential conjunctive and sequential disjunctive refinements have been added to attack–defense trees to complement the two standard refinements `OR` and `AND`. The expressive power of attack–defense trees from [ANP16] is thus richer than in the case of our work. However, from the perspective of quantitative analysis, our framework of Chapter 5 is more general in a sense, because [ANP16] is limited to the evaluation of two specific attributes only, namely *expected cost* and *success probability*.

In the works described in Section 3.2.4 it is assumed that the attacker may try executing each of their actions several times, until executed successfully, with a certain probability of succeeding, which is not the case in our work. On the other hand, the

[4] The word *exact* is used to mark a difference with *expected cost* often used in the context of attack tree modeling.

[5] erPCTL stands for probabilistic computation tree logic with exact rewards.

Uppaal-based approach of [GHL+16] is tailored to specific attributes, namely *cost*, *probability*, and *time*, whereas the solution that we propose in Chapter 5 can be applied to a wide class of attributes whose domains satisfy the assumptions of Theorem 5.

Among all existing solutions for multi-parameter evaluation of security on attack–defense trees, the approach introduced by Aslanyan and Nielson in [AN15], described in Section 3.2.2, is the closest to our framework of Chapter 5. To the best of our knowledge, this is the only work considering Pareto optimization on attack–defense trees using the bottom-up approach. The advantages of our framework over the approach of [AN15] are that, first, it allows for computing strategies that optimize a number of different parameters, and second, that it can be applied to attack–defense trees containing clones.

3.3 Selection of countermeasures in attack–defense scenarios

Finding an optimal way of protecting a system is crucial from several perspectives. A security expert will mostly be interested in identifying a set of countermeasures that can cover the largest possible part of attack surface. The system owner will rather take an economic point of view and aim at spending on security only as much as it is really necessary. By estimating the cost of an optimal set of countermeasures, the security expert can provide to the system owner an impartial argument about the minimal budget that should be devoted for securing the system.

The optimization criteria of interest for security expert or a system owner can be diverse. On the one hand, they may want to select the countermeasures in such a way that the remaining uncountered attacks are as expensive for the attacker as possible (attacker investment problem), or that the number of countered attacks is maximal (attack coverage problem). Both these problems fall into the class where the objective is to *maximize* a certain function that quantifies possible attacks. On the other hand, the aim could also be to *minimize* the defender's investment under some constraints (defender investment problem). Classically, if one is able to express which countermeasures disable which attacks, the aforementioned problems can be addressed with the help of integer linear programing, see, e.g., [RDR12, Saw13, ZALT19] and references therein.

In this section, we focus mostly on works that aim at extracting the above-mentioned pairs (attack, countermeasure) from attack–defense trees. Similarly as in the previous section, we briefly compare them with our approach to this problem, developed in Chapter 6, in a way that does not require being familiar with the approach. Here, we would like to only emphasize the fact that our method can be applied to any attack–defense tree, which is not the case in any of the works described in the following paragraphs.

In [RKT12], Roy et al. use attack countermeasure trees (ACT), which are attack trees augmented with countermeasure nodes composed of a detective and a mitigating part. They exploit what can be seen as our method of Chapter 6 in the special case of attack

trees in which to each node of the attacker a single non-refined countermeasure node can be attached. In other words, this modeling framework does not allow for nodes of the opponent to be refined or countered. The authors propose a linear programming–based solution to the problem of *minimizing defender's investment* while covering some of the attacks, and the problem of *maximizing the defender's return on investment* (ROI). While the former can be applied to trees containing clones, the latter cannot, as it relies on the bottom-up computation of success probability that is known not to work in trees with clones. The main research focus of [RKT12] is on algorithms for solving the optimization problems, while we mostly concentrate on extracting information on reasonable ways of achieving the root goal and on reasonable behavior countering these ways from trees.

Maximization of the defender's ROI in scenarios modeled with attack–defense trees has also been addressed in [MHM16]. Trees considered by Muller et al. in [MHM16] are assumed to have no clones of the proponent, and, similarly as in [RKT12], the nodes of the opponent can have no children, i.e., they can be neither refined nor countered. The main contribution of [MHM16] is a branch-and-bound algorithm that iterates in a non-naive way over sets of countermeasures that the opponent can implement, in the search of the one that maximizes the value of the opponent's ROI.

The framework described in Section 3.2.3 could also be applied to `AND/OR` attack–defense trees for the purpose of optimal selection of countermeasures. That is, an attack–defense tree could be transformed into a stochastic two-player game, in which an optimal strategy for the defender, corresponding to an optimal set of countermeasures in the modeled scenario, could be synthesized. This method, however, can only be applied to *small* trees: as detailed in the last paragraph of Section 3.2.3, the size of the resulting game is exponential in the size of the tree.

The complexity of our framework described in Chapter 6 originates from the fact that the dependencies between basic actions of the actors are *encoded* in attack–defense trees; they are complex, and to make use of them, one needs to decode them. In the approaches developed for optimal selection of countermeasures in [BCSW06, KLM19], the relations between behaviors of the actors are simple: in [BCSW06], every attacker's actions disables a set of defender's actions, and in [KLM19], every defender's action impacts success probability of each of the attacker's actions. This simplicity allows for an immediate formulation of the optimization problems as bilevel mixed integer programming (MIP) programs [MB90, Woo93], which can be solved using standard methods. We note that, in the light of Definition 21 of goal achievement, the root node of an attack–defense tree being achieved corresponds to a propositional formula being satisfied. The optimization problems considered in Chapter 6 could thus have been expressed as variants of the satisfiability problem, which in turn could be directly encoded as MIP programs [Hoo88, GWH+18]. Since the goals of the actors are conflicting (e.g., the attacker wants to minimize, and the defender wants to maximize the value of the objective function), and the defender is the first one to act, the result of such encoding would be

a bilevel MIP problem resembling the ones considered in [BCSW06, KLM19]. A standard technique for dealing with bilevel programs involves replacing the inner problem with its dual or the dual of its linear relaxation [Woo93, BCSW06, KLM19]. If the inner problem is a linear programming problem or the integrality gap of its linear relaxation is 1, one eventually obtains a single level optimization problem equivalent to the initial one. In our case, the integrality gap is greater than 1, i.e., the difference between the optimal solution of the final program and the optimal solution of the initial one can not be predicted. Therefore, even though this method would allow for omitting the computationally expensive construction of defense semantics, we decided to pursue the approach yielding the exact optimal solutions. We believe that our framework could thus play an important role in assessing performance of heuristic methods for optimal selection of countermeasures in attack–defense trees developed in the future.

Chapter 4

Evaluation of attributes on attack–defense trees with clones

When discussing the bottom-up evaluation of attributes, we have mentioned on page 37 that it might return incorrect results for attack–defense trees containing clones. This is a widely known issue, motivating the work of [BLWC17] and causing some analysis frameworks to be developed under the explicit assumption of trees not having repeated basic actions [AN15, MHM16, KW17]. The difficulty introduced by the presence of clones can be sometimes bypassed by transforming a tree into another object, and performing quantitative analysis on this object. An example of such approach is the method of evaluation of attributes on the set semantics, defined in [BK17].

We begin this chapter with a preliminary Section 4.1, in which the notion of the evaluation of attributes on the set semantics is recalled. In the same section, the *normal form of the bottom-up evaluation*, a useful tool for analyzing parallels between attribute domains, is introduced. In Section 4.2, properties of the set semantics are studied, providing insights into the actual contents of the semantics. We discuss the complexity of the evaluation of attributes on the set semantics, and present conditions under which the result of this evaluation can be quickly obtained using the bottom-up evaluation in Section 4.3. For the case when these conditions are not satisfied, we develop an alternative method for evaluation of attributes. It is described in Section 4.4. Finally, for attributes such as *minimal cost for the proponent* or *maximal probability for the proponent*, we tackle the issue of efficiently extracting the optimal strategies, i.e., the ones achieving the optimal value of an attribute, from attack–defense trees. An algorithm solving this problem is presented in Section 4.5. In Section 4.6, we sketch possible applications of the methods developed in the previous sections in fields related to attack–defense trees. Finally, Section 4.7 is devoted to experimental results highlighting the differences between various evaluation procedures. We conclude in Section 4.8

4.1 Preliminaries

When introducing a new attribute domain and studying its properties, one might wish to examine its relations with another domain. In such a case, valuable insights can be sometimes obtained by comparing the bottom-up evaluations of the two attributes. To make such comparison straightforward, we employ the term rewriting techniques.

Consider an attribute domain $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ with the operations $\oplus$ and $\otimes$ being associative and commutative, and with $\otimes$ distributing over $\oplus$. Let β be a basic assignment for α, and let L be the language generated by the grammar

$$t ::= \beta(\mathtt{b}) \mid \oplus(t, \ldots, t) \mid \otimes(t, \ldots, t), \tag{5}$$

for $\mathtt{b} \in \mathbb{B}$.

Let $\rightarrow$ be a reduction relation on L defined as follows: for $t, t' \in A$, we say that $t \rightarrow t'$ if t' can be obtained from t by replacing the leftmost subterm of t that is in one of the forms $\otimes(t_1, \oplus(t_2, t_3))$, $\otimes(\oplus(t_2, t_3), t_1)$ with $\oplus(\otimes(t_1, t_2), \otimes(t_2, t_3))$ or $\oplus(\otimes(t_2, t_1), \otimes(t_3, t_1))$, respectively. In other words, the first place in which the distributivity rule of the semiring $(D_\alpha, \oplus, \otimes)$ can be applied is identified, and the appropriate rule is applied. Note that this definition implies that for every $t \in L$ there is at most one $t' \in L$ such that $t \rightarrow t'$. Thus, the reduction $\rightarrow$ is locally confluent. It is also easy to see that $\rightarrow$ is terminating. Therefore, by Lemma 2, every element in L has a unique normal form.

Suppose now that during the bottom-up evaluation of α on tree T under the basic assignment β no *evaluation* of the operations takes place at the intermediate nodes, but rather that the *expressions are propagated* up to the root of the tree, eventually yielding an algebraic expression, involving the operators $\oplus$, $\otimes$ and the values assigned to the basic actions. By switching to the prefix notation, the expression becomes a term belonging to the language L. By reducing this term to its normal form and switching back to the infix notation, one obtains an expression of the form

$$\begin{aligned} \alpha_B(T, \beta) = {} & (\beta(\mathtt{b}_1^1) \otimes \beta(\mathtt{b}_2^1) \otimes \ldots \otimes \beta(\mathtt{b}_{k_1}^1)) \oplus \\ & \ldots \\ & \oplus (\beta(\mathtt{b}_1^i) \otimes \beta(\mathtt{b}_2^i) \otimes \ldots \otimes \beta(\mathtt{b}_{k_i}^i)) \oplus \\ & \ldots \\ & \oplus (\beta(\mathtt{b}_1^n) \otimes \beta(\mathtt{b}_2^n) \otimes \ldots \otimes \beta(\mathtt{b}_{k_n}^n)), \end{aligned} \tag{6}$$

where $\bigcup_{i=1}^{n} \bigcup_{j=1}^{k_i} \{\mathtt{b}_j^i\} = \mathbb{B}_T$.

We call the result of the above procedure *the normal form of the bottom-up evaluation* $\alpha_B(T, \beta)$. Note that the attribute domains induced by semirings admit the normal form of the bottom-up evaluation.

Example 24. *Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be an attribute domain induced by a semiring, let T be the attack–defense tree depicted in Figure 5 and let β be a basic assignment*

for α. After the first step of the procedure described in the previous paragraph, that is, after propagating algebraic expressions up to the root of T, one obtains the expression

$$\begin{aligned}\alpha_B(T,\beta) =& \big(\beta(\mathtt{a}) \otimes ((\beta(\mathtt{d}_1) \oplus \beta(\mathtt{c})) \oplus \beta(\mathtt{d}_2) \oplus \beta(\mathtt{d}_3))\big) \oplus \\ & \oplus \Big(\big(\beta(\mathtt{a}) \otimes ((\beta(\mathtt{d}_1) \oplus \beta(\mathtt{c})) \oplus \beta(\mathtt{d}_2) \oplus \beta(\mathtt{d}_3))\big) \otimes \beta(\mathtt{b}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{d}_4) \Big).\end{aligned}$$

Switching to the prefix notation yields term

$$\begin{aligned}\alpha_B(T,\beta) = \oplus \Big(& \otimes \big(\beta(\mathtt{a}), \oplus(\beta(\mathtt{d}_1), \beta(\mathtt{c}), \beta(\mathtt{d}_2), \beta(\mathtt{d}_3))\big), \\ & \otimes \big(\otimes (\beta(\mathtt{a}), \oplus(\beta(\mathtt{d}_1), \beta(\mathtt{c}), \beta(\mathtt{d}_2), \beta(\mathtt{d}_3))), \otimes(\beta(\mathtt{b}), \beta(\mathtt{d}_3), \beta(\mathtt{d}_4))\big) \Big),\end{aligned}$$

which reduces to

$$\begin{aligned}\alpha_B(T,\beta) = \oplus \Big(& \oplus \big(\otimes (\beta(\mathtt{a}), \beta(\mathtt{d}_1)), \otimes(\beta(\mathtt{a}), \beta(\mathtt{c})), \otimes(\beta(\mathtt{a}), \beta(\mathtt{d}_2)), \otimes(\beta(\mathtt{a}), \beta(\mathtt{d}_3))\big), \\ & \otimes \big(\otimes (\beta(\mathtt{a}), \oplus(\beta(\mathtt{d}_1), \beta(\mathtt{c}), \beta(\mathtt{d}_2), \beta(\mathtt{d}_3))), \otimes(\beta(\mathtt{b}), \beta(\mathtt{d}_3), \beta(\mathtt{d}_4))\big) \Big).\end{aligned}$$

Reducing the last term to its normal form and switching back to the infix notation results in the following normal form of $\alpha_B(T,\beta)$:

$$\begin{aligned}\alpha_B(T,\beta) = & \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_1)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{c})\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_2)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_3)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_1) \otimes \beta(\mathtt{b}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{d}_4)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{c}) \otimes \beta(\mathtt{b}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{d}_4)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_2) \otimes \beta(\mathtt{b}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{d}_4)\big) \\ & \oplus \big(\beta(\mathtt{a}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{b}) \otimes \beta(\mathtt{d}_3) \otimes \beta(\mathtt{d}_4)\big).\end{aligned}$$

The next example provides an illustration of the normal form of the set semantics.

Example 25. *Consider again the tree T from Figure 5. Example 24 implies that the normal form of the set semantics $\mathcal{S}(T)$ of T is*

$$\begin{aligned}\mathcal{S}(T) = & \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_1\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\{\mathtt{c}\}, \emptyset)\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_2\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_1\}) \odot (\{\mathtt{b}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\}) \odot (\emptyset, \{\mathtt{d}_4\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\{\mathtt{c}\}, \emptyset) \odot (\{\mathtt{b}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\}) \odot (\emptyset, \{\mathtt{d}_4\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_2\}) \odot (\{\mathtt{b}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\}) \odot (\emptyset, \{\mathtt{d}_4\})\big\} \\ & \cup \big\{(\{\mathtt{a}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\}) \odot (\{\mathtt{b}\}, \emptyset) \odot (\emptyset, \{\mathtt{d}_3\}) \odot (\emptyset, \{\mathtt{d}_4\})\big\}.\end{aligned}$$

The usefulness of the normal form of the bottom-up evaluation will be demonstrated in the remaining sections of this chapter.

To tackle the difficulties in the evaluation of attributes in the presence of clones, Bossuat and Kordy introduced in [BK17] the evaluation of attributes on the set semantics.

Definition 24 (Evaluation of attributes on the set semantics)**.** *Let α be an attribute with the attribute domain $(D_\alpha, \mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha, \mathtt{OR}^{\mathrm{o}}_\alpha, \mathtt{AND}^{\mathrm{o}}_\alpha, \mathtt{C}^{\mathrm{p}}_\alpha, \mathtt{C}^{\mathrm{o}}_\alpha)$ such that the operations $\mathtt{OR}^{\mathrm{p}}_\alpha, \mathtt{AND}^{\mathrm{p}}_\alpha$ and $\mathtt{OR}^{\mathrm{o}}_\alpha$ are associative and commutative. Let T be an attack–defense tree, and let β be a basic assignment for α. The value of α for T under β evaluated on the set semantics, denoted by $\alpha_{\mathcal{S}}(T, \beta)$, is defined as*

$$\alpha_{\mathcal{S}}(T,\beta) := (\mathtt{OR}^{\mathrm{p}}_\alpha)_{(P,O)\in\mathcal{S}(T)} \left(\mathtt{C}^{\mathrm{p}}_\alpha \big((\mathtt{AND}^{\mathrm{p}}_\alpha)_{\mathtt{b}\in P} \beta(\mathtt{b}), (\mathtt{OR}^{\mathrm{o}}_\alpha)_{\mathtt{b}\in O} \beta(\mathtt{b}) \big) \right).$$

In the notation $\alpha_{\mathcal{S}}(T, \beta)$, the subscript $\mathcal{S}$ refers to the computation on the "set semantics".

From now on, we shall call the elements of set semantics *strategies.* The attribute evaluation on the set semantics consists of computing values of the attribute corresponding to particular strategies, and then combining these values using the $\mathtt{OR}^{\mathrm{p}}_\alpha$ operator. This is visible in the following two examples.

Example 26. *Consider an attribute domain $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ induced by a semiring $(D_\alpha, \oplus, \otimes)$. For a tree T and a basic assignment β for α, the evaluation of α on the set semantics of T acquires the form*

$$\begin{aligned} \alpha_{\mathcal{S}}(T,\beta) &= \bigoplus_{(P,O)\in\mathcal{S}(T)} \left(\bigotimes \big(\bigotimes_{\mathtt{b}\in P} \beta_\alpha(\mathtt{b}), \bigotimes_{\mathtt{b}\in O} \beta_\alpha(\mathtt{b}) \big) \right) = \\ &= \bigoplus_{(P,O)\in\mathcal{S}(T)} \bigotimes_{\mathtt{b}\in P\cup O} \beta_\alpha(\mathtt{b}). \end{aligned}$$

Example 27. *Consider the tree T from Figure 2 and the attribute domain $A_{\mathtt{time}} = (\mathbb{N}\cup\{+\infty\}, \min, +, +, \min, +, \min)$ for the* minimal time for the proponent *attribute. Let $\beta_{\mathtt{time}}$ be the basic assignment that assigns $+\infty$ to the basic actions of the opponent and the values given in Table 3 to those of the proponent. The evaluation of* minimal time

for the proponent *for* T *under* $\beta_{\texttt{time}}$ *is*

$$\begin{aligned}\texttt{time}_{\mathcal{S}}(T, \beta_{\texttt{time}}) = \min \big(&10 + 120 + 5,\\ &60 + 360 + 120 + 5,\\ &360 + 120 + 5 + \infty,\\ &100 + 5 + \infty,\\ &100 + 20 + 5 + \infty,\\ &100 + 300 + 5 + \infty + \infty,\\ &20 + 300 + 5 + \infty + \infty,\\ &100 + 20 + 5,\\ &100 + 20 + 20 + 5,\\ &100 + 300 + 20 + 5 + \infty,\\ &20 + 300 + 20 + 5 + \infty\big) =\\ =&125,\end{aligned}$$

where the consecutive elements correspond to the elements of the set semantics of T*, as presented in Example 23 on page 42. Following Remark 1 from page 34, the result means that if the opponent executes all of their actions, then the minimal time needed for achieving the root goal by the proponent is* 125 *units of time. It corresponds to the execution of the strategy* $(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$*.*

Intuitively, the result obtained in Example 27 seems to be correct: the time needed for execution of any set of actions of the proponent that achieves the root goal when the opponent executes all of their actions is at least 125. We note that in this particular case, the bottom-up evaluation fails, as illustrated in the next example.

Table 3: Basic assignment of *time* to the basic actions of the proponent from tree in Figure 2.

Basic action b	$\beta_{\texttt{time}}(\texttt{b})$	**Basic action** b	$\beta_{\texttt{time}}(\texttt{b})$
cam	60	eav	360
force	10	card	120
cash	5	pwd	300
phish	100	uname	20
log&trans	5	phone	20

Example 28. *Consider again the tree, the attribute domain and the basic assignment*

from Example 27. In this setting, the bottom-up evaluation yields

$$\begin{aligned}&\mathtt{time}_B(T, \beta_{\mathtt{time}}) = \\ &\min\Big(\min\big(360 + \min(60, +\infty), 10\big) + 120 + 5, \\ &\min(100, 300 + \infty) + \min(100, 20) + \big(5 + \min(+\infty, 20)\big)\Big) \\ &= \min(135, 145) = 135,\end{aligned}$$

which is the minimal time necessary to achieve the root of T when executing the strategy $(\{\mathtt{force}, \mathtt{card}, \mathtt{cash}\}, \emptyset)$. *It exceeds the actual minimal time, obtained using the evaluation on the set semantics in Example 27, by* 10 *time units. This is the case, because the bottom-up disregards any piece of information about nodes other than value assigned to them. In consequence, the value corresponding to the cloned* `phish` *action has been taken into account twice, in the expressions* $\max(100, 300, +\infty)$ *and* $\min(100, 20)$, *leading to the value of* 125 *time units not appearing in the computations.*

Similarly as it is the case with the bottom-up evaluation of attributes, the evaluation on the set semantics should be applied with care. In particular, the result of this evaluation method is not meaningful for all attributes. This fact is visible in the following example.

Example 29. *Consider the attribute domain* $A = ([0, 1], \circ, \cdot, \circ, \cdot, \bullet, \bullet)$, *where*

$$\begin{aligned}p_1 \circ p_2 \circ \ldots \circ p_n &:= 1 - \prod_{i=1}^{n}(1 - p_i), \\ p_1 \bullet p_2 &:= p_1 \cdot (1 - p_2),\end{aligned}$$

for $p_1, \ldots, p_n \in [0, 1]$ *and* $n \in \mathbb{N}_{\geqslant 1}$. *This domain has been used in [KMS12, AN15] and [EK19] for formalizing the attribute called* success probability, *with the authors of [KMS12] stating explicitly that the bottom-up evaluation of this attribute yields meaningful results only in trees with no dependencies between the basic actions.*

Let $T = \mathtt{OR}^{\mathrm{p}}(\mathtt{a}, \mathtt{AND}^{\mathrm{p}}(\mathtt{a}, \mathtt{b}))$. *Assume that the probabilities of successful execution of actions* `a` *and* `b` *are* $p(\mathtt{a})$ *and* $p(\mathtt{b})$, *respectively. Furthermore, assume that the actions are independent, i.e., that neither an attempted nor a successful execution of any of the two actions impacts the probability of a successful execution of the other one, and that a successful execution of any of them does not cancel the consequences of a successful execution of the other one.*

Following Definition 21 of achievement, there are two sets of actions that achieve the root goal of T: the singleton $\{\mathtt{a}\}$ *and the set* $\{\mathtt{a}, \mathtt{b}\}$. *Thus, if the root goal of T is achieved, then the attacker must have executed successfully either the action* `a` *or else both actions* `a` *and* `b`. *In either case, the attacker must have executed successfully the action* `a`. *On the other hand, since the singleton* $\{\mathtt{a}\}$ *achieves the root goal of T, should the attacker execute the action* `a` *successfully, they will have achieved the root goal of T. It follows*

that the root goal of T is achieved if and only if the basic action $\mathtt{a}$ *is executed successfully. Therefore, it seems plausible to conclude that the probability of the attacker achieving the root goal cannot be greater than* $p(\mathtt{a})$.

However, regardless of the intuitive meaning of the attribute corresponding to the domain A, the result of its evaluation on the set semantics of T is

$$1 - (1 - p(\mathtt{a}))(1 - p(\mathtt{a})p(\mathtt{b})),$$

and it is greater than $p(\mathtt{a})$ *for* $p(\mathtt{a}), p(\mathtt{b}) \notin \{0, 1\}$.

Therefore, for the attribute domain A, turning to the evaluation on the set semantics is not enough to obtain meaningful results in the presence of clones.

The least that one could require from a method of evaluation of attributes, is that it returns the same result for, possibly syntactically different, trees describing the same attack–defense scenario. This requirement being satisfied by a particular attribute and its evaluation on the set semantics can indicate that the results obtained with this evaluation method of the attribute are meaningful. To be more specific, if for a non-trivial attribute domain[1] A_α and any two attack–defense trees T_1 and T_2 satisfying $\mathcal{S}(T_1) = \mathcal{S}(T_2)$ the value of $\alpha_{\mathcal{S}}(T_1, \beta)$ is the same as $\alpha_{\mathcal{S}}(T_2, \beta)$ for any basic assignment β for α, it seems reasonable to expect that the evaluation of α on the set semantics yields meaningful results.

In [KMRS14], where the authors consider any equivalence relation on the set of all terms produced by the grammar (1) to be a semantics for attack–defense trees, this requirement is formalized for the bottom-up evaluation with the notion of *compatibility* of attribute domain with semantics for attack-defense trees. Since the result of any evaluation method of attributes on attack–defense trees relies on the tree under consideration and some additional data, such as basic assignment, we generalize the compatibility notion of [KMRS14] as follows.

Definition 25 (Compatibility with an equivalence relation on $\mathbb{T}$)**.** *Let X be a set and let f be a function on $\mathbb{T} \times X$. For $\equiv$ being an equivalence relation on $\mathbb{T}$, function f is said to be compatible with $\equiv$ if for any two trees T_1, T_2 satisfying $T_1 \equiv T_2$ the equality $f(T_1, x) = f(T_2, x)$ holds for every $x \in X$.*

In particular, we say that a function is *compatible with the set semantics* if it is compatible with the equivalence relation defined on the set $\mathbb{T}$ of all trees by $T_1 \equiv_{\mathcal{S}} T_2$ if and only if $\mathcal{S}(T_1) = \mathcal{S}(T_2)$.

[1] A trivial attribute domain could be, e.g., a domain with the set of values that the attribute can attain being a singleton, and with all six domain operations being the same idempotent operation. The result of evaluation of the corresponding attribute, whether using the bottom-up procedure or evaluation on the set semantics, would be the same for all trees.

Example 30. *Considerations from Example 26 imply that if A_α is an attribute domain induced by a semiring, then the evaluation of α on the set semantics, seen as a function defined on the set $\mathbb{T} \times \{\beta \colon \beta$ is a basic assignment for $\alpha\}$ is compatible with the set semantics.*

While every attribute domain should be examined carefully before employing the evaluation of the corresponding attribute on the set semantics, Example 29 and 30 suggest that the results obtained for the attributes induced by semirings are likely to be meaningful.

4.2 Properties of the set semantics

In order to study the properties of the set semantics, we employ the notion of *minimal strategy*.

Definition 26 (Minimal strategy)**.** *Let T be an attack–defense tree. A minimal strategy in T is a pair $(P, O) \in 2^{\mathbb{B}^{\mathrm{P}T}} \times 2^{\mathbb{B}^{\mathrm{O}T}}$ such that*

1. *if the proponent executes all the actions from P, and the opponent does not perform any of the actions from O, then the root goal of T is achieved, i.e., for every set $O' \subseteq \mathbb{B}^{\mathrm{O}T} \backslash O$ the equality $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O') = 1$ holds,*
2. *all the actions from P need to be executed when O is not performed in order for the root goal to be achieved: should the proponent perform only a nonempty proper subset of P, the opponent could prevent them from succeeding by executing some of the allowed actions. That is, for every nonempty subset $P' \subset P$, there is a set $O' \subseteq \mathbb{B}^{\mathrm{O}T} \backslash O$ such that $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P' \cup O') = 0$,*
3. *none of the actions from O can be performed by the opponent so that the proponent executing P cannot be prevented by the opponent from succeeding: if only a subset O'' of O was forbidden, execution of P could be countered by the opponent. That is, if the set O is not empty, then for every subset $O'' \subset O$, there is a set $O' \subseteq \mathbb{B}^{\mathrm{O}T} \backslash O''$ such that the equality $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O') = 0$ holds.*

Intuitively, the non-minimal strategies are the strategies that do not provide any additional insight into the scenario modeled with an attack–defense tree: for every non-minimal strategy describing a way of achieving the root goal, there is a minimal one from which this description can be deduced.

Example 31. *Consider again the tree $T_1 = \mathtt{OR}^{\mathrm{P}}(\mathtt{a}, \mathtt{AND}^{\mathrm{P}}(\mathtt{a}, \mathtt{b}))$ from Figure 7a on page 44, whose set semantics is*

$$\mathcal{S}(T_1) = \{(\{\mathtt{a}\}, \emptyset), (\{\mathtt{a}, \mathtt{b}\}, \emptyset)\}.$$

The pair $(\{\mathtt{a}\}, \emptyset)$ is a minimal strategy in T_1: the condition $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), \{\mathtt{a}\} \cup \emptyset) = 1$ holds, and the remaining two conditions are vacuously true.

The strategy $(\{\mathtt{a},\mathtt{b}\},\emptyset)$ *is not minimal, as it does not satisfy the second condition of Definition 26. Indeed, for* $P' = \{\mathtt{a}\} \subseteq \{\mathtt{a},\mathtt{b}\}$, *and* $O' = \emptyset$, *which is the only subset of* $\mathbb{B}^{\mathrm{o}_T}\backslash\emptyset$, *the equality* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P' \cup O') = 1$ *holds.*

On the intuitive level, knowing that execution of only the action `a` *is enough for achieving the root goal allows for deducing that executing both* `a` *and* `b` *achieves the root goal, too.*

The next example illustrates the intuition behind the third condition of Definition 26.

Example 32. *Consider the attack–defense tree* $T = \mathtt{C}^{\mathrm{p}}(\mathtt{AND}^{\mathrm{p}}(\mathtt{a},\mathtt{b}), \mathtt{C}^{\mathrm{o}}(\mathtt{d},\mathtt{b}))$. *Its set semantics is*

$$\mathcal{S}(T) = \{(\{\mathtt{a},\mathtt{b}\},\emptyset), (\{\mathtt{a},\mathtt{b}\},\{\mathtt{d}\})\}.$$

The pair $(\{\mathtt{a},\mathtt{b}\},\emptyset)$ *is a minimal strategy in* T. *This is the case, since* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), \{\mathtt{a},\mathtt{b}\} \cup \{\mathtt{d}\}) = 1$, *and for* P' *being any of the sets* $\{\mathtt{a}\}$ *and* $\{\mathtt{b}\}$ *setting* $O' = \emptyset$ *yields* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P' \cup O') = 0$.

The strategy $(\{\mathtt{a},\mathtt{b}\},\{\mathtt{d}\})$ *is not minimal in* T, *as it does not satisfy the third condition of Definition 26. Indeed, for* $O'' = \emptyset$ *the only possible choice of* O' *is* $O' = \emptyset$, *and the equality* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), \{\mathtt{a},\mathtt{b}\} \cup O') = 1$ *holds.*

Intuitively, knowing that the execution of `a` *and* `b` *achieves the root goal regardless of the behavior of the opponent, allows for deducing that the same actions achieve the root goal when the opponent does not execute* `d`.

Finally, we illustrate the notion of the minimal strategy on our running example.

Example 33. *Consider again the tree* T *from Figure 2. As described in Example 23 on page 42 the pairs* $(P,O) = (\{\mathtt{phish},\mathtt{log\&trans}\},\{\mathtt{sms}\})$ *and* $(P',O') = (\{\mathtt{phish},\mathtt{uname},\mathtt{log\&trans}\},\{\mathtt{sms}\})$ *both belong to the set semantics of* T. *It is easy to verify that both pairs satisfy the first condition of Definition 26. Since the set* P *is contained in* P', *the pair* (P',O') *is not a minimal strategy in* T: *it fails to satisfy the second condition of the definition.*

Since there is no opponent in attack trees, each pair belonging to the set semantics of an attack tree has the empty set as its second component. Thus, the elements of the set semantics of an attack tree can be seen as sets of actions. It follows that the minimal strategies in the case of attack trees are the minimal (w.r.t. inclusion) sets of actions of the proponent that achieve the root goal of the tree.

Definition 26 is intentionally verbose, to ensure that it indeed formalizes our intuition behind the minimal strategies. Nevertheless, a simpler characterization of minimal strategies can be derived from it instantaneously. Let T be a tree and let $(P,O),(P',O') \in 2^{\mathbb{B}^{\mathrm{p}}T} \times 2^{\mathbb{B}^{\mathrm{o}}T}$. Assume that $(P,O) \neq (P',O')$, $P' \subseteq P$, $O' \subseteq O$ and that both pairs of sets satisfy the first condition of Definition 26. It is easy to see that if $P' \neq P$, then the pair (P,O) does not satisfy the second condition of Definition 26. Similarly, if $O' \neq O$, then the third condition of Definition 26 is not satisfied by (P,O). This implies the following.

Table 4: Two satisfiability domains for attack–defense trees

attribute α	D_α	$\mathtt{OR}^{\mathrm{p}}_\alpha$	$\mathtt{AND}^{\mathrm{p}}_\alpha$	$\mathtt{OR}^{\mathrm{o}}_\alpha$	$\mathtt{AND}^{\mathrm{o}}_\alpha$	$\mathtt{C}^{\mathrm{p}}_\alpha(x,y)$	$\mathtt{C}^{\mathrm{o}}_\alpha(x,y)$
`satp`	$\{0,1\}$	$\vee$	$\wedge$	$\wedge$	$\vee$	$x \wedge y$	$x \vee y$
`sat`	$\{0,1\}$	$\vee$	$\wedge$	$\vee$	$\wedge$	$x \wedge \neg y$	$x \wedge \neg y$

Corollary 1. *Let T be an attack–defense tree and let $\preceq$ be the partial order defined on the set X of all elements of $2^{\mathbb{B}^{\mathrm{p}_T}} \times 2^{\mathbb{B}^{\mathrm{o}_T}}$ satisfying the first condition of Definition 26 by*

$$(P', O') \preceq (P, O) \text{ if and only if } P' \subseteq P \text{ and } O' \subseteq O.$$

The elements minimal in X w.r.t. the partial order $\preceq$ are the minimal strategies in T.

The above formulation will be useful in studying the properties of the set semantics. We will begin with proving that every pair (P, O) belonging to the set semantics of an attack–defense tree satisfies the first condition of Definition 26, i.e., that for every such pair the root goal is indeed achieved when P is executed and none of the actions from O are executed. Then, we will demonstrate that every minimal strategy in T belongs to the set semantics of T, and that if there are no clones in T, then in fact each of the strategies in T is minimal.

Our proofs rely on the following lemma, which shows that in order to verify whether a set of actions achieves the root goal in a tree, one can use the *satisfiability for the proponent* attribute (abbreviated as `satp`; see Table 4) instead of the *satisfiability* attribute (`sat`). Since, contrary to the latter, the domain of the former is induced by a semiring, this allows for exploiting the normal form of its bottom-up evaluation.

Lemma 3. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree, and let $P \subseteq \mathbb{B}^{\mathrm{p}_T}$, $O \subseteq \mathbb{B}^{\mathrm{o}_T}$. Let $\beta_{\mathtt{sat}}$ and $\beta_{\mathtt{satp}}$ be basic assignments of* satisfiability *and* satisfiability for the proponent *attributes, respectively, defined as*

$$\beta_{\mathtt{sat}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} \in P \cup O, \\ 0 \text{ otherwise,} \end{cases}$$

$$\beta_{\mathtt{satp}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} \in P \text{ or } \mathtt{b} \in \mathbb{B}^{\mathrm{o}_T} \backslash O, \\ 0 \text{ if } \mathtt{b} \in O \text{ or } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T} \backslash P. \end{cases}$$

For every $v \in V$ the following holds.

- *If* $\mathrm{actor}(v) = \mathrm{p}_T$, *then* $\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) = \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v)$,
- *if* $\mathrm{actor}(v) = \mathrm{o}_T$, *then* $\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) = \neg\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v)$.

In particular, $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = \mathtt{satp}_B(T, \beta_{\mathtt{satp}})$.

Proof. The proof is by induction on the structure of the subdag $T(v)$. For the base case, assume that v is a non-refined node and that $\bar{v}$ does not exist. In this case the required equalities follow immediately from definitions of the basic assignments $\beta_{\mathtt{sat}}$ and $\beta_{\mathtt{satp}}$. We now proceed with the remaining cases.

Case 1. The node v is not refined and $\bar{v}$ exists.

If $\mathrm{actor}(v) = \mathrm{p}_T$, then

$$\begin{aligned}\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) &= \beta_{\mathtt{satp}}(\lambda(v)) \wedge \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v}) \\ &= \beta_{\mathtt{sat}}(\lambda(v)) \wedge \neg\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v}) \\ &= \mathtt{sat}_B(T, \beta_{\mathtt{satp}}, v),\end{aligned}$$

where the second of the equalities follows from the definitions of the two basic assignments and the induction hypothesis, and the remaining ones from definitions of the two satisfiability attributes' domains.

Similarly, if $\mathrm{actor}(v) = \mathrm{o}_T$, then

$$\begin{aligned}\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) &= \beta_{\mathtt{satp}}(\lambda(v)) \vee \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v}) \\ &= \neg\beta_{\mathtt{sat}}(\lambda(v)) \vee \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v}) \\ &= \neg\big(\beta_{\mathtt{sat}}(\lambda(v)) \wedge \neg\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v})\big) \\ &= \neg\mathtt{sat}_B(T, \beta_{\mathtt{satp}}, v).\end{aligned}$$

For the cases when v is a refined node, we let $\mathtt{OP} = \mathrm{ref}(v)$ and $\mathrm{children}_T(v) = \{v_1, \ldots, v_k\}$.

Case 2. The node v is refined and $\mathrm{actor}(v) = \mathrm{p}_T$.

If $\bar{v}$ does not exist, then

$$\begin{aligned}\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) &= \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k) \\ &= \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_k) \\ &= \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v),\end{aligned}$$

where the second equality follows from the fact that $\mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} = \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}}$ and from the induction

hypothesis. Similarly, if $\bar{v}$ does exist, then

$$\begin{aligned}
&\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) = \\
&= \mathtt{C}^{\mathrm{p}}_{\mathtt{satp}}\big(\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k), \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v})\big) \\
&= \big(\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k)\big) \wedge \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v}) \\
&= \big(\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k)\big) \wedge \neg\big(\neg \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v})\big) \\
&= \big(\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_k)\big) \wedge \neg \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v}) \\
&= \mathtt{C}^{\mathrm{p}}_{\mathtt{sat}}\big(\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_1)\, \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{p}}_{\mathtt{sat}} \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_k), \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v})\big) \\
&= \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v),
\end{aligned}$$

as required.

Case 3. The node v is refined and $\mathrm{actor}(v) = \mathrm{o}_T$.

To prove the lemma's conclusion in this case, we employ de Morgan's laws, which imply that for $x, y \in \{1, 0\}$ the equalities $x\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} y = \neg(\neg x\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \neg y)$ and $\mathtt{C}^{\mathrm{o}}_{\mathtt{satp}}(x, y) = \neg \mathtt{C}^{\mathrm{o}}_{\mathtt{sat}}(\neg x, y)$ hold. Similarly as in the previous case, we begin with the subcase when $\bar{v}$ does not exist. Then,

$$\begin{aligned}
\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) &= \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k) \\
&= \neg\big(\neg \mathtt{sat}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \neg \mathtt{sat}_B(T, \beta_{\mathtt{satp}}, v_k)\big) \\
&= \neg\big(\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_k)\big) \\
&= \neg \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v),
\end{aligned}$$

where the second equality follows from the induction hypothesis.

If $\bar{v}$ does exist, then

$$\begin{aligned}
&\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v) = \\
&= \mathtt{C}^{\mathrm{o}}_{\mathtt{satp}}\big(\mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k), \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, \bar{v})\big) \\
&= \mathtt{C}^{\mathrm{o}}_{\mathtt{satp}}\Big(\neg\big(\neg \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \neg \mathtt{satp}_B(T, \beta_{\mathtt{satp}}, v_k)\big), \mathtt{sat}_B(T, \beta_{\mathtt{satp}}, \bar{v})\big)\Big) \\
&= \neg \mathtt{C}^{\mathrm{o}}_{\mathtt{sat}}\big(\mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_1)\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \ldots \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v_k), \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, \bar{v})\big) \\
&= \neg \mathtt{sat}_B(T, \beta_{\mathtt{sat}}, v),
\end{aligned}$$

where for the consecutive equalities we used the equality $x\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{satp}} y = \neg(\neg x\, \mathtt{OP}^{\mathrm{o}}_{\mathtt{sat}} \neg y)$, the induction hypothesis, and the equality $\mathtt{C}^{\mathrm{o}}_{\mathtt{satp}}(x, y) = \neg \mathtt{C}^{\mathrm{o}}_{\mathtt{sat}}(\neg x, y)$, respectively. □

With the next proposition we establish that each element of the set semantics of a tree indeed describes a way of achieving the root goal of the tree.

Proposition 1. *Let T be an attack–defense tree. If a pair (P, O) belongs to the set semantics $\mathcal{S}(T)$ of T, then for every set $O' \subseteq \mathbb{B}^{\mathrm{o}_T} \backslash O$ the equality* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O') = 1$ *holds.*

Proof. Recall that the set semantics $\mathcal{S}(T)$ is a result of the bottom-up evaluation of the attribute $\mathcal{S}$ whose domain is $A_{\mathcal{S}} = (2^{2^{\mathbb{B}}\times 2^{\mathbb{B}}}, \cup, \odot, \odot, \cup, \odot, \cup)$, where $\odot$ is the operation defined on page 42, under the basic assignment

$$\beta(\mathtt{b}) = \begin{cases} \big\{\big(\{\mathtt{b}\}, \emptyset\big)\big\} \text{ if } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}, \\ \big\{\big(\emptyset, \{\mathtt{b}\}\big)\big\} \text{ otherwise.} \end{cases}$$

Let $O' \subseteq \mathbb{B}^{\mathrm{o}_T}$ be a set satisfying $O' \subseteq \mathbb{B}^{\mathrm{o}_T} \backslash O$, and let

$$\beta_{\mathtt{satp}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} \in P \text{ or } \mathtt{b} \in \mathbb{B}^{\mathrm{o}_T} \backslash O', \\ 0 \text{ if } \mathtt{b} \in O' \text{ or } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T} \backslash P. \end{cases}$$

Lemma 3 implies that $\mathrm{achieved}_{\mathrm{T}}(T, P \cup O') = \mathtt{satp}_B(T, \beta_{\mathtt{satp}})$.

Since both $(2^{2^{\mathbb{B}}\times 2^{\mathbb{B}}}, \cup, \odot)$ and $(\{0,1\}, \vee, \wedge)$ are commutative idempotent semirings, the bottom-up evaluations of $\mathcal{S}$ and $\mathrm{achieved}_{\mathrm{T}}(T, P \cup O')$ can be represented using their normal forms

$$\begin{aligned} \mathcal{S}(T) = {} & (\beta(\mathtt{b}_1^1) \odot \beta(\mathtt{b}_2^1) \odot \ldots \odot \beta(\mathtt{b}_{k_1}^1)) \cup \\ & \ldots \\ & \cup (\beta(\mathtt{b}_1^i) \odot \beta(\mathtt{b}_2^i) \odot \ldots \odot \beta(\mathtt{b}_{k_i}^i)) \cup \\ & \ldots \\ & \cup (\beta(\mathtt{b}_1^n) \odot \beta(\mathtt{b}_2^n) \odot \ldots \odot \beta(\mathtt{b}_{k_n}^n)), \end{aligned} \tag{7}$$

and

$$\begin{aligned} \mathrm{achieved}_{\mathrm{T}}(T, P \cup O') = {} & (\beta_{\mathtt{satp}}(\mathtt{b}_1^1) \wedge \beta_{\mathtt{satp}}(\mathtt{b}_2^1) \wedge \ldots \wedge \beta_{\mathtt{satp}}(\mathtt{b}_{k_1}^1)) \vee \\ & \ldots \\ & \vee (\beta_{\mathtt{satp}}(\mathtt{b}_1^i) \wedge \beta_{\mathtt{satp}}(\mathtt{b}_2^i) \wedge \ldots \wedge \beta_{\mathtt{satp}}(\mathtt{b}_{k_i}^i)) \vee \\ & \ldots \\ & \vee (\beta_{\mathtt{satp}}(\mathtt{b}_1^n) \wedge \beta_{\mathtt{satp}}(\mathtt{b}_2^n) \wedge \ldots \wedge \beta_{\mathtt{satp}}(\mathtt{b}_{k_n}^n)). \end{aligned} \tag{8}$$

From the definition of the basic assignment β and the operation $\odot$ it follows that for every $i \in \{1, \ldots, n\}$ the ith term

$$\beta(\mathtt{b}_1^i) \odot \beta(\mathtt{b}_2^i) \odot \ldots \odot \beta(\mathtt{b}_{k_i}^i)$$

of representation (7) is a set consisting of exactly one pair of sets. Let us denote this term with $\{(P_i, O_i)\}$. Since $(P, O) \in \mathcal{S}(T)$, there is $i \in \{1, \ldots, n\}$ such that $(P_i, O_i) = (P, O)$. Thus, under the basic assignment $\beta_{\mathtt{satp}}$, the corresponding term

$$\beta_{\mathtt{satp}}(\mathtt{b}_1^i) \wedge \beta_{\mathtt{satp}}(\mathtt{b}_2^i) \wedge \ldots \wedge \beta_{\mathtt{satp}}(\mathtt{b}_{k_i}^i)$$

of representation (8) evaluates to 1, implying that $\mathrm{achieved}_{\mathrm{T}}(T, P \cup O') = 1$, as required. □

Next, we shall demonstrate that among the strategies in an attack–defense tree there are all of the minimal strategies in this tree.

Proposition 2. *Let T be an attack–defense tree. If (P, O) is a minimal strategy in T, then $(P, O) \in \mathcal{S}(T)$.*

Proof. Let $O' = \mathbb{B}^{o_T} \backslash O$. Since (P, O) is a minimal strategy, the equality $\text{achieved}_{\text{T}}(T, P \cup O') = 1$ holds. Let $\beta_{\mathtt{satp}}$ be the basic assignment of the *satisfiability for the proponent* attribute defined by

$$\beta_{\mathtt{satp}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} \in P \cup O, \\ 0 \text{ otherwise.} \end{cases}$$

Then, by Lemma 3, $\text{achieved}_{\text{T}}(T, P \cup O') = \mathtt{satp}_B(T, \beta_{\mathtt{satp}})$, and so there is $i \in \{1, \ldots, n\}$ such that the ith term of the representation (8) of $\text{achieved}_{\text{T}}(T, P \cup O')$ evaluates to 1. The definition of $\beta_{\mathtt{satp}}$ together with the choice of O' imply that

$$P \cup O \supseteq \{\mathtt{b}_1^i, \mathtt{b}_2^i, \ldots, \mathtt{b}_{k_i}^i\}.$$

We will prove that the two sets are actually equal. Towards a contradiction, suppose that this is not the case. Then for

$$\overline{P} = P \cap \{\mathtt{b}_1^i, \mathtt{b}_2^i, \ldots, \mathtt{b}_{k_i}^i\},$$
$$\overline{O} = O \cap \{\mathtt{b}_1^i, \mathtt{b}_2^i, \ldots, \mathtt{b}_{k_i}^i\},$$

it holds that $\overline{P} \neq P$ or $\overline{O} \neq O$.

Suppose first that $\overline{P} \neq P$. Note that the value of the ith term of the representation (8) of $\text{achieved}_{\text{T}}(T, P \cup O')$ is the same as that of $\text{achieved}_{\text{T}}(T, \overline{P} \cup O')$. Thus, $\text{achieved}_{\text{T}}(T, \overline{P} \cup O') = 1$, contradicting the assumption of (P, O) being the minimal strategy.

Suppose now that $\overline{O} \neq O$, i.e., that $\overline{O}$ is a strict subset of O. Then for any $\widetilde{O} \subseteq \mathbb{B}^{o_T} \backslash \overline{O}$ setting

$$\widetilde{\beta}_{\mathtt{satp}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} \in P \text{ or } \mathtt{b} \in \mathbb{B}^{o_T} \backslash \widetilde{O}, \\ 0 \text{ if otherwise,} \end{cases}$$

yields $\widetilde{\beta}_{\mathtt{satp}}\Big|_{\overline{O}} \equiv 1$, implying that the ith term of representation (8) of $\mathtt{satp}_B(T, \widetilde{\beta}_{\mathtt{satp}})$ evaluates to 1. Since $\text{achieved}_{\text{T}}(\text{root}(T), P \cup \widetilde{O}) = \mathtt{satp}_B(T, \widetilde{\beta}_{\mathtt{satp}})$, by Lemma 3, it follows that $\text{achieved}_{\text{T}}(\text{root}(T), P \cup \widetilde{O}) = 1$. This means that the pair (P, O) does not satisfy the third condition of Definition 26, in contradiction with the choice of (P, O) as a minimal strategy in T.

The above reasoning proves that $P \cup O = \{\mathtt{b}_1^i, \mathtt{b}_2^i, \ldots, \mathtt{b}_{k_i}^i\}$. It follows that the ith term of the representation (7) of the set semantics of T is $\beta(\mathtt{b}_1^i) \odot \beta(\mathtt{b}_2^i) \odot \ldots \odot \beta(\mathtt{b}_{k_i}^i) = (P, O)$, completing the proof. □

Proposition 2 shows that minimal strategies are strategies. Combining it with Corollary 1 leads to the following corollary.

Corollary 2. *The minimal strategies in an attack–defense tree T are the minimal elements in $\mathcal{S}(T)$ w.r.t. the partial order $\preceq$ defined in Corollary 1.*

We finish our characterization of the set semantics by demonstrating that for many trees the converse of Proposition 2 holds, i.e., that in a class of trees with no clones every strategy is a minimal strategy.

Proposition 3. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree. If there are no clones in T, then every element of the set semantics $\mathcal{S}(T)$ is a minimal strategy in T.*

Proof. Recall that the set semantics of T is the result $\mathcal{S}_B(T, \beta)$ of the bottom-up evaluation of the attribute $\mathcal{S}$ whose attribute domain is $A_{\mathcal{S}} = (2^{2^{\mathbb{B}} \times 2^{\mathbb{B}}}, \cup, \odot, \odot, \cup, \odot, \cup)$, where $\odot$ is the operation defined on page 42, under the basic assignment β defined as

$$\beta(\mathtt{b}) = \begin{cases} \big\{\big(\{\mathtt{b}\}, \emptyset\big)\big\} \text{ if } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}, \\ \big\{\big(\emptyset, \{\mathtt{b}\}\big)\big\} \text{ otherwise.} \end{cases}$$

Informally speaking, we shall prove that for every node $v \in V$, every pair belonging to the intermediate result $\mathcal{S}_B(T, \beta, v)$ of the process of the set semantics creation satisfies locally the definition of minimal strategy. That is, relying on Corollary 2, for every $v \in V$ we will prove that every element of $\mathcal{S}_B(T, \beta, v)$ is a minimal element in $\mathcal{S}_B(T, \beta, v)$ w.r.t. the relation $\preceq$ defined by

$$(P', O') \preceq (P, O) \text{ if and only if } P' \subseteq P \text{ and } O' \subseteq O.$$

Since $\mathcal{S}(T) = \mathcal{S}_B(T, \beta, \mathrm{root}(T))$, the claimed statement will follow.

The proof is by induction on the structure of the subdag $T(v)$. For the base case, assume that v is a non-refined node and that $\bar{v}$ does not exist. The required minimality condition is then trivially satisfied, since $\mathcal{S}_B(T, \beta, v)$ is a singleton. We now proceed with the remaining cases.

Case 1. The node v is not refined and $\bar{v}$ exists.

To ease the presentation, for the proof of this case we let $\mathtt{b} := \lambda(v)$.

Case 1.1 $\mathrm{actor}(v) = \mathrm{p}_T$

In this case, definition of the attribute domain $A_{\mathcal{S}}$ and the basic assignment β imply that every element of $\mathcal{S}_B(T, \beta, v)$ is of the form $(\{\mathtt{b}\}, \emptyset) \odot (P', O') = (P' \cup \{\mathtt{b}\}, O')$, for some $(P', O') \in \mathcal{S}_B(T, \beta, \bar{v})$. Note that for $(P', O'), (P'', O'') \in \mathcal{S}_B(T, \beta, \bar{v})$ the relation $(P' \cup \{\mathtt{b}\}, O') \preceq (P'' \cup \{\mathtt{b}\}, O'')$ holds if and only if $(P', O') \preceq (P'', O'')$. Thus, since every element of $\mathcal{S}_B(T, \beta, \bar{v})$ is minimal in $\mathcal{S}_B(T, \beta, \bar{v})$ w.r.t. $\preceq$, by the induction hypothesis, it

follows that every element of $\mathcal{S}_B(T,\beta,v)$ is also minimal in $\mathcal{S}_B(T,\beta,v)$ w.r.t. $\preceq$.

Case 1.2 $\mathrm{actor}(v) = \mathrm{o}_T$

In this case, $\mathcal{S}_B(T,\beta,v) = \mathcal{S}_B(T,\beta,\bar{v}) \cup \{(\emptyset,\{\mathtt{b}\})\}$. Thus, the order between the elements of $\mathcal{S}_B(T,\beta,\bar{v})$ in $\mathcal{S}_B(T,\beta,v)$ is the same as in $\mathcal{S}_B(T,\beta,\bar{v})$. Since there are no clones in T, for every $(P',O') \in \mathcal{S}_B(T,\beta,\bar{v})$ it holds that $\mathtt{b} \notin O'$, and so $(\emptyset,\{\mathtt{b}\}) \preceq (P',O')$ does not hold. Therefore, by the induction hypothesis, every element of $\mathcal{S}_B(T,\beta,\bar{v})$ is a minimal element in $\mathcal{S}_B(T,\beta,v)$ w.r.t. $\preceq$. Clearly, the pair $(\emptyset,\{\mathtt{b}\})$ is also a minimal element in $\mathcal{S}_B(T,\beta,v)$ w.r.t. $\preceq$. Thus, the statement holds.

For a proof of the remaining cases, when v is a refined node, we let $\mathrm{children}_T(v) = \{v_1,\ldots,v_k\}$ and assume that the node $\bar{v}$ exists. The proof for the cases when $\bar{v}$ does not exist is obtained by skipping the parts related to $\bar{v}$ in what follows. Finally, we denote with (P,O) an arbitrary but fixed element of $\mathcal{S}_B(T,\beta,v)$.

Case 2. The node v is refined and $\mathrm{actor}(v) = \mathrm{p}_T$.

Case 2.1 $\mathrm{ref}(v) = \mathtt{OR}$

For a proof by contradiction, suppose that (P,O) is not a minimal element in $\mathcal{S}_B(T,\beta,v)$ w.r.t. the order $\preceq$. Then, there is an element $(P',O') \in \mathcal{S}_B(T,\beta,v)$ such that $(P',O') \prec (P,O)$. From definition of the attribute domain $A_{\mathcal{S}}$ it follows that there are $i,j \in \{1,\ldots,k\}$, nodes v_i, v_j and pairs $(P_i,O_i) \in \mathcal{S}_B(T,\beta,v_i)$, $(P_j,O_j) \in \mathcal{S}_B(T,\beta,v_j)$, $(\overline{P},\overline{O}), (\overline{P}',\overline{O}') \in \mathcal{S}_B(T,\beta,\bar{v})$, such that

$$(P,O) = (P_i \cup \overline{P}, O_i \cup \overline{O})$$
$$(P',O') = (P_j \cup \overline{P}', O_j \cup \overline{O}').$$

Note that the definition of $A_{\mathcal{S}}$ and the basic assignment β imply that none of the sets P_i and P_j is empty.

Since there are no clones in T, if $P_i \neq P_j$, then the sets in each of the triples $(P_i,P_j,\overline{P})$, $(P_i,P_j,\overline{P}')$ are pairwise disjoint. Thus, the condition $P' \subseteq P$ is not satisfied, contradicting the choice of (P',O') and implying that (P,O) is indeed a minimal element in $\mathcal{S}_B(T,\beta,v)$ w.r.t. $\preceq$.

Suppose now that $P_i = P_j$. Since there are no clones in T, this implies that $v_i = v_j$. Furthermore, from the choice of (P',O') as an element satisfying $(P',O') \prec (P,O)$ it follows that $\overline{P}' \subseteq \overline{P}$. Since every element of $\mathcal{S}_B(T,\beta,\bar{v})$ is minimal in $\mathcal{S}_B(T,\beta,\bar{v})$ w.r.t. $\preceq$, by the induction hypothesis, this implies that either $(\overline{P},\overline{O}) = (\overline{P}',\overline{O}')$ or else $\overline{O}'$ is not a subset of $\overline{O}$. If $(\overline{P},\overline{O}) = (\overline{P}',\overline{O}')$, then, since $(P',O') \prec (P,O)$, it follows that $O_i \neq O_j$ and $O_i \subseteq O_j$. But then $(P_i,O_i) \preceq (P_i,O_j)$, contradicting the induction hypothesis for v_i.

Thus, $P_i = P_j$, $\overline{P}' \subseteq \overline{P}$ and $\overline{O}'$ is not a subset of $\overline{O}$. The last of these facts implies in particular the the set $\overline{O}'$ is not empty. But, since $(P',O') \prec (P,O)$, we have that

$O_j \cup \overline{O}' \subseteq O_i \cup \overline{O}$, and so the intersection $\overline{O}' \cap O_i$ cannot be empty. And yet, empty it surely is, since there are no clones in T. This final contradiction completes the proof of this case.

Case 2.2 $\mathrm{ref}(v) = \texttt{AND}$

In this case, the pair (P, O) can be represented as

$$(P, O) = (P_1 \cup \ldots \cup P_k \cup \overline{P}, O_1 \cup \ldots \cup O_k \cup \overline{O}),$$

for some $(P_i, O_i) \in \mathcal{S}_B(T, \beta, v_i)$, for $i \in \{1, \ldots, k\}$, and some $(\overline{P}, \overline{O}) \in \mathcal{S}_B(T, \beta, \bar{v})$. For a proof by contradiction, suppose again that (P, O) is not a minimal element in $\mathcal{S}_B(T, \beta, v)$ w.r.t. the order $\preceq$, i.e., that there is an element $(P', O') \in \mathcal{S}_B(T, \beta, v)$ such that $(P', O') \prec (P, O)$. Let

$$(P', O') = (P'_1 \cup \ldots \cup P'_k \cup \overline{P}', O'_1 \cup \ldots \cup O'_k \cup \overline{O}'),$$

for some $(P'_i, O'_i) \in \mathcal{S}_B(T, \beta, v_i)$, for $i \in \{1, \ldots, k\}$, and some $(\overline{P}', \overline{O}') \in \mathcal{S}_B(T, \beta, \bar{v})$. Since there are no clones in T, for $i, j \in \{1, \ldots, k\}$, $i \neq j$, each of the intersections

$$\begin{array}{l} P_i \cap P_j, P_i \cap P'_j, P_i \cap \overline{P}, P_i \cap \overline{P}', P'_i \cap \overline{P}, P'_i \cap \overline{P}', \\ O_i \cap O_j, O_i \cap O'_j, O_i \cap \overline{O}, O_i \cap \overline{O}', O'_i \cap \overline{O}, O'_i \cap \overline{O}' \end{array}$$

is empty. Note that, since $(P', O') \neq (P, O)$, either there is $i \in \{1, \ldots, k\}$ such that $(P_i, O_i) \neq (P'_i, O'_i)$ or else $(\overline{P}, \overline{O}) \neq (\overline{P}', \overline{O}')$. But then it follows from Lemma 1 that either $(P_i, O_i) \prec (P'_i, O'_i)$ or else $(\overline{P}, \overline{O}) \prec (\overline{P}', \overline{O}')$. This contradicts the induction hypothesis for v_i or $\bar{v}$.

Case 3. The node v is refined and $\mathrm{actor}(v) = \mathrm{o}_T$.

Case 3.1 $\mathrm{ref}(v) = \texttt{OR}$

From the definition of the set semantics it follows that $(P, O) = (P_1 \cup \ldots \cup P_k, O_1 \cup \ldots \cup O_k)$ for some $(P_i, O_i) \in \mathcal{S}_B(T, \beta, v_i)$, for $i \in \{1, \ldots, k\}$, or else $(P, O) \in \mathcal{S}_B(T, \beta, \bar{v})$. If the former is true, then, since the operation performed during the bottom-up creation of the set semantics at the `AND` nodes of the proponent and the `OR` nodes of the opponent is the same, a proof of the declared statement is obtained by repeating the reasoning from Case 2.2, combined with the fact that for every $i \in \{1, \ldots, k\}$, every $(P', O') \in \mathcal{S}_B(T, \beta, v_i)$ and every $(\overline{P}, \overline{P}') \in \mathcal{S}_B(T, \beta, \bar{v})$, neither $(P', O') \prec (\overline{P}, \overline{O})$ nor $(\overline{P}, \overline{O}) \prec (P', O')$. Thus, we assume that $(P, O) \in \mathcal{S}_B(T, \beta, \bar{v})$. But now the statement follows immediately from the induction hypothesis for $\bar{v}$ and the last part of the previous sentence, i.e., neither $(P', O') \prec (P, O)$ nor $(P, O) \prec (P', O')$ being satisfied for any of $i \in \{1, \ldots, k\}$, and any of $(P', O') \in \mathcal{S}_B(T, \beta, v_i)$.

Case 3.2 $\mathrm{ref}(v) = \texttt{AND}$

In this case, definition of the set semantics implies that

$$\mathcal{S}_B(T,\beta,v) = \bigcup_{v'\in\{v_1,\ldots,v_k,\bar{v}\}} \mathcal{S}_B(T,\beta,v').$$

Since there are no clones in T, for every $v', v'' \in \{v_1,\ldots,v_k,\bar{v}\}$, $v' \neq v''$ and every $(P',O') \in \mathcal{S}_B(T,\beta,v')$, $(P'',O'') \in \mathcal{S}_B(T,\beta,v')$, the sets P', P'' are disjoint, and the sets O', O'' are disjoint. Thus, neither $(P',O') \prec (P'',O'')$ nor $(P'',O'') \prec (P',O')$. Combined with the induction hypothesis, this implies that every element in $\mathcal{S}_B(T,\beta,v)$ is minimal w.r.t. the order $\preceq$. □

Summarizing the results presented in Proposition 1– 3, (1) the elements of the set semantics indeed describe ways of achieving the root goal of a tree, (2) among them there are all of the minimal strategies in the tree, and (3) there are trees, in particular the class of trees with no clones, in which every element of the set semantic is a minimal strategy. We believe that this characterization can be useful for proper interpretation of the results of evaluation of attributes on the set semantics. The following example supports this belief.

Example 34. *Consider the* maximal probability for the proponent *attribute* `prob`*, whose attribute domain is* $([0,1],\max,\cdot,\cdot,\max,\cdot,\max)$ *(cf. Table 1). Let* (P,O) *and* (P',O') *be strategies in an attack–defense tree* T *such that* $(P,O) \neq (P',O')$ *and* $P \subseteq P'$, $O \subseteq O'$. *Then, for any basic assignment* β *the equality*

$$\max\left(\prod_{\mathtt{b}\in P\cup O}\beta(\mathtt{b}), \prod_{\mathtt{b}\in P'\cup O'}\beta(\mathtt{b}),\right) = \prod_{\mathtt{b}\in P\cup O}\beta(\mathtt{b})$$

holds. It follows that the result of evaluation of `prob` *on the set semantics in* T *under the basic assignment* β *is*

$$\mathtt{prob}_{\mathcal{S}}(T,\beta) = \max_{(P,O)\in\mathcal{S}(T)} \prod_{\mathtt{b}\in P\cup O}\beta(\mathtt{b}) = \max_{\substack{(P,O)\in\mathcal{S}(T)\\ (P,O)\text{ is a minimal strategy in }T}} \prod_{\mathtt{b}\in P\cup O}\beta(\mathtt{b}).$$

Therefore, if β *assigns to basic actions the* probability of successful execution*, then the value of* $\mathtt{prob}_{\mathcal{S}}(T,\beta)$ *represents the maximal probability of achieving the root goal of* T *when executing exactly one of the minimal strategies in* T.

4.3 Computational aspects of the evaluation of attributes on the set semantics

Having provided means for intuitive interpretation of its results, we now turn our attention to the computational aspects of the evaluation of attributes on the set semantics. We begin with noting that the first step of this evaluation method is the creation of the set semantics, which is highly complex, due to the $\odot$ operation defined by formula (4).

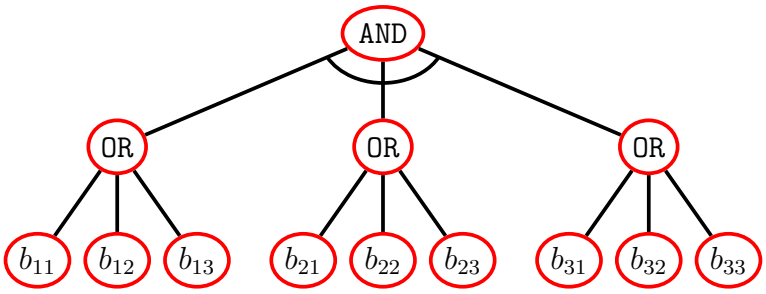


Figure 11: An example of a tree with the size of the set semantics exponential in the number of basic actions (see Example 35)

Indeed, because of this operation, the size of the set semantics might be exponential in both the number of basic actions in the tree and the total number of nodes, as illustrated by the following construction (see also Figure 11).

Example 35. *Let* $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{11}, \mathtt{b}_{12}, \mathtt{b}_{13}), \ldots, \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{k1}, \mathtt{b}_{k2}, \mathtt{b}_{k3}))$ *be an attack tree with* $n = 3k$ *basic actions and* $4k + 1$ *nodes, containing no clones. Proposition 3 implies that the set semantics of* T *consists of the minimal sets achieving the root goal of* T*. The number of such sets is* $3^k = 3^{n/3}$*.*

Therefore, while the set semantics is useful for formalizing intuition behind an attribute with an appropriate attribute domain, there are trees for which its practical application for evaluation of attributes is limited. What is important, however, is that this is not the case for all trees. That is, even if a tree is big, in the sense of the number of nodes or basic actions, the size of its set semantics might be small enough so that the evaluation of attributes on the semantics will perform well.

Should one want to evaluate an attribute on a tree with clones, it seems thus reasonable to try to estimate the size of the set semantics of the tree before trying to create it. Should the obtained estimate be reasonable, one could proceed with this evaluation method. The following proposition provides a fast procedure for computing an upper bound on the size of the set semantics of a given tree.

Proposition 4. *Let* $\mathtt{SetSemBound}$ *be an attribute with the attribute domain* $A_{\mathtt{SetSemBound}} = (\mathbb{N}, +, \cdot, \cdot, +, \cdot, +)$*, where* $\cdot$ *is the multiplication operator. Let* $\beta_{\mathtt{SetSemBound}} \equiv 1$ *be a basic assignment of* $\mathtt{SetSemBound}$*. Then the inequality*

$$|\mathcal{S}(T)| \leqslant \mathtt{SetSemBound}_B(T, \beta_{\mathtt{SetSemBound}}) \tag{9}$$

holds for every attack–defense tree T*.*

Proof. Since $(\mathbb{N}, +, \cdot)$ is a commutative semiring, the result of the bottom-up computation

of the SetSemBound attribute can be represented in the normal form

$$\begin{aligned}\mathtt{SetSemBound}_B(T, \beta_{\mathtt{SetSemBound}}) = {} & (1 \cdot 1 \cdot \ldots \cdot 1) + \\ & \ldots \\ & + (1 \cdot 1 \cdot \ldots \cdot 1) + \\ & \ldots \\ & + (1 \cdot 1 \cdot \ldots \cdot 1).\end{aligned}$$

The number of terms in the above expression is the same as the number of terms in the normal form (7) of the set semantics of T. Since the latter is at least $|\mathcal{S}(T)|$, the statement follows. □

Proposition 4 shows that an upper bound on the number of strategies can be found in time linear in the size of the tree. We note that the bound provided by the inequality (9) is tight: the equality is attained for instance for trees from Example 35[2]. Nevertheless, the difference between the bound and the actual size of the set semantics can be arbitrarily large. For example, there are three elements in the set semantics of the tree $\mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{a},\mathtt{b}),\ldots,\mathtt{OR}^{\mathrm{p}}(\mathtt{a},\mathtt{b}))$, while the bound is equal to 2 to the power equal to the number of OR nodes.

Having an easily computable formula for a non-trivial *lower bound* on the size of the set semantics would also be very useful. It seems that such a lower bound cannot be computed using a single bottom-up procedure that would simply propagate natural numbers throughout the tree. This is the case, because such a procedure would have to yield 1 for every attack tree in which all the leaf nodes bear the same label, irrespective of the tree structure. To obtain a non-trivial lower bound, one would have to propagate, along a number, some additional information about the repeated basic actions seen so far in the tree.

There are at least two other ways of avoiding the possible complexity of the evaluation of attributes on the set semantics. One of them would be to use the bottom-up evaluation, while being sure that it will return the same, correct result. The other one, to be employed if the bottom-up procedure fails, is to devise yet another, alternative method of attributes evaluation. With the following theorem we establish some sufficient conditions for employing the simple bottom-up evaluation instead of the evaluation on the set semantics with the guarantee of obtaining the correct result.

Theorem 1. *Let T be an attack–defense tree and let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be an attribute domain such that the operations $\oplus$ and $\otimes$ are associative and commutative, $\oplus$ is idempotent, and $\otimes$ distributes over $\oplus$. Furthermore, let β' be a basic assignment of α. If*

[2]In fact, we believe that the equality is attained for every tree that does not contain clones. We are, however, unable to prove this statement.

– *there are no repeated labels in T, or*

– *the operator $\otimes$ is idempotent, or*

– *for every clone $\mathtt{b}$ in T it holds that $\beta'(\mathtt{b}) \in \{\mathtt{a}_\otimes, \mathtt{e}_\otimes\}$,*

then the equality $\alpha_B(T, \beta') = \alpha_{\mathcal{S}}(T, \beta')$ holds.

Proof. Let β be the basic assignment defined for T as in Definition 23 of the set semantics. Consider the normal forms

$$\begin{aligned}\mathcal{S}(T) = {} & (\beta(\mathtt{b}_1^1) \odot \beta(\mathtt{b}_2^1) \odot \ldots \odot \beta(\mathtt{b}_{k_1}^1)) \cup \\ & \ldots \\ & \cup (\beta(\mathtt{b}_1^i) \odot \beta(\mathtt{b}_2^i) \odot \ldots \odot \beta(\mathtt{b}_{k_i}^i)) \cup \\ & \ldots \\ & \cup (\beta(\mathtt{b}_1^n) \odot \beta(\mathtt{b}_2^n) \odot \ldots \odot \beta(\mathtt{b}_{k_n}^n)),\end{aligned} \tag{10}$$

$$\begin{aligned}\alpha_B(T, \beta') = {} & (\beta'(\mathtt{b}_1^1) \otimes \beta'(\mathtt{b}_2^1) \otimes \ldots \otimes \beta'(\mathtt{b}_{k_1}^1)) \oplus \\ & \ldots \\ & \oplus (\beta'(\mathtt{b}_1^i) \otimes \beta'(\mathtt{b}_2^i) \otimes \ldots \otimes \beta'(\mathtt{b}_{k_i}^i)) \oplus \\ & \ldots \\ & \oplus (\beta'(\mathtt{b}_1^n) \otimes \beta'(\mathtt{b}_2^n) \otimes \ldots \otimes \beta'(\mathtt{b}_{k_n}^n)).\end{aligned} \tag{11}$$

Relying on the idempotency of both sets union and the operator $\oplus$, we assume that every term appearing in the two representations is unique, i.e., that for any two distinct i_1 and i_2 belonging to the set $\{1, \ldots, n\}$ the multisets $\{|\mathtt{b}_1^{i_1}, \mathtt{b}_2^{i_1}, \ldots, \mathtt{b}_{k_{i_1}}^{i_1}|\}$ and $\{|\mathtt{b}_1^{i_2}, \mathtt{b}_2^{i_2}, \ldots, \mathtt{b}_{k_{i_2}}^{i_2}|\}$ are different.

Let us denote again the ith term of representation (10) with $\{(P_i, O_i)\}$. Note that if the operation $\otimes$ is idempotent or every cloned basic action is assigned $\mathtt{a}_\otimes$ or $\mathtt{e}_\otimes$ under the basic assignment β', then for every clone $\mathtt{b}$ in T the equality

$$\beta'(\mathtt{b}) \otimes \beta'(\mathtt{b}) \otimes \ldots \otimes \beta'(\mathtt{b}) = \beta'(\mathtt{b})$$

holds. Furthermore, if there are no clones in T, then for every $i \in \{1, \ldots, n\}$ the multiset $\{|\mathtt{b}_1^i, \mathtt{b}_2^i, \ldots, \mathtt{b}_{k_i}^i|\}$ is in fact a set, i.e., every basic action appears in each of the terms of representation (10) and (11) at most once. It follows that under any of the three conditions we have

$$\beta'(\mathtt{b}_1^i) \otimes \beta'(\mathtt{b}_2^i) \otimes \ldots \otimes \beta'(\mathtt{b}_{k_i}^i) = \bigotimes_{\mathtt{b} \in P_i \cup O_i} \beta'(\mathtt{b}),$$

implying that

$$\alpha_B(T, \beta') = \bigoplus_{(P,O) \in \mathcal{S}(T)} \bigotimes_{\mathtt{b} \in P \cup O} \beta'(\mathtt{b}) = \alpha_{\mathcal{S}}(T, \beta'),$$

as required. □

Among the attribute domains gathered in Table 1, there are two of the form $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ satisfying the assumptions of Theorem 1, with the operation $\otimes$ being idempotent. Therefore, the bottom-up procedure can be used to evaluate these attributes on attack–defense trees containing repeated basic actions, yielding the same result as the evaluation on the set semantics. The attribute domains in which the operation $\otimes$ is not idempotent include the domains for the *minimal cost for the proponent* and the *maximal probability for the proponent* attributes. Nevertheless, these two domains enjoy a useful property that can be exploited for the purpose of the attribute evaluation on attack–defense trees with clones. This property is captured by the following notion.

Definition 27 (Non-increasing attribute domain)**.** *An attribute domain A_α is non-increasing if A_α is of the form $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, where $(D_\alpha, \oplus, \otimes)$ is a commutative idempotent semiring, such that for every $c, d \in D_\alpha$ the equality $c \oplus (c \otimes d) = c$ holds*[3]*.*

To give some intuition regarding the non-increasing attribute domains, assume that $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ is such a domain. Then, for an attack–defense tree T, two sets $P, P' \subseteq \mathbb{B}^{\mathrm{p}_T}$ and two sets $O, O' \subseteq \mathbb{B}^{\mathrm{o}_T}$ satisfying $P \subseteq P'$, $O \subseteq O'$, the equality

$$\Big(\bigotimes_{\mathtt{b} \in P \cup O} \beta(\mathtt{b}) \Big) \oplus \Big(\bigotimes_{\mathtt{b} \in P' \cup O'} \beta(\mathtt{b}) \Big) = \bigotimes_{\mathtt{b} \in P \cup O} \beta(\mathtt{b})$$

holds for any basic assignment β for α. Combining this fact with Example 26 and Corollary 2, one can see that for the attribute α the equality

$$\alpha_{\mathcal{S}}(T, \beta) = \bigoplus_{(P,O) \in \mathcal{S}(T)} \bigotimes_{\mathtt{b} \in P \cup O} \beta_\alpha(\mathtt{b}) = \bigoplus_{\substack{(P,O) \in \mathcal{S}(T) \\ (P,O) \text{ is a minimal strategy in } T}} \bigotimes_{\mathtt{b} \in P \cup O} \beta_\alpha(\mathtt{b}) \tag{12}$$

holds. In other words, if an attribute has a non-increasing domain, then the non-minimal strategies have no impact on the evaluation of this attribute on the set semantics.

We note that from the attribute domains displayed in Table 1, only the *maximal damage done by the proponent* and *satisfiability* domains are *not* non-increasing. This is because the equality $\max(c, c + d) = c$ does not hold for every $c, d \in \mathbb{R}_{\geqslant 0}$, and because the *satisfiability* domain is not induced by a semiring.

In the next section, we present an alternative method for attributes evaluation that can be employed for attributes having non-increasing domains, yielding the same result as the evaluation on the set semantics. We finish this section with a remark regarding the compatibility of the bottom-up evaluation of attributes with the set semantics, which follows immediately from Theorem 1.

Proposition 5. *Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be an attribute domain such that the operations $\oplus$ and $\otimes$ are associative and commutative, $\oplus$ is idempotent, and $\otimes$ distributes*

[3]This condition is equivalent to the inequality $d \otimes c \preceq c$, where $\preceq$ stands for the canonical partial order on the semiring $(D_\alpha, \oplus, \otimes)$. This is the reason for the name *non-increasing*.

over $\oplus$. *Let* $D_\alpha^{\mathbb{B}}$ *and be the set of all basic assignments for* α *and let* $\mathbb{T}'$ *be the set of all trees containing no clones. Finally, let*

$$f' \colon \mathbb{T}' \times D_\alpha^{\mathbb{B}} \ni (T, \beta) \mapsto \alpha_B(T, \beta)$$

and

$$f \colon \mathbb{T} \times D_\alpha^{\mathbb{B}} \ni (T, \beta) \mapsto \alpha_B(T, \beta).$$

The function f' *is compatible with the set semantics, and the function* f *is compatible with the set semantics if and only if the operation* $\otimes$ *is idempotent.*

Proof. By Theorem1, the equality $f'(T, \beta) = \alpha_{\mathcal{S}}(T, \beta)$ holds for every $(T, \beta) \in \mathbb{T}' \times D_\alpha^{\mathbb{B}}$, and if $\otimes$ is an idempotent operation, then $f(T, \beta) = \alpha_{\mathcal{S}}(T, \beta)$ for every $(T, \beta) \in \mathbb{T} \times D_\alpha^{\mathbb{B}}$. The assumptions on A_α imply that

$$\alpha_{\mathcal{S}}(T, \beta) = \bigoplus_{(P,O) \in \mathcal{S}(T)} \bigotimes_{\mathtt{b} \in P \cup O} \beta_\alpha(\mathtt{b}).$$

It follows that for any two trees T_1 and T_2 having the same set semantics the equality $\alpha_{\mathcal{S}}(T_1, \beta) = \alpha_{\mathcal{S}}(T_2, \beta)$ holds for every $\beta \in D_\alpha^{\mathbb{B}}$. Thus, the function f' is compatible with the set semantics, and if $\otimes$ is idempotent, then f is also compatible with the set semantics.

To prove that f is not compatible with the set semantics when the operation $\otimes$ is not idempotent, it is enough to consider trees $T_1 = \mathtt{b}$ and $T_2 = \mathtt{AND}^{\mathrm{p}}(\mathtt{b}, \mathtt{b})$. Note that $\mathcal{S}(T_1) = \mathcal{S}(T_2)$. If $\otimes$ is not idempotent, then there exists $x \in D_\alpha$ such that $x \otimes x \neq x$. Thus, for every basic assignment β assigning x to the basic action $\mathtt{b}$ the value of $f(T_1, \beta)$ is different from $f(T_2, \beta)$. □

4.4 A method for evaluation of attributes in trees with clones

In the previous section, we have identified the class of attributes having *non-increasing attribute domains*, which includes such important attributes like *minimal cost for the proponent* and *maximal probability for the proponent* (cf. Example 34). As the bottom-up evaluation of these attributes might result in incorrect results, and their evaluation on the set semantics might be computationally infeasible, we are going to develop a new method of evaluation of attributes. It is tailored specifically for the attributes having non-increasing attribute domains, it can be applied on trees having clones, and in terms of computational complexity, it offers a compromise between the bottom-up evaluation and the evaluation on the set semantics.

The idea behind our method is simple. The values assigned to the repeated basic actions are temporarily modified, and for each such modification the bottom-up evaluation is performed. The values modification mimics the proponent performing some of the clones, and not performing others. The results obtained in this way are eventually

combined in an appropriate manner, yielding the same result as the computation on the set semantics.

4.4.1 Necessary and optional clones

Since we would like to be able to perform a "what-if" analysis similar to the one enabled by the remaining two methods, we begin with determining, for a given set O of actions of the opponent, the clones that the proponent *needs to* execute in order to achieve the root goal when the opponent performs O. This knowledge will determine the way in which the values assigned to the clones will be tackled later. The clones that need to be executed under fixed behavior of the opponent are called *necessary clones*.

Definition 28 (Necessary and optional clones)**.** *Let* $\mathtt{b}$ *be a cloned basic action of the proponent in an attack–defense tree* T *and let* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$*. The action* $\mathtt{b}$ *is a necessary clone w.r.t.* O *in* T *if*

- *there is a strategy* $(P, O') \in \mathcal{S}(T)$ *satisfying* $O \cap O' = \emptyset$*, and*
- *for every strategy* $(P, O') \in \mathcal{S}(T)$ *satisfying* $O \cap O' = \emptyset$ *it holds that* $\mathtt{b} \in P$*.*

If $\mathtt{b}$ *is not a necessary clone w.r.t.* O*, then it is called an optional clone w.r.t.* O*.*

In the case of attack trees, the set of basic actions of the opponent is empty, and so the only set that a clone can be necessary or optional w.r.t., is the empty set $\emptyset$. Hence, in the case of attack trees we reason simply about necessary and optional clones, without specifying the corresponding set. A necessary clone in an attack tree is a one that belongs to every strategy in this tree, as illustrated by the following example.

Example 36. *Consider the attack tree* T *depicted in Figure 12. The set semantics of* T *is*

$$\mathcal{S}(T) = \{(\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \emptyset), (\{\mathtt{a}, \mathtt{c}, \mathtt{d}\}, \emptyset), (\{\mathtt{b}, \mathtt{c}, \mathtt{d}\}, \emptyset), (\{\mathtt{b}, \mathtt{c}\}, \emptyset)\}.$$

Hence, $\mathtt{c}$ *is the only necessary clone in* T*, and the only optional clone is* $\mathtt{b}$*; in order to achieve the root goal, the attacker has to perform action* $\mathtt{c}$*, and there are ways of achieving the root goal that do not involve executing* $\mathtt{b}$*.*

In the case of attack–defense trees, clone can be optional w.r.t. some of the sets of basic actions of the opponent, and necessary w.r.t. to others, as illustrated by Example 37.

Example 37. *Let* T *be the attack–defense tree from Figure 13. The set semantics of* T *is*

$$\mathcal{S}(T) = \{(\{\mathtt{a}, \mathtt{c}\}, \{\mathtt{d}\}), (\{\mathtt{b}, \mathtt{c}\}, \{\mathtt{d}\}), (\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \emptyset), (\{\mathtt{b}, \mathtt{c}\}, \emptyset)\}.$$

The only clone in T *is* $\mathtt{b}$*. It is a necessary clone w.r.t.* $\{\mathtt{d}\}$*, and an optional clone w.r.t.* $\emptyset$*. This reflects the fact that if the opponent executes the action* $\mathtt{d}$*, then in order to achieve the root goal the proponent* has to *execute* $\mathtt{b}$*; if the opponent does nothing, the execution of* $\mathtt{b}$ *is not necessary.*

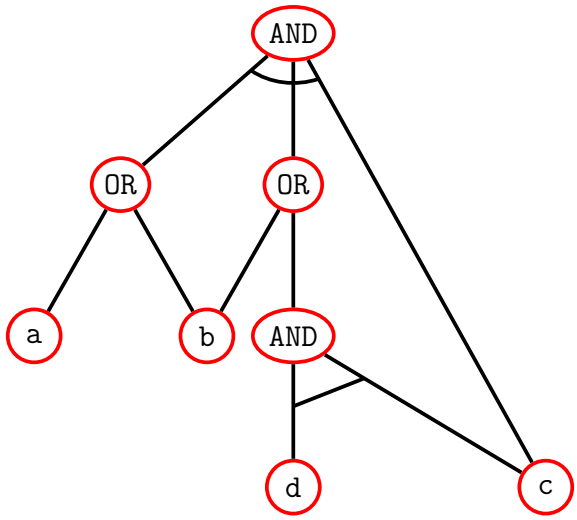


Figure 12: In the attack–defense tree $T = \mathtt{AND}^{\mathrm{p}}\big(\mathtt{OR}^{\mathrm{p}}(\mathtt{a},\mathtt{b}), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}, \mathtt{AND}^{\mathrm{p}}(\mathtt{d},\mathtt{c})), \mathtt{c}\big)$, the clone $\mathtt{c}$ is necessary, and the clone $\mathtt{b}$ is optional.

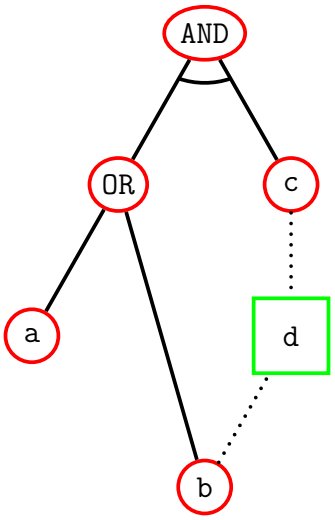


Figure 13: In the attack–defense tree $T = \mathtt{AND}^{\mathrm{p}}\big(\mathtt{OR}^{\mathrm{p}}(\mathtt{a},\mathtt{b}), \mathtt{C}^{\mathrm{p}}(\mathtt{c}, \mathtt{C}^{\mathrm{o}}(\mathtt{d},\mathtt{b}))\big)$, the clone $\mathtt{b}$ is necessary w.r.t. $\{\mathtt{d}\}$.

Example 38. *Consider the attack–defense tree T from Figure 2 on page 15, whose set semantics is given in Example 23 on page 42. The only clone in this tree is* `phish`*. This action is an optional clone w.r.t. every set $O \subseteq \mathbb{B}^{o_T}$. This is the case, because under any behavior of the opponent the proponent can achieve the root goal without executing the phishing action.*

The sets of all necessary and optional clones w.r.t. a set $O \subseteq \mathbb{B}^{o_T}$ in a tree T are denoted with $\mathcal{C}_N(T,O)$ and $\mathcal{C}_O(T,O)$, respectively. When there is no danger of ambiguity, we use $\mathcal{C}_N(O)$ and $\mathcal{C}_O(O)$ instead of $\mathcal{C}_N(T,O)$ and $\mathcal{C}_O(T,O)$. In the next lemma, a simple method for determining whether a cloned basic action of the proponent is a necessary clone w.r.t. a given set of the opponent's actions is provided.

Lemma 4. *Let T be an attack–defense tree, $\mathtt{a} \in \mathbb{B}^{\mathrm{P}_T}$ be a cloned basic action of the proponent in T and $\overline{O} \subseteq \mathbb{B}^{\mathrm{O}_T}$ be a set of basic actions of the opponent. Let $\beta_{\mathtt{skill}}$ be the basic assignment of the* minimal skill of the proponent *attribute defined as*

$$\beta_{\mathtt{skill}}(\mathtt{b}) = \begin{cases} 1 \text{ if } \mathtt{b} = \mathtt{a}, \\ +\infty \text{ if } \mathtt{b} \in \overline{O}, \\ 0 \text{ otherwise.} \end{cases}$$

The basic action $\mathtt{a}$ *is a necessary clone w.r.t. $\overline{O}$ if and only if $\mathtt{skill}_B(T, \beta_{\mathtt{skill}}) = 1$.*

Proof. Recall that the attribute domain for the *minimal skill of the proponent* attribute is $(\mathbb{N} \cup \{0, +\infty\} \min, \max, \max, \min, \max, \min)$. Since max is an idempotent operation, it follows from Theorem 1 that $\mathtt{skill}_B(T, \beta_{\mathtt{skill}}) = \mathtt{skill}_{\mathcal{S}}(T, \beta_{\mathtt{skill}})$. Observe that $\mathtt{skill}_{\mathcal{S}}(T, \beta_{\mathtt{skill}})$ can be represented as

$$\begin{aligned} \mathtt{skill}_{\mathcal{S}}(T, \beta_{\mathtt{skill}}) &= \min_{(P,O)\in\mathcal{S}(T)} \max_{\mathtt{b}\in P\cup O} \beta_{\mathtt{skill}}(\mathtt{b}) \\ &= \min \Big(\min_{\substack{(P,O)\in\mathcal{S}(T) \\ \mathtt{a}\in P \\ \overline{O}\cap O=\emptyset}} \max_{\mathtt{b}\in P\cup O} \beta_{\mathtt{skill}}(\mathtt{b}), \\ &\qquad \min_{\substack{(P,O)\in\mathcal{S}(T) \\ \mathtt{a}\notin P \\ \overline{O}\cap O=\emptyset}} \max_{\mathtt{b}\in P\cup O} \beta_{\mathtt{skill}}(\mathtt{b}), \\ &\qquad \min_{\substack{(P,O)\in\mathcal{S}(T) \\ \overline{O}\cap O\neq\emptyset}} \max_{\mathtt{b}\in P\cup O} \beta_{\mathtt{skill}}(\mathtt{b}) \Big). \end{aligned} \tag{13}$$

Assume first that $\mathtt{a}$ is a necessary clone w.r.t. $\overline{O}$. It follows from Definition 28 that the set semantics $\mathcal{S}(T)$ of T contains no pairs (P, O) satisfying $\overline{O} \cap O = \emptyset$ and $\mathtt{a} \notin P$. Thus, the expression (13) reduces to

$$\mathtt{skill}_{\mathcal{S}}(T, \beta_{\mathtt{skill}}) = \min\Big(\min_{\substack{(P,O)\in\mathcal{S}(T) \\ \mathtt{a}\in P \\ \overline{O}\cap O=\emptyset}} \max_{\mathtt{b}\in P\cup O} \mathtt{b}, \min_{\substack{(P,O)\in\mathcal{S}(T) \\ \overline{O}\cap O\neq\emptyset}} \max_{\mathtt{b}\in P\cup O} \mathtt{b} \Big).$$

Together with the basic assignment $\beta_{\mathtt{skill}}$ this implies that $\mathtt{skill}_{\mathcal{S}}(T, \beta_{\mathtt{skill}}) = 1$, whether the set semantics $\mathcal{S}(T)$ contains pairs (P, O) satisfying $\overline{O} \cap O \neq \emptyset$ or not.

Assume now that $\mathtt{skill}_B(T, \beta_{\mathtt{skill}}) = 1$. Then, it immediately follows from (13) and the definition of $\beta_{\mathtt{skill}}$ that there is a strategy $(P, O) \in \mathcal{S}(T)$ satisfying $\overline{O} \cap O = \emptyset$, and that $\mathtt{a} \in P$ for every strategy $(P, O) \in \mathcal{S}(T)$ satisfying $\overline{O} \cap O = \emptyset$, i.e., that $\mathtt{a}$ is a necessary clone w.r.t. $\overline{O}$. □

4.4.2 Repeated bottom-up evaluation of attributes

The idea behind our novel method of evaluation of attributes, given in Algorithm 1, is to first recognize the set $\mathcal{C}_N(O)$ of necessary clones and temporarily ensure that the values of the attribute assigned to them do not influence the result of the bottom–up

Algorithm 1 Repeated bottom-up evaluation of attributes

Input: Attack–defense tree T, attribute domain $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, basic assignment $\beta\colon \mathbb{B} \to D_\alpha$, set $O \subseteq \mathbb{B}^{o_T}$

Output: $\alpha_{RB}(T, \beta, O)$

1: $\alpha_{RB}(T, \beta, O) \leftarrow \mathtt{e}_\oplus$
2: initialize $\mathcal{C}_N(O)$, $\mathcal{C}_O(O)$
3: $\beta'(\mathtt{b}) \leftarrow \mathtt{e}_\otimes$ for every $\mathtt{b} \in \mathcal{C}_N(O)$
4: $\beta'(\mathtt{b}) \leftarrow \beta(\mathtt{b})$ for every $\mathtt{b} \in \mathbb{B}_T \backslash (\mathcal{C}_N(O) \cup \mathcal{C}_O(O))$
5: **for** every subset $\mathcal{C} \subseteq \mathcal{C}_O(O)$ **do**
6: $\quad \beta'(\mathtt{b}) \leftarrow \mathtt{a}_\otimes$ for every $\mathtt{b} \in \mathcal{C}$
7: $\quad \beta'(\mathtt{b}) \leftarrow \mathtt{e}_\otimes$ for every $\mathtt{b} \in \mathcal{C}_O(O) \backslash \mathcal{C}$
8: $\quad r^{\mathcal{C}} \leftarrow \alpha_B(T, \beta') \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O(O) \backslash \mathcal{C}} \beta(\mathtt{b})$
9: $\quad \alpha_{RB}(T, \beta, O) \leftarrow \alpha_{RB}(T, \beta, O) \oplus r^{\mathcal{C}}$
10: **end for**
11: $\alpha_{RB}(T, \beta, O) \leftarrow \alpha_{RB}(T, \beta, O) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N(O)} \beta(\mathtt{b})$
12: **return** $\alpha_{RB}(T, \beta, O)$

procedure. Then the values of the optional clones are also temporarily modified, and the corresponding bottom-up evaluations are performed. Only then the result is adjusted in such a way that the original values of the necessary clones are taken into account. We now proceed with providing the details.

Algorithm 1 takes as input an attack–defense tree T, an attribute domain A_α, a basic assignment β for α, and a set O of basic actions of the opponent in T. Once the sets of necessary and the optional clones w.r.t. O have been determined, new basic assignments are created. Under each of these assignments β', the clones necessary w.r.t. O receive the neutral element $\mathtt{e}_\otimes$ (in line 3). Intuitively, this ensures that in the final result of the algorithm, the values of β assigned to the necessary clones are taken into account exactly once (with the expression $\bigotimes_{\mathtt{b} \in \mathcal{C}_N(O)} \beta(\mathtt{b})$ in line 11).

In lines 6–7, an assignment β' is created for every subset $\mathcal{C}$ of the set of optional clones $\mathcal{C}_O(O)$. The clones from $\mathcal{C}$ are assigned $\mathtt{a}_\otimes = \mathtt{e}_\oplus$, which intuitively ensures that they are ignored by the bottom-up procedure whenever possible, and the remaining optional clones are assigned $\mathtt{e}_\otimes$ (again, to ensure that their values under β will eventually be counted exactly once; this happens in line 8).

The result of the computations performed in the **for** loop is eventually combined in line 11 with the values assigned to the necessary clones, and the result is returned. The subscript RB in the notation $\alpha_{RB}(T, \beta, O)$ refers to the "repeated bottom-up" evaluation.

Before analyzing the results provided by Algorithm 1 and its complexity, we illustrate its behavior with two examples.

Example 39. *Let T be the tree from Figure 13 and $A_{\mathtt{prob}}$ be the attribute domain for the* maximal probability for the proponent *attribute, given in Table 1. Let β be the basic*

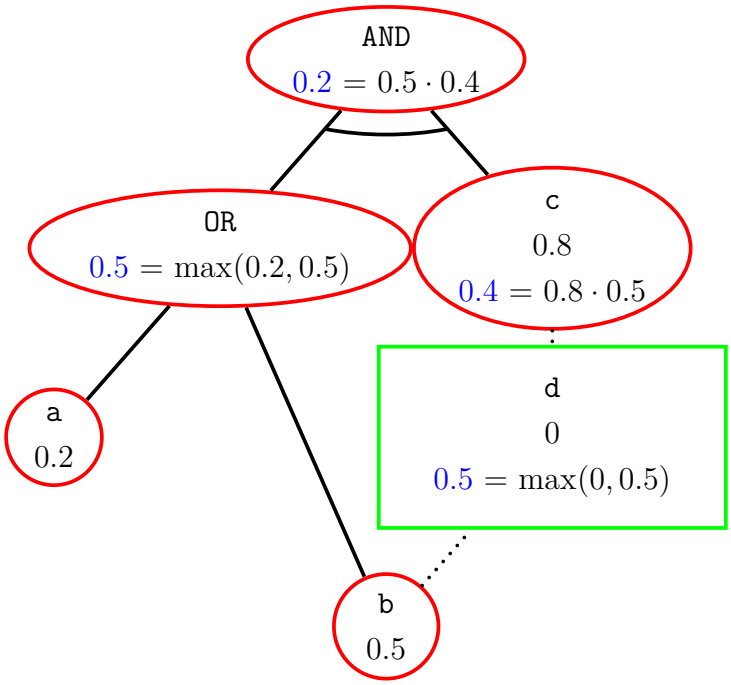


Figure 14: Bottom-up evaluation of the *maximal probability for the proponent* attribute. Values assigned to the basic actions are given in black, values computed at the intermediate nodes – in dark blue

assignment of probability in T as given in Figure 14. Finally, let $O = \{d\}$.

As illustrated in Figure 14, the bottom-up evaluation of prob *in T results in the value of* $\texttt{prob}_B(T, \beta) = 0.2$. *When performing evaluation on the set semantics of T (given in Example 37), one obtains* $\texttt{prob}_{\mathcal{S}}(T, \beta) = 0.4$, *which is the probability of successful execution of both actions* b *and* c.

Consider now the behavior of Algorithm 1. The initialization phase consists of setting

$$\begin{aligned} &\texttt{prob}_{RB}(T, \beta, O) = 0, \\ &\mathcal{C}_N(O) = \{\texttt{b}\}, \\ &\mathcal{C}_O(O) = \emptyset, \end{aligned}$$

and of creating the basic assignment β' which differs from β only in the value assigned to the necessary clone b, *i.e., $\beta'|_{\{\texttt{a},\texttt{c},\texttt{d}\}} \equiv \beta$ and $\beta'(\texttt{b}) = 1$.*

The only subset of the set of clones optional w.r.t. O is the empty set. Therefore, no modification of values takes place in the ***for*** *loop, and the value of* $\texttt{prob}_{RB}(T, \beta)$ *is set in line 9 to*

$$\max(0, r^{\emptyset}) = \max(0, \texttt{prob}_B(T, \beta')) = \max(0, 0.8) = 0.8$$

(see Figure 15 for the bottom-up evaluation of $\texttt{prob}_B(T, \beta')$*). Then, in line 11, the final result of*

$$\texttt{prob}_{RB}(T, \beta, O) = 0.8 \cdot 0.5 = 0.4$$

is obtained. Note that $\texttt{prob}_{RB}(T, \beta, O) = \texttt{prob}_{\mathcal{S}}(T, \beta)$.

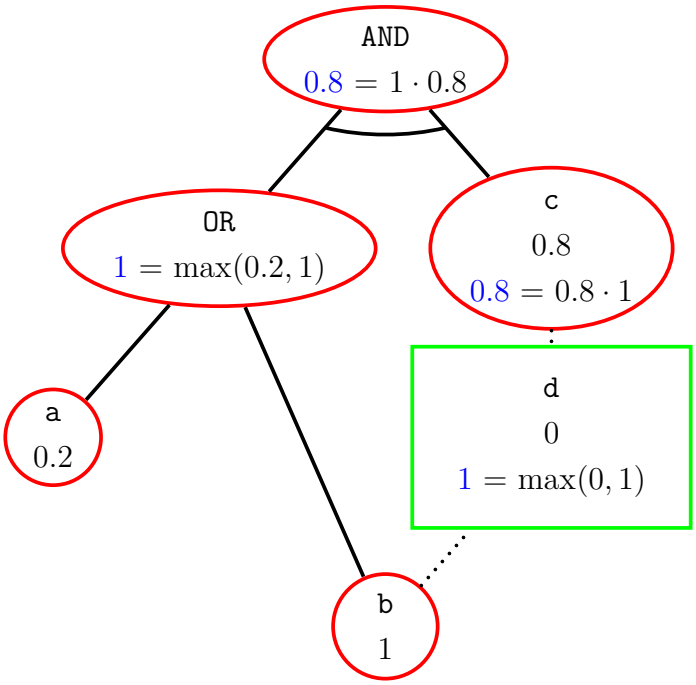


Figure 15: Bottom-up evaluation of the *maximal probability for the proponent* attribute. Values assigned to the basic actions are given in black, values computed at the intermediate nodes – in dark blue

Example 40. *Let T be the attack tree from Figure 12 on page 85, whose set semantics is given in Example 36. Recall the* minimal cost for the proponent *attribute domain given in Table 1; let β be the basic assignment of* minimal cost for the proponent *in T defined as*

$$\beta(\mathtt{a}) = 10, \qquad \beta(\mathtt{b}) = 16,$$
$$\beta(\mathtt{c}) = 10, \qquad \beta(\mathtt{d}) = 5.$$

In this setting, it is easy to compute $\mathtt{cost}_B(T, \beta) = 35$ and $\mathtt{cost}_{\mathcal{S}}(T, \beta) = 25$, the latter value being the cost of execution of the actions $\mathtt{a}, \mathtt{c}$ and $\mathtt{d}$.

Consider now the behavior of Algorithm 1 for T, β, $O = \emptyset$ and the minimal cost for the proponent *attribute domain. The initialization phase consists of setting*

$$\mathtt{cost}_{RB}(T, \beta, O) = +\infty,$$
$$\mathcal{C}_N(O) = \{\mathtt{c}\},$$
$$\mathcal{C}_O(O) = \{\mathtt{b}\},$$

and of creating the basic assignment β' which differs from β only in the value assigned to the necessary clone c*, i.e., $\beta'|_{\{\mathtt{a},\mathtt{b},\mathtt{d}\}} \equiv \beta$ and $\beta'(\mathtt{c}) = 0$.*

The sets $\mathcal{C}$ considered in the ***for*** *loop, their influence on the assignment of cost, and*

their corresponding results $r^{\mathcal{C}}$ *are the following*

$$\begin{aligned} &\mathcal{C} = \emptyset, && \beta'_{\mathtt{cost}}(\mathtt{b}) = 0, && r^{\emptyset} = 21 \\ &\mathcal{C} = \{\mathtt{b}\}, && \beta'_{\mathtt{cost}}(\mathtt{b}) = +\infty, && r^{\{\mathtt{b}\}} = 15. \end{aligned}$$

Thus, after the ***for*** *loop is executed, the value assigned to* $\mathtt{cost}_{RB}(T, \beta)$ *is*

$$\mathtt{cost}_{RB}(T, \beta) = \min(+\infty, 21, 15) = 15,$$

and it is modified in line 11, taking the value of the necessary clone into account, yielding the final result of

$$\mathtt{cost}_{RB}(T, \beta, O) = 15 + 10 = 25.$$

Note that $\mathtt{cost}_{RB}(T, \beta, O) = \mathtt{cost}_{\mathcal{S}}(T, \beta)$.

Example 41. *Let* T *be the attack–defense tree from Figure 2, let* β *be the basic assignment of* minimal time for the proponent *given in Table 3, and let* $O = \mathbb{B}^{\mathrm{o}_T}$. *As illustrated in Example 38,* $\mathcal{C}_N(O) = \emptyset$ *and* $\mathcal{C}_O(O) = \{\mathtt{phish}\}$. *Thus, the sets* $\mathcal{C}$ *considered in the* ***for*** *loop, their influence on the assignment of time, and their corresponding results* r^c *are the following*

$$\begin{aligned} &\mathcal{C} = \emptyset, && \beta'_{\mathtt{cost}}(\mathtt{phish}) = 0, && r^c = 125 \\ &\mathcal{C} = \{\mathtt{phish}\}, && \beta'_{\mathtt{cost}}(\mathtt{phish}) = +\infty, && r^c = 135. \end{aligned}$$

Thus, after the ***for*** *loop is executed, the value assigned to* $\mathtt{cost}_{RB}(T, \beta)$ *is*

$$\mathtt{cost}_{RB}(T, \beta) = \min(+\infty, 125, 135) = 125,$$

and it is returned in line 12.

In Theorem 2 we give sufficient conditions for the result $\alpha_{RB}(T, \beta, O)$ of Algorithm 1 to be equal to the result $\alpha_{\mathcal{S}}(T, \beta)$ of evaluation on the set semantics.

Theorem 2. *Let* T *be an attack–defense tree,* $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ *be a non–increasing attribute domain and let* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$. *If the basic assignment* β *of the attribute* α *satisfies*

$$\begin{aligned} \beta(\mathtt{b}) &= \mathtt{a}_{\otimes} \text{ for } \mathtt{b} \in O, \\ \beta(\mathtt{b}) &= \mathtt{e}_{\otimes} \text{ for } \mathtt{b} \in \mathbb{B}^{\mathrm{o}_T} \backslash O, \end{aligned}$$

then the equality $\alpha_{RB}(T, \beta, O) = \alpha_{\mathcal{S}}(T, \beta)$ *holds.*

Proof. Let $\mathcal{S}(T) = \{(P_1, O_1), \ldots, (P_n, O_n)\}$. Consider the result $r^{\mathcal{C}}$ of the bottom–up procedure obtained in the line 8 of Algorithm 1 for a set $\mathcal{C} \subseteq \mathcal{C}_O(O)$ of optional clones. Due to the values assigned to clones by both basic assignments β and β', we have

$$\begin{aligned} r^{\mathcal{C}} &= \alpha_B(T, \beta') \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O(O) \backslash \mathcal{C}} \beta(\mathtt{b}) \\ &= \alpha_{\mathcal{S}}(T, \beta') \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O(O) \backslash \mathcal{C}} \beta(\mathtt{b}), \end{aligned}$$

by Theorem 1. Thus,

$$r^{\mathcal{C}} = \Big[\bigoplus_{i=1}^{n} \bigotimes_{\mathtt{b}\in P_i\cup O_i} \beta'(\mathtt{b})\Big] \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}} \beta(\mathtt{b}) = \bigoplus_{i=1}^{n} \Big[\Big(\bigotimes_{\mathtt{b}\in P_i\cup O_i} \beta'(\mathtt{b})\Big) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}} \beta(\mathtt{b})\Big].$$

Denote by $r_i^{\mathcal{C}}$ the ith term of the above expression, i.e., set

$$r_i^{\mathcal{C}} := \Big(\bigotimes_{\mathtt{b}\in P_i\cup O_i} \beta'(\mathtt{b})\Big) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}} \beta(\mathtt{b}).$$

Note that if $\mathcal{C}\cap P_i \neq \emptyset$, then $r_i^{\mathcal{C}} = \mathtt{a}_{\otimes}$, due to the values assigned to the clones belonging to $\mathcal{C}$ in the **for** loop. Furthermore, observe that the result of Algorithm 1 is

$$\alpha_{RB}(T,\beta,O) = \Big[\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O(O)} r^{\mathcal{C}}\Big] \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_N} \beta(\mathtt{b}) = \Big(\bigoplus_{i=1}^{n} \Big[\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O(O)} r_i^{\mathcal{C}}\Big]\Big) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_N} \beta(\mathtt{b}). \tag{14}$$

Since $\mathtt{a}_{\otimes} = \mathtt{e}_{\oplus}$, the inner expression can be expanded as

$$\begin{aligned}
\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O(O)} r_i^{\mathcal{C}} &= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i\neq\emptyset}} r_i^{\mathcal{C}} \oplus \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset}} r_i^{\mathcal{C}} \\
&= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset}} \left[\Big(\bigotimes_{\substack{\mathtt{b}\in P_i\\ \mathtt{b}\notin\mathcal{C}_N(O)\cup\mathcal{C}_O(O)}} \beta(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\in P_i\\ \mathtt{b}\in\mathcal{C}_N(O)\cup\mathcal{C}_O(O)\setminus\mathcal{C}}} \mathtt{e}_{\otimes} \otimes \bigotimes_{\mathtt{b}\in O_i} \beta(\mathtt{b})\Big) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}} \beta(\mathtt{b})\right] \\
&= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset}} \left[\Big(\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N(O)\cup\mathcal{C}_O(O)}} \beta(\mathtt{b})\Big) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}} \beta(\mathtt{b})\right] \\
&= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\notin P_i\\ \mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}}} \beta(\mathtt{b})\right],
\end{aligned}$$

where the last transition is a simple regrouping of factors.

Due to the values assigned to the basic actions of the opponent by β, and because $\mathtt{a}_{\otimes} = \mathtt{e}_{\oplus}$, the last expression can be transformed to the form

$$\begin{aligned}
\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O(O)} r_i^{\mathcal{C}} &= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset\\ O\cap O_i=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i\cup O_i\\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\notin P_i\\ \mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}}} \beta(\mathtt{b})\right] \\
&= \bigoplus_{\substack{\mathcal{C}\subseteq\mathcal{C}_O(O)\\ \mathcal{C}\cap P_i=\emptyset\\ O\cap O_i=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i\\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b}) \otimes \bigotimes_{\substack{\mathtt{b}\notin P_i\\ \mathtt{b}\in\mathcal{C}_O(O)\setminus\mathcal{C}}} \beta(\mathtt{b})\right].
\end{aligned}$$

Since the attribute domain is non–increasing, the last "sum" is absorbed by the term corresponding to the set $\mathcal{C}$ for which no $\mathtt{b} \in \big(\mathcal{C}_O(O)\backslash\mathcal{C}\big)\backslash P_i$ exists, i.e., the set $\mathcal{C}$ satisfying $\mathcal{C}_O(O)\backslash\mathcal{C} = P_i \cap \mathcal{C}_O$. The corresponding term is $\bigotimes_{\substack{\mathtt{b}\in P_i \\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b})$. Thus,

$$\bigoplus_{\mathcal{C}\subseteq\mathcal{C}_O(O)} r_i^{\mathcal{C}} = \begin{cases} \bigotimes_{\substack{\mathtt{b}\in P_i \\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b}), \text{ if } O \cap O_i = \emptyset \\ \mathtt{e}_\oplus, \text{ otherwise.} \end{cases}$$

Substituting to (14) yields

$$\begin{aligned} \alpha_{RB}(T,\beta,O) &= \left(\bigoplus_{\substack{i\in\{1,\ldots,n\} \\ O\cap O_i=\emptyset}} \left[\bigotimes_{\substack{\mathtt{b}\in P_i \\ \mathtt{b}\notin\mathcal{C}_N(O)}} \beta(\mathtt{b}) \right] \right) \otimes \bigotimes_{\mathtt{b}\in\mathcal{C}_N(O)} \beta(\mathtt{b}) \\ &= \bigoplus_{\substack{i\in\{1,\ldots,n\} \\ O\cap O_i=\emptyset}} \bigotimes_{\mathtt{b}\in P_i} \beta(\mathtt{b}) \\ &= \bigoplus_{i=1}^{n} \bigotimes_{\mathtt{b}\in P_i\cup O_i} \beta(\mathtt{b}) \\ &= \alpha_{\mathcal{S}}(T,\beta), \end{aligned}$$

where the second equality follows from Definition 28 of necessary clones w.r.t. O, and the third one from the definition of the basic assignment β, i.e., from the fact that $\beta|_O \equiv \mathtt{a}_\otimes = \mathtt{e}_\oplus$. The proof is complete. □

Theorem 2 specifies conditions under which the evaluation of attributes on the set semantics can be replaced with the repeated bottom-up evaluation, i.e., the conditions under which Algorithm 1 can be employed for the purpose of a "what-if" analysis of security scenarios modeled with attack–defense trees. We note that if there are no clones in a given tree, the repeated bottom-up evaluation boils down to a single bottom-up evaluation.

4.4.3 Complexity of repeated bottom-up evaluation of attributes

We now turn our attention to the complexity of Algorithm 1. Among the operations performed in lines 1–4, the most complex one is the initialization of the sets $\mathcal{C}_N(O)$ and $\mathcal{C}_O(O)$. For a tree with n nodes, the time complexity of this step is in $\mathcal{O}(n^2)$, by Lemma 4. The **for** loop from line 5 iterates over all of the subsets of the optional clones, and the most complex of the operations performed within the loop is the bottom-up evaluation, the complexity of which depends on the complexity of operators $\oplus$ and $\otimes$. By combining these considerations with Theorem 2, we get the following result, in which we use $|\beta|$ to denote the number of bits needed for storing the basic assignment β.

Theorem 3. *Let T be an attack–defense tree with n nodes and k repeated basic actions of the proponent. Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be a non–increasing attribute domain*

such that for a basic assignment β for α a single bottom-up computation $\alpha_B(T, \beta)$ is performed in time $\mathcal{O}(f(n, |\beta|))$, for some function $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{R}$. Finally, let $O \subseteq \mathbb{B}^{o_T}$ and let $\beta'\colon \mathbb{B} \to D_\alpha$ be a basic assignment satisfying

$$\beta(\mathtt{b}) = \mathtt{a}_\otimes \text{ for } \mathtt{b} \in O,$$
$$\beta(\mathtt{b}) = \mathtt{e}_\otimes \text{ for } \mathtt{b} \in \mathbb{B}^{o_T} \setminus O.$$

On input (T, A_α, β, O), Algorithm 1 returns $\alpha_{\mathcal{S}}(T, \beta)$ in time

$$\mathcal{O}\left(\max(n^2 + f(n, |\beta|), 2^k f(n, |\beta|))\right).$$

Recall that even in the simplest case of attack trees and *minimal cost for the proponent* attribute, the problem of determining the cost of a cheapest attack is equivalent to solving the *weighted monotone satisfiability problem*, which is known to be NP-complete [BLWC17]. Theorems 1 and 3 indicate that this difficulty originates from the presence of repeated basic actions of the proponent. In particular, the complexity of the repeated bottom-up evaluation is exponential in *the number of repeated basic actions of the proponent*. Therefore, among the trees containing n basic actions the running time of the repeated bottom-up procedure is maximized for a tree in which every basic action is a repeated basic action of the proponent. This is the case, for instance, for the tree

$$T = \mathtt{AND}^{\mathrm{P}}(\mathtt{OR}^{\mathrm{P}}(\mathtt{b}_1, \ldots, \mathtt{b}_n), \mathtt{OR}^{\mathrm{P}}(\mathtt{b}_1, \ldots, \mathtt{b}_n)).$$

Contrarily, in the worst-case, the size of the set semantics, and so the complexity of the evaluation on the set semantics, is exponential in *the total number of nodes*. Nevertheless, the evaluation on the set semantics has at least one advantage over the repeated bottom-up evaluation. If the attribute is such that its value correspond to the execution of exactly one strategy, as it is the case for the *minimal cost for the proponent*, knowing the set semantics allows not only for computing the *value* of the attribute, but also for extracting the strategy for which this value is achieved. In the next section we will demonstrate how, under appropriate assumptions, the two methods of evaluation can be combined for extracting such strategy without creating the whole set semantics of a tree.

4.5 Extraction of optimal strategies

Both the standard bottom-up evaluation and the repeated bottom-up evaluation of attributes are suitable for performing a "what-if" analysis, the result of which is a value of an attribute under specified behavior of the opponent. In many cases, such a value is a solution to an optimization problem, providing an answer to questions such as "what is the *minimal cost* of achieving the root goal?" or "what is the *maximal probability* of achieving the root goal when executing exactly one of the minimal strategies?". The corresponding strategy however, is not obtained. We shall now present a method for

obtaining the strategies corresponding to such optimal values, in a way that, if possible, does not involve the creation of the set semantics of a tree. We begin with defining the object that we want to extract from a tree.

Definition 29 (Optimal strategy)**.** *Let* $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ *be a non-increasing attribute domain with* $D_\alpha \subseteq \mathbb{R}$ *and with* $\oplus$ *being the operation of taking maximum or minimum*[4]*. A pair* $(P, O) \in \mathcal{S}(T)$ *is a* strategy in T optimal w.r.t. α under the basic assignment β *if*

$$\alpha_{\mathcal{S}}(T, \beta) = \bigotimes_{\mathtt{b} \in P \cup O} \beta(\mathtt{b}).$$

Example 42. *As illustrated in Example 27, the strategy* $(\{\mathtt{phish}, \mathtt{phone}, \mathtt{log\&trans}\}, \emptyset)$ *is optimal in the tree* T *from Figure 2 w.r.t.* minimal time for the proponent *attribute, under the basic assignment* $\beta_{\mathtt{time}}$ *given in Table 3.*

4.5.1 Tree pruning procedure

Our method for determining optimal strategies relies on the repeated bottom-up evaluation of attributes and the following lemma.

Lemma 5. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree,* A_α *be an attribute domain induced by a semiring* $(D_\alpha, \oplus, \otimes)$*, and let* β *be a basic assignment of* α *satisfying* $\alpha_B(T, \beta) \neq \mathtt{a}_\otimes$*. For* $T' = (V', A', \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *being the component containing the root of* T *of the subdag of* T *induced by the set*

$$\{v \in V : \alpha_B(T, \beta, v) \neq \mathtt{a}_\otimes\},$$

the equality

$$\alpha_B(T, \beta) = \alpha_B(T', \beta)$$

holds.

Proof. We shall prove that for every node $v \in V'$ the equality $\alpha_B(T', \beta, v) = \alpha_B(T, \beta, v)$ holds. The proof is by induction on the structure of the subdag $T(v)$. For the base case, assume that v is not refined in T and has no countermeasure attached in T. Then, v is also not refined and has no countermeasure attached in T'. Thus, $\alpha_B(T', \beta, v) = \beta(\lambda(v)) = \alpha_B(T, \beta, v)$.

Assume now that $\mathrm{ref}(v) \neq \mathtt{N}$ or that $\bar{v}$ exists in T. Recall that for $\mathrm{s} = \mathrm{actor}(v)$ and $\mathtt{OP} = \mathrm{ref}(v)$ the value of $\alpha_B(T, \beta, v)$ is

$$\alpha_B(T, \beta, v) = \begin{cases} \beta(\lambda(v)), \text{ if } \mathtt{OP} = \mathtt{N} \text{ and } \bar{v} \text{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}_\alpha\left(\beta(\lambda(v)), \alpha_B(T, \beta, \bar{v})\right), \text{ if } \mathtt{OP} = \mathtt{N} \text{ and } \bar{v} \text{ exists,} \\ (\mathtt{OP}^{\mathrm{s}}_\alpha)_{v' \in \mathrm{children}_T(v)} \alpha_B(T, \beta, v'), \text{ if } \mathtt{OP} \neq \mathtt{N} \text{ and } \bar{v} \text{ does not exist,} \\ \mathtt{C}^{\mathrm{s}}_\alpha\left((\mathtt{OP}^{\mathrm{s}}_\alpha)_{v' \in \mathrm{children}_T(v)} \alpha_B(T, \beta, v'), \alpha_B(T, \beta, \bar{v})\right), \text{ otherwise.} \end{cases}$$

[4]Note that all of the attribute domains from Table 1 other than the *satisfiability* domain are of this form.

We consider three cases.

Case 1. The node v is not refined and $\bar{v}$ exists.

In this case, $\alpha_B(T,\beta,v) = \mathtt{C}^{\mathrm{s}}_{\alpha}\left(\beta(\lambda(v)), \alpha_B(T,\beta,\bar{v})\right)$. Thus, if $\bar{v} \in V'$, then the required equality follows from the induction hypothesis. Otherwise, $\alpha_B(T',\beta,v) = \beta(\lambda(v))$ and $\alpha_B(T,\beta,\bar{v}) = \mathtt{a}_{\otimes}$. Since $v \in V'$, we have $\alpha_B(T,\beta,v) \neq \mathtt{a}_{\otimes}$, which combined with the fact that $\mathtt{a}_{\otimes} = \mathtt{e}_{\oplus}$ implies that $\mathtt{C}^{\mathrm{s}}_{\alpha} = \oplus$. Hence, $\alpha_B(T,\beta,v) = \beta(\lambda(v)) \oplus \mathtt{e}_{\oplus} = \beta(\lambda(v)) = \alpha_B(T',\beta,v)$.

Case 2. The node v is not refined and $\bar{v}$ does not exist.

In this case, the value of $\alpha_B(T,\beta,v)$ is $(\mathtt{OP}^{\mathrm{s}}_{\alpha})_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v')$. If $\mathrm{children}_T(v) \subset V'$, then the required equality follows from the induction hypothesis. Otherwise, there is $v' \in \mathrm{children}(v)$ such that $\alpha_B(T,\beta,v') = \mathtt{a}_{\otimes}$. Since $\alpha_B(T,\beta,v) \neq \mathtt{a}_{\otimes}$ and $\mathtt{a}_{\otimes} = \mathtt{e}_{\oplus}$, it follows again that the operation performed by the bottom-up evaluation at the node v in T is $\oplus$, i.e., $\mathtt{OP}^{\mathrm{s}}_{\alpha} = \oplus$. Thus,

$$\begin{aligned}\alpha_B(T,\beta,v) &= \bigoplus_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v') = \bigoplus_{v' \in \mathrm{children}_{T'}(v)} \alpha_B(T,\beta,v') \\ &= \alpha_B(T',\beta,v).\end{aligned}$$

Case 3. The node v is refined and $\bar{v}$ exists in T.

Under the assumptions of this case, the equality

$$\alpha_B(T,\beta,v) = \mathtt{C}^{\mathrm{s}}_{\alpha}\left((\mathtt{OP}^{\mathrm{s}}_{\alpha})_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v'), \alpha_B(T,\beta,\bar{v})\right)$$

holds. Similarly as in the previous cases, if all of the children of v in T belong to V', then the claim follows from the induction hypothesis. If this is not the case, then $V' \cap (\mathrm{children}_T(v) \cup \{\bar{v}\}) \neq \emptyset$. Note that regardless of whether or not there is a node $v' \in \mathrm{children}_T(v)$ not belonging to V', by repeating the reasoning from the proof Case 2 one obtains the equality $(\mathtt{OP}^{\mathrm{s}}_{\alpha})_{v' \in \mathrm{children}_T(v)} \alpha_B(T,\beta,v') = (\mathtt{OP}^{\mathrm{s}}_{\alpha})_{v' \in \mathrm{children}_{T'}(v)} \alpha_B(T,\beta,v')$. Combining this equality with the reasoning from the proof of Case 1 leads to the claimed statement. □

4.5.2 Tree reduction technique preserving optimal strategies

The idea behind our method for determining the optimal strategies is the following. Should the result of the bottom-up evaluation be equal to that of evaluation on the set semantics, one could apply Lemma 5 repetitively, thus reducing the size of the tree while keeping the result of the bottom-up evaluation unchanged. If the equality of the results provided by the two evaluation methods was maintained after each application of Lemma 5, the eventually obtained tree would contain an optimal strategy that is also optimal in the original tree. Hopefully, after the reduction is performed, the set semantics

of the final tree is significantly smaller than that of the original tree, and can be computed easily.

The starting point of the procedure sketched above, and given in detail in Algorithm 2, is the repeated bottom-up evaluation. Note that for an attribute domain A_α as in Definition 29, the operation performed in line 9 of the repeated bottom-up evaluation consists of setting the value of $\alpha_{RB}(T,\beta,O)$ to be the minimum/maximum of the currently stored value and the result of the bottom-up procedure performed in the current iteration of the **for** loop. Algorithm 1 could be therefore modified, so that along the optimal value, the set $\mathcal{C}$ of optional clones corresponding to the iteration in which the value has been obtained is stored. Suppose that the pair $(\alpha_{RB}(T,\beta,O),\mathcal{C})$ is returned by such modified algorithm, for a tree $T=(V,A,\mathrm{L},\lambda,\mathrm{actor},\mathrm{ref})$, an attribute domain A_α, a basic assignment β and a set O satisfying the assumptions of Theorem 2. Note that $\alpha_{RB}(T,\beta,O)$ is then

$$\alpha_{RB}(T,\beta,O)=\alpha_B(T,\beta')\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\backslash\mathcal{C}}\beta(\mathtt{b})\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_N(O)}\beta(\mathtt{b}), \tag{15}$$

where

$$\beta'(\mathtt{b})=\begin{cases}\beta(\mathtt{b}),\ \text{if } \mathtt{b}\in\mathbb{B}_T\backslash(\mathcal{C}_N(O)\cup\mathcal{C}_O(O)),\\ \mathtt{e}_\otimes,\ \text{if } \mathtt{b}\in\mathcal{C}_N(O)\cup(\mathcal{C}_O(O)\backslash\mathcal{C}),\\ \mathtt{a}_\otimes,\ \text{if } \mathtt{b}\in\mathcal{C}.\end{cases}$$

Assume that $\alpha_{RB}(T,\beta,O)\neq\mathtt{a}_\otimes$, since otherwise there is no need for optimization: if $\alpha_{RB}(T,\beta,O)=\mathtt{a}_\otimes$, then $\alpha_{\mathcal{S}}(T,\beta)=\mathtt{a}_\otimes$, implying that for every strategy $(\overline{P},\overline{O})\in\mathcal{S}(T)$ the equality $\bigoplus_{\mathtt{b}\in\overline{P}\cup\overline{O}}=\mathtt{a}_\otimes$ holds. This assumption implies that none of the actions from the set $\mathcal{C}_N(O)\cup(\mathcal{C}_O(O)\backslash\mathcal{C})$ is assigned $\mathtt{a}_\otimes$ under the basic assignment β, and that the value of $\alpha_B(T,\beta')$ is also different from $\mathtt{a}_\otimes$. Thus, Lemma 5 can be applied to T, A_α and β'.

Let T' be the tree obtained from T as described in Lemma 5. Then

$$\begin{aligned}\alpha_{\mathcal{S}}(T,\beta)&=\alpha_B(T,\beta')\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\backslash\mathcal{C}}\beta(\mathtt{b})\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_N(O)}\beta(\mathtt{b})\\&=\alpha_B(T',\beta')\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\backslash\mathcal{C}}\beta(\mathtt{b})\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_N(O)}\beta(\mathtt{b})\\&=\alpha_{\mathcal{S}}(T',\beta')\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_O(O)\backslash\mathcal{C}}\beta(\mathtt{b})\otimes\bigotimes_{\mathtt{b}\in\mathcal{C}_N(O)}\beta(\mathtt{b}),\end{aligned} \tag{16}$$

where the first equality follows from (15) and Theorem 2, the second one from Lemma 5, and the last one from Theorem 1 and the fact that under the basic assignment β' every clone in T' is assigned either $\mathtt{a}_\otimes$ or $\mathtt{e}_\otimes$.

Note that for every node v in T', the value of $\alpha_B(T',\beta',v)$ is different from $\mathtt{a}_\otimes$. To reduce the size of T' further, consider a basic action $\mathtt{b}$ in $\mathbb{B}_{T'}$ that does not belong to the set $\mathcal{C}_N(T,O)\cup\mathcal{C}_O(T,O)$. Create a new basic assignment for α, say, β'', that differs from β' in that it assigns $\mathtt{a}_\otimes$ to $\mathtt{b}$. That is, define β'' with $\beta''|_{\mathbb{B}_{T'}\backslash\{\mathtt{b}\}}\equiv\beta'$ and $\beta''(\mathtt{b})=\mathtt{a}_\otimes$. Under this new basic assignments, there is at least one node v in T' for which

$\alpha_B(T', \beta'', v) = \mathtt{a}_\otimes$. Thus, if $\alpha_B(T', \beta'') \neq \mathtt{a}_\otimes$, Lemma 5 can be applied again, reducing tree T' to a smaller tree, say T''. Such a reduction might not be beneficial: it should be performed only if $\alpha_B(T', \beta'') = \alpha_B(T', \beta')$, as otherwise it might happen that all the optimal strategies in T'' under β are suboptimal in T. Thus, if $\alpha_B(T', \beta'') = \alpha_B(T', \beta')$, apply Lemma 5 to T', obtaining T''. Since both β' and β'' satisfy the assumptions of Theorem 1, the equalities $\alpha_B(T', \beta') = \alpha_\mathcal{S}(T', \beta')$ and $\alpha_B(T'', \beta'') = \alpha_\mathcal{S}(T'', \beta'')$ hold. Furthermore, Lemma 5 implies that the equality $\alpha_B(T'', \beta'') = \alpha_B(T', \beta'')$ holds, and the definition of T'' and the definition of the basic assignment β'' imply that $\alpha_\mathcal{S}(T'', \beta'') = \alpha_\mathcal{S}(T'', \beta)$. To summarize, we have

$$\alpha_\mathcal{S}(T', \beta') = \alpha_B(T', \beta') = \alpha_B(T', \beta'') = \alpha_B(T'', \beta'') = \alpha_\mathcal{S}(T'', \beta'') = \alpha_\mathcal{S}(T'', \beta).$$

Substituting $\alpha_\mathcal{S}(T'', \beta)$ to (16) for $\alpha_\mathcal{S}(T', \beta')$ yields

$$\alpha_\mathcal{S}(T, \beta) = \alpha_\mathcal{S}(T'', \beta) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O(O) \setminus \mathcal{C}} \beta(\mathtt{b}) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N(O)} \beta(\mathtt{b}).$$

Thus, if the pair $(\overline{P}, \overline{O})$ is an optimal strategy in T'' w.r.t. α under the basic assignment β, then the pair $(\overline{P} \cup \mathcal{C}_N(O) \cup \mathcal{C}_O(O), \overline{O})$ is *as good as an optimal strategy in* T, in the sense that for any optimal strategy $(\hat{P}, \hat{O})$ in T the equality

$$\bigotimes_{\mathtt{b} \in \hat{P} \cup \hat{O}} \beta(\mathtt{b}) = \bigotimes_{\mathtt{b} \in (\overline{P} \cup \mathcal{C}_N(O) \cup \mathcal{C}_O(O) \setminus \mathcal{C}, \overline{O})} \beta(\mathtt{b}) \tag{17}$$

holds.

The procedure described in the previous paragraph can be now performed again, for a basic action $\mathtt{b}$ in $\mathbb{B}_{T''}$ not belonging to the set $\mathcal{C}_N(T, O) \cup \mathcal{C}_O(T, O)$. This yields another, hopefully smaller tree, in which the procedure can be repeated again. Eventually, a tree T'' will be obtained, in which the procedure can no longer be applied, i.e., in which switching a value assigned to any of the actions not in $\mathcal{C}_N(T, O) \cup \mathcal{C}_O(T, O)$ to $\mathtt{a}_\otimes$ switches the result of the bottom-up evaluation to $\mathtt{a}_\otimes$. Since the equality (17) holds for this final tree T'', computing its set semantics allows for determining a strategy that is *as good as an optimal strategy in* T, in the sense explained above.

The above reasoning proves the following.

Theorem 4. *Let T be an attack–defense tree, $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be a non–increasing attribute domain with $D_\alpha \subseteq \mathbb{R}$ and with $\oplus$ being the operation of taking maximum or minimum, and let $O \subseteq \mathbb{B}^{\mathrm{o}_T}$. If the basic assignment β of the attribute α satisfies*

$$\beta(\mathtt{b}) = \mathtt{a}_\otimes \text{ for } \mathtt{b} \in O,$$
$$\beta(\mathtt{b}) = \mathtt{e}_\otimes \text{ for } \mathtt{b} \in \mathbb{B}^{\mathrm{o}_T} \setminus O,$$

and the tree T' is the output of Algorithm 2 on input T, A_α, β, O, then the equality

$$\alpha_\mathcal{S}(T, \beta) = \alpha_\mathcal{S}(T', \beta) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_O(T,O) \setminus \mathcal{C}} \beta(\mathtt{b}) \otimes \bigotimes_{\mathtt{b} \in \mathcal{C}_N(T,O)} \beta(\mathtt{b})$$

holds.

We believe that the following, stronger statement, that we are currently unable to prove, is true.

Conjecture 1. *Let T be an attack–defense tree, $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be a non–increasing attribute domain with $D_\alpha \subseteq \mathbb{R}$ and with $\oplus$ being the operation of taking maximum or minimum, and let $O \subseteq \mathbb{B}^{o_T}$. If the basic assignment β of the attribute α satisfies*

$$\beta(\mathtt{b}) = \mathtt{a}_\otimes \text{ for } \mathtt{b} \in O,$$
$$\beta(\mathtt{b}) = \mathtt{e}_\otimes \text{ for } \mathtt{b} \in \mathbb{B}^{o_T} \backslash O,$$

and the tree T' is the output of Algorithm 2 on input T, A_α, β, O, then every strategy in T'' optimal w.r.t. α under the basic assignment β is also an optimal strategy in T.

Algorithm 2 Tree reduction preserving optimal strategies

Input: Attack–defense tree $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$, attribute domain $(D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$, basic assignment $\beta\colon \mathbb{B} \to D_\alpha$, set $O \subseteq \mathbb{B}^{o_T}$

Output: Attack–defense tree $T' = T'(T, A_\alpha, \beta, O)$

1: initialize $\mathcal{C}_N(T, O)$, $\mathcal{C}_O(T, O)$
2: set $(\alpha_{RB}(T, \beta, O), \mathcal{C})$ to be as in text
3: **if** $\alpha_{RB}(T, \beta, O) = \mathtt{a}_\otimes$ **then**
4: return T
5: **end if**
6: $\beta'(\mathtt{b}) \leftarrow \beta(\mathtt{b})$ for every $\mathtt{b} \in \mathbb{B}_T \backslash (\mathcal{C}_N(T, O) \cup \mathcal{C}_O(T, O))$
7: $\beta'(\mathtt{b}) \leftarrow \mathtt{e}_\otimes$ for every $\mathtt{b} \in \mathcal{C}_N(T, O) \cup (\mathcal{C}_O(T, O) \backslash \mathcal{C})$
8: $\beta'(\mathtt{b}) \leftarrow \mathtt{a}_\otimes$ for every $\mathtt{b} \in \mathcal{C}$
9: $T' \leftarrow$ the connected component containing the root of T of the subdag of T induced by the set $\{v \in V\colon \alpha_B(T, \beta, v) \neq \mathtt{a}_\otimes\}$ that contains the root of T
10: **while** there is $\mathtt{b} \in \mathbb{B}_{T'} \backslash (\mathcal{C}_N(T, O) \cup \mathcal{C}_O(T, O))$ such that $\alpha_B(T', \beta'') \neq \mathtt{a}_\otimes$ for β'' defined with $\beta''|_{\mathbb{B}_{T'} \backslash \{\mathtt{b}\}} \equiv \beta'$ and $\beta''(\mathtt{b}) = \mathtt{a}_\otimes$ **do**
11: $\mathtt{b} \leftarrow$ one of the basic actions satisfying the condition in line 10
12: $\beta''(\mathtt{b}) \leftarrow \mathtt{a}_\otimes$
13: $\beta''(\mathtt{b}') \leftarrow \beta'(\mathtt{b}')$ for every $\mathtt{b}' \in \mathbb{B}_{T'} \backslash \{\mathtt{b}\}$
14: **if** $\alpha_B(T', \beta'') = \alpha_B(T', \beta')$ **then**
15: $T'' \leftarrow$ the connected component containing the root of T' of the subdag of T' induced by the set $\{v \in V\colon \alpha_B(T', \beta'', v) \neq \mathtt{a}_\otimes\}$
16: $T' \leftarrow T''$
17: **end if**
18: **end while**
19: **return** T'

Before analyzing the complexity of Algorithm 2, we illustrate its behavior with two examples.

Example 43. *Let* $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{11}, \mathtt{b}_{12}, \mathtt{b}_{13}), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{21}, \mathtt{b}_{22}, \mathtt{b}_{23})\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{31}, \mathtt{b}_{32}, \mathtt{b}_{33}))$ *be the attack tree depicted in Figure 11. Let* β *be the basic assignment for* minimal cost for the proponent *defined as* $\beta(\mathtt{b}_{ij}) = 4 - j$*, for* $i \in \{1, 2, 3\}, j \in \{1, 2, 3\}$ *(see Figure 16). Note that there are* 27 *elements in the set semantics of* T*, the value of* $\mathtt{cost}_{\mathcal{S}}(T, \beta)$ *is*

$$\mathtt{cost}_{\mathcal{S}}(T, \beta) = \mathtt{cost}_{RB}(T, \beta) = \mathtt{cost}_{B}(T, \beta) = 1 + 1 + 1 = 3,$$

and that the optimal strategy in T *w.r.t.* minimal cost for the proponent *under* β *is* $(\{\mathtt{b}_{13}, \mathtt{b}_{23}, \mathtt{b}_{33}\}, \emptyset)$.

Let $T, \beta, O = \emptyset$ *and the domain for* minimal cost for the proponent *be the input for Algorithm 2. In line 2,* $\mathcal{C}$ *is set to be the empty set. Since there are no clones in* T *and* $\mathcal{C} = \emptyset$*, no modification of the basic assignment* β *takes place in lines 6–8. That is, after line 8 is executed,* $\beta' \equiv \beta$*. Similarly, in line 9,* T' *is set to be* T*.*

Assume that in the ***while*** *loop, the search for a candidate basic action* $\mathtt{b}$ *satisfying the condition in line 10 is performed starting from the lowest index possible, and progressing towards the highest. Thus, when the algorithm enters the* ***while*** *loop for the first time,* $\mathtt{b}$ *is set in line 11 to be* $\mathtt{b}_{11}$*. Then, in lines 12 and 13, the basic assignment* β'' *is defined with* $\beta''|_{\mathbb{B}_{T'} \setminus \{\mathtt{b}_{11}\}} \equiv \beta'$ *and* $\beta''(\mathtt{b}_{11}) = +\infty$*. Thus,* $\mathtt{cost}_{B}(T', \beta'') = \mathtt{cost}_{B}(T', \beta') = 3$*, and in line 15 the node labeled* $\mathtt{b}_{11}$ *is removed from* T'*. Hence, after the* ***while*** *loop is executed for the first time, we have*

$$T' = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{12}, \mathtt{b}_{13}), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{21}, \mathtt{b}_{22}, \mathtt{b}_{23})\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{31}, \mathtt{b}_{32}, \mathtt{b}_{33})).$$

It is easy to see that in the next iteration of the ***while*** *loop the node* $\mathtt{b}_{12}$ *is removed from* T' *in the same manner, reducing the tree to*

$$T' = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{13}), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{21}, \mathtt{b}_{22}, \mathtt{b}_{23})\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{31}, \mathtt{b}_{32}, \mathtt{b}_{33})).$$

Now setting the value assigned to $\mathtt{b}_{13}$ *to* $+\infty$ *results in the bottom-up evaluation on* T' *returning* $+\infty$*. Thus,* $\mathtt{b}_{13}$ *is not a viable candidate for the action* $\mathtt{b}$ *in line 11. The next new* $\mathtt{b}$ *is therefore* $\mathtt{b} = \mathtt{b}_{21}$*.*

Due to the simple structure of T*, it is easy to see what will happen next: the nodes labeled with the basic actions* $\mathtt{b}_{21}$*,* $\mathtt{b}_{22}$*,* $\mathtt{b}_{31}$ *and* $\mathtt{b}_{32}$ *will be removed from* T'*, one by one. Thus, the tree returned by Algorithm 2 is*

$$T' = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{13}), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{23})\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{33})).$$

The set semantics of this tree is $\mathcal{S}(T') = \{(\{\mathtt{b}_{13}, \mathtt{b}_{23}, \mathtt{b}_{33}\}, \emptyset)\}$*, i.e., it is a singleton consisting of the optimal strategy in* T *w.r.t.* minimal cost for the proponent *under* β*.*

The previous example demonstrates that Algorithm 2 sometimes allows for transforming a tree having set semantics of size exponential in the tree size into a tree having very small number of strategies, while retaining at least one of the strategies optimal in the original tree in the set semantics. We now illustrate the behavior of Algorithm 2 on our running example.

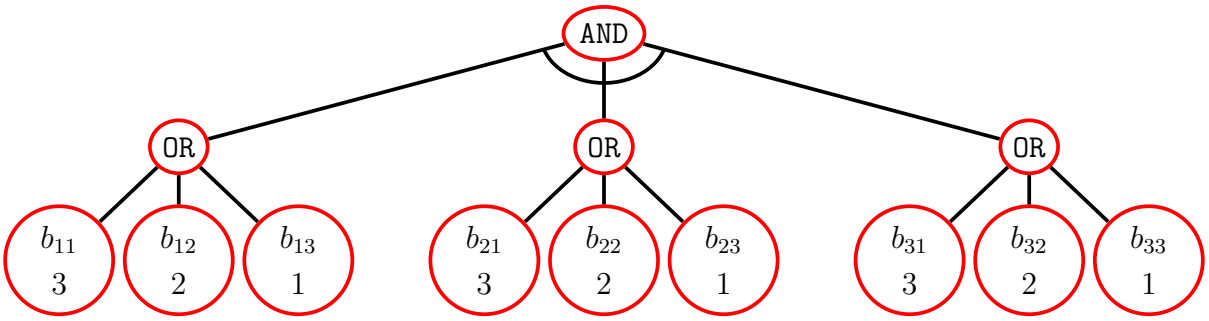


Figure 16: A tree with the size of the set semantics exponential in the number of basic actions, with the basic assignment of *minimal cost for the proponent* given in the nodes labeled with basic actions

Example 44. *Let T be the attack–defense tree from Figure 2, and let β be the basic assignment of* minimal time for the proponent *that assigns $+\infty$ to the basic actions of the opponent and the values given in Table 3 to the basic actions of the proponent. Recall that there are eleven strategies in T (listed in Example 23) and that the optimal strategy in T w.r.t. minimal time under β is* $(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$ *(see, e.g., Example 27).*

Let $O = \mathbb{B}^{o_T}$. Then, $\mathcal{C}_N(O) = \emptyset$ and $\mathcal{C}_O(O) = \{\texttt{phish}\}$. The (optimal) value returned by the repeated bottom-up evaluation of minimal time for the proponent *on T under β is* 125*, and it is obtained when in the* ***for*** *loop of Algorithm 1 the set $\mathcal{C} = \emptyset$ is considered (see Example 41. Thus, after the first nine lines of Algorithm 2 are executed on input T, β, O and the* minimal time for the proponent *attribute domain, we have*

$$\mathcal{C}_N(O) = \emptyset, \mathcal{C}_O(O) = \{\texttt{phish}\}, \mathcal{C} = \emptyset,$$

the assignment β' differs from β only in the value assigned to the action `phish`*, which is $\beta'(\texttt{phish}) = 0$, and T' is obtained by removing the nodes labeled* `pwd` *and* `spwd` *from T.*

Observe that $\texttt{time}_B(T, \beta') = 25$, and that this value comes from the subdag of T rooted in the node labeled ***via online banking****. Thus, setting the value assigned to any of the basic actions from the subdag of T rooted in the node labeled* ***via ATM*** *to $+\infty$ does not change the result of the bottom-up evaluation. Hence, after all of these basic actions are considered in the* ***while*** *loop, the subdag rooted in the* ***via ATM*** *node is removed from T.*

On the other hand, if the value of any of the basic actions `phone`, `log&trans` *and* `sms` *is set to $+\infty$, the value computed at the node labeled* ***via online banking*** *with the bottom-up evaluation will be $+\infty$, and so the value computed at the root will be different than $\texttt{time}_B(T, \beta')$. Thus, when any of these three basic actions is considered in the* ***while*** *loop, no modification of the tree occurs.*

Finally, setting the value assigned to the basic action `uname` *to $+\infty$ results in the node labeled with that action being removed from T.*

Thus, Algorithm 2 returns the tree T' depicted in Figure 17. The set semantics of this tree contains two strategies: the strategy $(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$ *and the strategy* $(\{\texttt{phish}, \texttt{log\&trans}\}, \texttt{sms})$

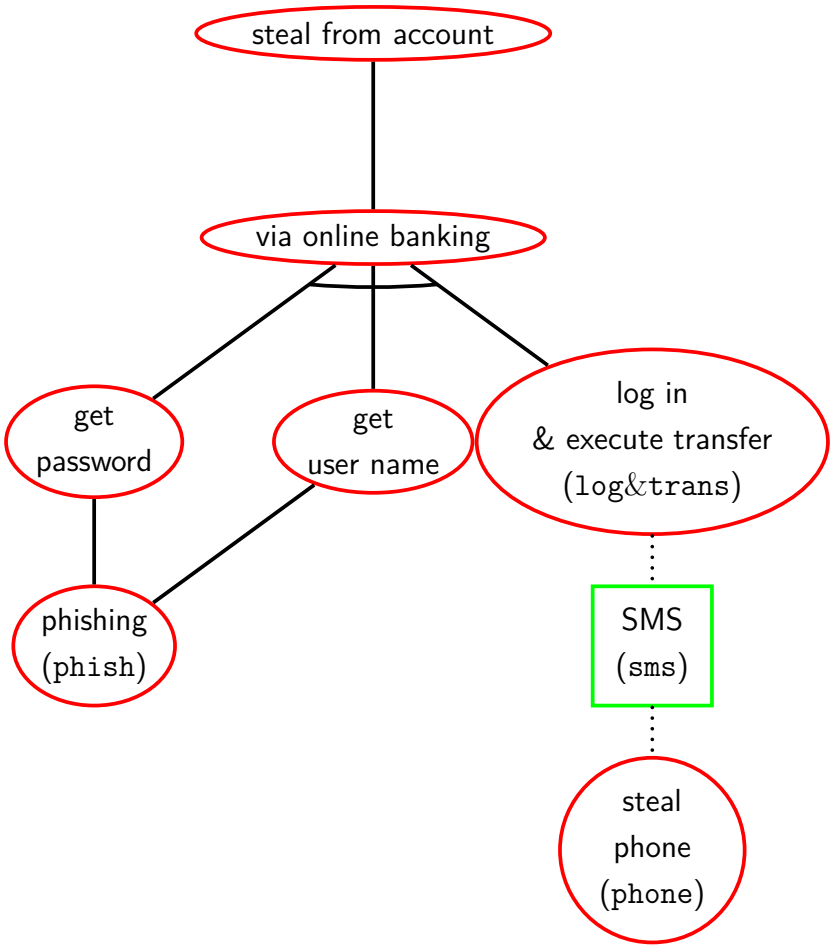


Figure 17: An attack–defense tree obtained by applying Algorithm 2 to the attack–defense tree from the running example

4.5.3 Complexity of Algorithm 2

Having illustrated the usefulness of Algorithm 2, we now turn to analyzing its complexity. For this purpose, assume that T is a tree on n nodes, containing k repeated basic actions of the proponent. Let $A_\alpha = (D_\alpha, \oplus, \otimes, \otimes, \oplus, \otimes, \oplus)$ be a non–increasing attribute attribute domain with $D_\alpha \subseteq \mathbb{R}$ and with $\oplus$ being the operation of taking maximum or minimum. Assume that for a basic assignment β for α a single bottom-up computation $\alpha_B(T, \beta)$ is performed in time $\mathcal{O}(f(n, |\beta|))$[5] for some function $f\colon \mathbb{N} \times \mathbb{N} \to \mathbb{R}$.

Among the operation performed by Algorithm 2 in lines 1 – 8 the most complex one is the repeated bottom-up evaluation, performed in time $\mathcal{O}\left(\max(n^2 + f(n, |\beta|), 2^k f(n, |\beta|))\right)$. The tree T' from line 9 can be identified in time linear in n, using a variant of graph traversal algorithm.

Since there are n nodes in T, the operation within the **while** loop will be performed at most n times. In the worst case, checking the **while** condition itself will require $n \cdot f(n, |\beta|)$ operations. Once this condition is checked, the value of $\alpha_B(T', \beta'')$ is known, and so the

[5]Similarly as in Theorem 2, $|\beta|$ denotes here the number of bits needed for storing the basic assignment β.

only operation performed in line 14 is comparison of two real numbers. Determination of T'' in line 15 is performed again in time linear in n, assuming that the values obtained at the intermediate nodes when computing $\alpha_B(T', \beta'')$ are stored once the algorithm enter the **while** loop.

It follows that Algorithm 2 will terminate in time $\mathcal{O}\left(\max(n^3 f(n, |\beta|), 2^k f(n, |\beta|))\right)$.

4.6 Relations to other formalisms

Attack and attack–defense trees are only one of many modeling frameworks employing `AND`/`OR` trees. In this section, we discuss relations between the results presented in this chapter and some of the other similar formalisms. We do not argue that the applications presented here are necessarily useful *per se*. We believe that their value lies in demonstrating that it is worthwhile to try to reformulate analysis methods developed in terms of one formalism in the language of another one.

4.6.1 Fault trees

Fault trees [HRVG81, RS15] is a modeling framework for depicting and studying dependencies between elements of complex systems. In their simplest form, they are syntactically equivalent to attack trees: they are directed acyclic graphs with leaf nodes corresponding to failures of system components (*basic events*) and the refined nodes (*gates*) modeling failures propagation throughout the system. *Static fault trees* (SFTs) admit three types of gates: `AND` gates, `OR` gates and *k-out-of-n* gates; the latter can be modeled using only `AND` and `OR` gates [RS15]. Thus, we shall consider SFTs to be attack trees.

In the field of fault trees analysis it is standard to interpret repeated basic events (called *shared basic events*) as clones [Ste86, Cod06, RS15]. The semantics of an SFT, as given in [RS15], describes for a given set of basic events that have occurred (equivalently, a set of components that have failed), for each element (gate or basic event) in the tree, whether the component or subsystem corresponding to this element failed. For a given set S of basic events that occurred, the semantics of an element represented with a node v in the fault tree T is achieved$_{\mathrm{T}}(v, S)$. A minimal set S of basic events for which achieved$_{\mathrm{T}}(v, S) = 1$ is called a *minimal cut set* (MCS) in T.

Minimal cut sets in SFTs are thus equivalent to minimal strategies in attack trees. Therefore, the necessary clones in an SFT are the events that are present in every MCS. Furthermore, by Proposition 2, MCSs belong to the set semantics of an SFT[6]. One of the reliability characteristics of MCSs considered in [HRVG81] is *minimal cut set unavailability*, which is the probability that all the basic events in the MCS occur. In the particular case when each of the basic events is assigned a constant probability of occurrence, and

[6]This is fact is neither suprising nor new. A method for minimal cut sets determination equivalent to the creation of set semantics has been given already in [HRVG81], Chapter XI.

under the assumption that the basic events are independent, the unavailability of a MCS is computed as the product of probabilities of its elements. Together with Theorem 2 and the fact that the attribute domain $([0, 1], \max, \cdot)$ is non-increasing, this implies that the maximal value of MCS unavailability over all the MCSs can be computed in SFTs containing shared subtrees using the repeated bottom-up evaluation; an MCS achieving this value can be extracted from the SFT using Algorithm 2.

When applying the repeated bottom-up evaluation to an SFT and the *minimal cost for the proponent* attribute, with all the basic events being assigned 1, one obtains the size of the smallest of all the MCSs in the SFT. Again, Algorithm 2 can be used for extracting such MCS from the SFT without extracting all of MCSs.

Finally, combining Algorithm 1 with the framework developed in Chapter 5 allows for determining pairs of the form (size, unavailability) corresponding to MCSs optimal (in the sense of Pareto optimality) w.r.t. both the size and unavailability, i.e., minimizing the former while maximizing the latter.

4.6.2 Weighted monotone satisfiability problem

In the case of attack trees, the problem of determining the value of a cheapest strategy is equivalent to determining the minimal sum of costs (or *weights*) assigned to propositional variables of an `AND`/`OR` propositional formula, over all sets of variables satisfying this formula. Determining this value solves the *weighted monotone satisfiability problem* [BLWC17]. Algorithm 2 can be used for reducing such monotone propositional formula to a smaller form, from which the set of variables corresponding to the optimal value could be extracted using the set semantics.

4.6.3 Attack graphs

In the work [WNJ06], the authors tackle the problem of determining the most cost-effective ways of increasing the security of networks (*network hardening* problem). They employ a variant of *attack graphs* for modeling dependencies between the *security conditions* related to hosts (e.g., existence of a vulnerability or existence of an established connection) and the possible *exploits*. Their goal is to determine a set of initially satisfied security conditions that should be disabled in order to secure the network, at the lowest cost possible. To achieve this goal, the authors of [WNJ06] translate attack graphs into weighted monotone propositional formulæ, which are later transformed into disjunctive normal form (DNF) and analyzed further. As a weighted monotone propositional formula is equivalent to an attack tree with a basic assignment of cost, Algorithm 2 could be applied for reducing the formula before the transformation into DNF, thus possibly avoiding the exponential explosion.

4.7 Empirical validation

We have implemented the two methods of evaluation of attributes (evaluation on the set semantics and the repeated bottom-up evaluation) and tested their performance on synthetic trees. The main goal of our experiment was to compare how the two methods perform, depending on the characteristics of the analyzed trees. An excerpt from the obtained results is presented in Table 5. Full description of the experimental setup, as well as all the sources necessary to reproduce the results are available at `https://github.com/wwidel/rbu-tests`.

For a tree T with n nodes and k repeated basic actions of the proponent, we used the two methods for evaluating the *minimal cost for the proponent* attribute. Basic assignments β were constructed under the assumption that the opponent performs all of their actions. Values assigned to the basic actions of the proponent were generated randomly. We have measured the time of the evaluation on the set semantics $\texttt{cost}_{\mathcal{S}}(T, \beta)$ (which includes the time needed for the construction of the set semantics itself) and using the repeated bottom-up evaluation (Algorithm 1). Each time value presented in Table 5 is an average over twenty measurements.

Table 5 is partitioned into three parts. For the trees from the first part, the performance of the two methods is comparable. For the trees presented in the second part, the computation on the set semantics outperforms Algorithm 1, while the opposite is true for the third part of the table.

We would like to point out that the trees from the second part of Table 5 have small set semantics, while having a significant number of repeated basic actions. The trees *tree10* and *tree13* have large set semantics, while having a very low number of repeated basic actions. These results are in line with the established complexity of the two methods of evaluation of attributes.

4.8 Conclusion and future work

The main focus of this chapter was the problem of evaluation of attributes on attack–defense trees containing clones. By determining several elementary properties of the set semantics, we motivated the usage of the evaluation of attributes on the set semantics in the case of attributes whose domains are induced by semirings, and in particular, the ones having non-increasing attribute domains.

With Theorem 1, we established sufficient conditions for the standard bottom-up evaluation of attributes returning meaningful results in attack–defense trees containing clones. An alternative method of evaluation, the repeated bottom-up evaluation, given in Algorithm 1, has been developed for the attributes having non-increasing attribute domains. It serves as the starting point of the tree reduction procedure given in Algorithm 2, which can be used for extracting optimal strategies from attack–defense trees.

Table 5: Running time of the methods for randomly generated trees with n nodes and k repeated basic actions of the proponent.

Parameters					**Time in sec**	
Name of file storing T	n	k	$\lvert\mathcal{S}(T)\rvert$	$\lvert\mathcal{S}(T))\rvert$ bound of Proposition 4	$\mathrm{cost}_{\mathcal{S}}(T,\beta)$	Algorithm 1
tree04	31	7	352	1024	0.01	0.02
tree08	37	9	928	4096	0.04	0.05
tree12	43	11	2436	16384	0.25	0.27
tree20	36	4	832	1024	0.04	< 0.01
tree03	31	4	640	1024	0.03	< 0.01
tree29	41	10	640	1280	0.02	0.12
tree30	43	11	704	1408	0.02	0.28
tree31	45	12	768	1536	0.02	0.54
tree32	47	13	832	1664	0.03	1.17
tree24	50	8	9536	16384	3.42	0.04
tree10	43	2	14336	16384	10.34	< 0.01
tree13	46	0	32768	32768	95.45	< 0.01
tree15	46	6	13824	32768	8.19	< 0.01

Both algorithms can prove useful in problems involving `AND`/`OR` trees or, more generally, monotone Boolean formulæ, as demonstrated in Section 4.6.

There are several interesting directions in which the work presented in this chapter could be developed further. First, it seems worthwhile to try verifying Conjecture 1. Second, since the running time of Algorithm 1 and 2 is exponential in the number of clones of the proponent, one could try to construct approximate variants of the two algorithms. For instance, Algorithm 1 could be parameterized with an upper bound on the number of subsets of the set of optional clones considered in the **for** loop, causing the complexity of the algorithm to depend mostly on the complexity of the bottom-up evaluation of the attribute domain provided as input. It seems that with Algorithm 2 relying on this heuristic variant of Algorithm 1, replacing the equality from line 14 with the "less than or equal to" inequality would be sufficient for obtaining a fast, approximate method for extracting optimal strategies from attack–defense trees.

Chapter 5

Multi-parameter analysis of security

In the previous chapter, we studied the problem of quantitative analysis of security using attributes of attack–defense trees. Classically, attribute domains have been used for formalizing single parameter optimization problems on attack–defense trees, such as determining the minimal cost or maximal probability of achieving the root goal. In this chapter, we tackle the issue of multi-parameter optimization. We demonstrate how multiple attribute domains can be combined into a single one, called *Pareto attribute domain*, with the evaluation of the corresponding attribute providing *Pareto optimal* values of attributes of achieving the root goal. We build upon the results presented in Chapter 4 to identify Pareto attribute domains whose attributes can be evaluated in trees containing clones using the repeated bottom-up evaluation (Algorithm 1).

The structure of this chapter is as follows. In Section 5.1, we recall the notion of Pareto optimality, adapted to the setting that we are mostly interested in. The construction and some properties of Pareto attribute domains are presented in Section 5.2. The applicability of Pareto attribute domains is illustrated in Section 5.3, with both a small case study, and with results of tests conducted on synthetic trees. We conclude the chapter and discuss possible future research directions in Section 5.4.

5.1 Preliminaries

To compare different strategies while taking multiple attributes related to their execution into account, we assign vectors of values to the strategies. Our main focus is on the attribute domains induced by semirings. Therefore, every set D_i considered in the remainder of this chapter is equipped with two binary operations $\oplus_i$ and $\otimes_i$, such that $(D_i, \oplus_i, \otimes_i)$ is a commutative idempotent semiring. Vectors belonging to $D_1 \times \ldots \times D_m$ will be marked in bold, and if $\mathbf{d}$ is a vector, d_i will stand for its ith coordinate. We use $\preceq_i$ to denote the canonical partial order on D_i, defined with $d \preceq_i d'$ if and only if $d \oplus_i d' = d'$, for $d, d' \in D_i$. Intuitively, $d \preceq_i d'$ if and only if d' is preferred over d. To compare the elements of the set $D_1 \times \ldots \times D_m$, we use the following standard partial

ordering[1] induced by the orders $\preceq_i$.

Definition 30 (Dominance)**.** *For* $\mathbf{d}, \mathbf{d}' \in D_1 \times \ldots \times D_m$*, the element* $\mathbf{d}'$ *dominates* $\mathbf{d}$ *(equivalently,* $\mathbf{d}$ *is dominated by* $\mathbf{d}'$*), denoted* $\mathbf{d} \preceq \mathbf{d}'$*, if the inequality* $d_i \preceq_i d'_i$ *holds for every* $i \in \{1, \ldots, m\}$*.*

Example 45. *Consider the* minimal time for the proponent *and the* maximal probability for the proponent *attribute domains (given in Table 1), which are induced by the commutative idempotent semirings* $(\mathbb{N} \cup \{+\infty\}, \min, +)$ *and* $([0,1], \max, \cdot)$*, respectively. To choose strategies optimal w.r.t. both attributes, we consider the set* $(\mathbb{N} \cup \{+\infty\}) \times [0,1]$*. Following Definition 30, a point* (d_1, d_2) *belonging to this set is dominated by a point* (d'_1, d'_2) *if* $\min(d_1, d'_1) = d'_1$ *and* $\max(d_2, d'_2) = d'_2$*. In other words,* $(d_1, d_2) \preceq (d'_1, d'_2)$ *if* $d_1 \geqslant d'_1$ *and* $d_2 \leqslant d'_2$*.*

For example, let $D = \{(125, 0.114), (135, 0.057), (145, 2^{-23})\}$ *be the set of points representing the minimal time and the maximal success probability of the strategies*

$$(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset),$$
$$(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset)$$
and
$$(\{\texttt{phish}, \texttt{uname}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset),$$

respectively, under the basic assignments given by Table 3 and 6. The points $(145, 2^{-131})$ *and* $(135, 0.057)$ *are both dominated by* $(125, 0.114)$*.*

Table 6: Basic assignment of *probability* to the basic actions of the proponent from tree in Figure 2.

Basic action b	$\beta_{\texttt{prob}}(\texttt{b})$	**Basic action** b	$\beta_{\texttt{prob}}(\texttt{b})$
cam	0.8	eav	0.5
force	0.3	card	0.2
cash	0.95	pwd	2^{-48}
phish	0.6	uname	2^{-20}
log&trans	0.95	phone	0.2

If an element of $D_1 \times \ldots \times D_m$ corresponding to the value of a strategy (P, O) is dominated by the value of a strategy (P', O'), e.g., the two strategies are equally likely to succeed, but the cost of execution of (P', O') is smaller, then the proponent has no incentive to execute (P, O). Therefore, the interesting elements of $D_1 \times \ldots \times D_m$ are the ones that are not dominated by others.

[1] In the general case of partially ordered sets (not necessarily commutative idempotent semirings) the definitions are analogous, cf. [GBTO07].

Definition 31 (Pareto point)**.** *An element* $\mathbf{d} \in D \subseteq D_1 \times \ldots \times D_m$ *is called a Pareto point of* D *if it is not dominated by any other element of* D*, i.e., if* $\mathbf{d} \not\preceq \mathbf{d}'$ *holds for every* $\mathbf{d}' \in D, \mathbf{d}' \neq \mathbf{d}$*.*

Definition 32 (Pareto frontier)**.** *The set of all Pareto points of a finite set* $D \subseteq D_1 \times \ldots \times D_m$*, denoted* $\max(D)$[2]*, is called Pareto frontier of* D*.*

Example 46. *Consider again the two domains and the set* D *from Example 45. As already observed, the point* $(125, 0.114)$ *dominates the remaining points of* D*. Thus, the Pareto frontier of* D *is*

$$\max(D) = \{(125, 0.114)\}.$$

Our ultimate goal is to identify values of strategies that are not dominated by values corresponding to the execution of other strategies. In other words, the final result of our analysis will be a set whose every element is a Pareto point.

Definition 33 (Pareto optimal set)**.** *A finite set* $D \subseteq D_1 \times \ldots \times D_m$ *satisfying* $D = \max(D)$ *is called a Pareto optimal set. We use* $P(D_1 \times \ldots \times D_m)$ *to denote the set of all Pareto optimal sets in* $D_1 \times \ldots \times D_m$*.*

The considerations in Example 45 and 46 show that D defined in Example 45 is not a Pareto optimal set.

5.2 Pareto attribute domains

We are now ready to develop a general method for combining attribute domains into a single domain suitable for determining Pareto optimal strategies in attack–defense trees.

For $i \in \{1, \ldots, m\}$, let A_{α_i} be the attribute domain $(D_i, \oplus_i, \otimes_i, \otimes_i, \oplus_i, \otimes_i, \oplus_i)$. Given basic assignments β_{α_i} for the attributes α_i, we create a new assignment, which assigns the singleton $\{(\beta_{\alpha_1}(\mathtt{b}), \ldots, \beta_{\alpha_m}(\mathtt{b}))\}$ to each basic action $\mathtt{b} \in \mathbb{B}$. Note that this singleton is a Pareto optimal set, and it contains the optimal value corresponding to the execution of $\mathtt{b}$. Such singletons will be combined using appropriate operations, eventually resulting in a Pareto optimal set of values corresponding to strategies in an attack–defense tree. We now define these operations.

For $\mathbf{d}, \mathbf{d}' \in D_1 \times \ldots \times D_m$, let

$$\mathbf{d} \otimes \mathbf{d}' := (d_1 \otimes_1 d'_1, \ldots, d_m \otimes_m d'_m), \tag{18}$$

and, with a slight abuse of notation, let

$$D \otimes D' := \{\mathbf{d} \otimes \mathbf{d}' \colon \mathbf{d} \in D, \mathbf{d}' \in D'\}, \tag{19}$$

$$D \mathbin{\hat{\otimes}} D' := \max(D \otimes D'), \tag{20}$$

$$D \mathbin{\hat{\oplus}} D' := \max(D \cup D'), \tag{21}$$

[2]The choice of the $\max(\cdot)$ notation is dictated by the fact that Pareto points are the maximal elements w.r.t. the dominance relation.

for $D, D' \in P(D_1 \times \ldots \times D_m)$.

The intuition behind the above construction is the following. Suppose that two sets D and D' contain Pareto optimal values corresponding to the achievement of two different subgoals by the proponent in a tree with no repeated basic actions. If in order to achieve the root goal of T the proponent has to achieve at least one of the two subgoals, then the set of Pareto optimal values of achieving the root goal is computed as $D \hat{\oplus} D'$: this operation first gathers all the values corresponding to the strategies achieving the root goal in a single set, and then returns the Pareto frontier of this set. Similarly, if the proponent had to achieve both of the aforementioned goals, then the Pareto optimal values of strategies in T would be obtained by computing $D \hat{\otimes} D'$: here the result is the Pareto frontier of the set of all possible values corresponding to simultaneous achievement of the two subgoals.

Given the above construction, the values of Pareto optimal strategies can be obtained using the attribute domain $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$. Throughout the rest of the thesis, we refer to the attribute domains resulting from the above process as *Pareto attribute domains*.

Definition 34 (Pareto attribute domain)**.** *A* Pareto attribute domain *is an algebraic structure of the form* $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$*, for some attribute domains* $A_{\alpha_i} = (D_i, \oplus_i, \otimes_i, \otimes_i, \oplus_i, \otimes_i, \oplus_i)$ *induced by semirings, for* $i \in \{1, \ldots, m\}$*, and with the operations* $\hat{\oplus}$ *and* $\hat{\otimes}$ *defined by* (18)–(21)*. We say that the Pareto attribute domain* $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$ *is* induced by *the attribute domains* A_{α_i}*, for* $i \in \{1, \ldots, m\}$*.*

Pareto attribute domains enjoy the following fundamental properties.

Theorem 5. *A Pareto attribute domain* $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$ *induced by attribute domains* A_{α_i} *induced by semirings, for* $i \in \{1, \ldots, m\}$*, is an attribute domain (in the sense of Definition 18), and* $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes})$ *is a commutative idempotent semiring.*

Furthermore, if the domains A_{α_i}*,* $i \in \{1, \ldots, m\}$*, are non-increasing, then the induced Pareto attribute domain is also non-increasing.*

Before presenting its proof, we briefly discuss the immediate consequences of Theorem 5. The first of them follows from Theorem 1: if there are no repeated basic actions in an attack–defense tree, then the evaluation of a number of attributes having domains induced by semirings can be performed using a single bottom-up procedure. Second, if a tree contains repeated basic actions and the Pareto attribute domain is induced by non-increasing attribute domains, then, by Theorem 2, the repeated bottom-up evaluation given in Algorithm 1 can be applied, and the values of Pareto optimal strategies can still be obtained without the need of constructing the set semantics of the entire tree. Third, note that if a Pareto domain is induced by attribute domains whose multiplicative

operations are idempotent, then the operation $\hat{\otimes}$ is itself idempotent. Therefore, again due to Theorem 1, in such a case the evaluation of a Pareto attribute can be performed using a single bottom-up procedure.

The above discussion is summarized in the following theorem.

Theorem 6. *Let T be an attack–defense tree and let $A_{Par} = (P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$ be a Pareto attribute domain induced by the attribute domains A_{α_i}, for $i \in \{1, \ldots, m\}$. Then*

- *if there are no repeated labels in T, then the equality $Par_B(T, \beta_{Par}) = Par_{\mathcal{S}}(T, \beta_{Par})$ holds for any basic assignment β_{Par},*
- *if the operator $\otimes_i$ is idempotent, for every $i \in \{1, \ldots, m\}$, then the equality $Par_B(T, \beta_{Par}) = Par_{\mathcal{S}}(T, \beta_{Par})$ holds for any basic assignment β_{Par},*
- *if A_{α_i} is a non-increasing attribute domain, for every $i \in \{1, \ldots, m\}$, then for every $O \subseteq \mathbb{B}^{o_T}$ and every basic assignment β_{Par} satisfying*

$$\beta_{Par}(\mathtt{b}) = \mathtt{a}_{\hat{\otimes}} \text{ for } \mathtt{b} \in O,$$
$$\beta_{Par}(\mathtt{b}) = \mathtt{e}_{\hat{\otimes}} \text{ for } \mathtt{b} \in \mathbb{B}^{o_T} \backslash O,$$

the equality $Par_{RB}(T, \beta_{Par}, O) = Par_{\mathcal{S}}(T, \beta_{Par})$ holds.

5.2.1 Proof of Theorem 5

Our proof of Theorem 5 exploits some elementary properties of the dominance relation and of the Pareto frontier, stated in Lemma 6–9. Recall that, for every $i \in \{1, \ldots, m\}$, $(D_i, \oplus_i, \otimes_i)$ is an idempotent commutative semiring and that the dominance relation $\preceq$ in $D_1 \times \ldots \times D_m$ is defined w.r.t. the canonical partial orders $\preceq_i$.

Lemma 6. *Let $\mathbf{d}, \mathbf{d}'$ and $\mathbf{d}''$ be elements of $D_1 \times \ldots \times D_m$. Then,*

1. *if $\mathbf{d}' \preceq \mathbf{d}''$, then $\mathbf{d} \otimes \mathbf{d}' \preceq \mathbf{d} \otimes \mathbf{d}''$,*
2. *if the relation $d \otimes_i d' \preceq_i d'$ holds for every $i \in \{1, \ldots, m\}$ and for every $d, d' \in D_i$, then $\mathbf{d} \otimes \mathbf{d}' \preceq \mathbf{d}'$.*

Proof. For every $i \in \{1, \ldots, m\}$, $(D_i, \oplus_i, \otimes_i)$ is an idempotent commutative semiring. Therefore, for $d, d', d'' \in D_i$ we have that if $d' \preceq_i d''$, then

$$d \otimes_i d'' = d \otimes_i (d' \oplus_i d'') = (d \otimes_i d') \oplus_i (d \otimes_i d''),$$

meaning that $d \otimes_i d' \preceq_i d \otimes_i d''$. Together with definition of the dominance relation, this implies the first statement.

The second statement follows immediately from the definition of $\mathbf{d} \otimes \mathbf{d}'$, defined by (18) on page 109, and the definition of the dominance relation. □

Lemma 7. *If A and B are finite subsets of $D_1 \times \ldots \times D_m$, then*

1. $\max(A \cup B) \subseteq \max(A) \cup \max(B)$,
2. $\max(A \otimes B) \subseteq \max(A) \otimes \max(B)$.

Proof. For a proof of the first of the two statements, let $\mathbf{d} \in \max(A \cup B)$. Since $\mathbf{d}$ is not dominated by any other element of $A \cup B$, it follows that if $\mathbf{d} \in A$, then $\mathbf{d} \in \max(A)$, and if $\mathbf{d} \in B$, then $\mathbf{d} \in \max(B)$. Hence, $\mathbf{d} \in \max(A) \cup \max(B)$.

Now, let $\mathbf{d} = \mathbf{d}_A \otimes \mathbf{d}_B \in \max(A \otimes B)$ for some $\mathbf{d}_A \in A$ and $\mathbf{d}_B \in B$. Towards a contradiction, suppose that $\mathbf{d} \notin \max(A) \otimes \max(B)$. Then, there exist elements $\mathbf{d}'_A \in \max(A)$, $\mathbf{d}'_B \in \max(B)$, such that $\mathbf{d}'_A$ dominates $\mathbf{d}_A$ and $\mathbf{d}'_B$ dominates $\mathbf{d}_B$, with $\mathbf{d}'_A \neq \mathbf{d}_A$ or $\mathbf{d}'_B \neq \mathbf{d}_B$. Since $\mathbf{d} \notin \max(A) \otimes \max(B)$, it follows that $\mathbf{d} \neq \mathbf{d}'_A \otimes \mathbf{d}'_B$. Furthermore, by Lemma 6, it holds that $\mathbf{d} \preceq \mathbf{d}'_A \otimes \mathbf{d}'_B$. This contradicts the choice of $\mathbf{d}$ as a Pareto point in $A \otimes B$. □

Lemma 8. *If A and B are finite subsets of $D_1 \times \ldots \times D_m$, then* $\max(\max(A) \cup B) = \max(A \cup B)$.

Proof. Let $\mathbf{d} \in \max(A \cup B)$. Observe that $\mathbf{d} \in \max(A) \cup B$, by Lemma 7. Furthermore, since $\mathbf{d}$ is not dominated by any of the points in $A \cup B$, it is also not dominated by any of the points in $\max(A) \cup B$. This proves that $\max(\max(A) \cup B) \supseteq \max(A \cup B)$.

For a proof of the inclusion $\max(\max(A) \cup B) \subseteq \max(A \cup B)$, let $\mathbf{d}$ be a Pareto point in $\max(A) \cup B$. Suppose that $\mathbf{d}$ is not a Pareto point in $A \cup B$. Then there exists $\mathbf{d}' \in A \cup B$, $\mathbf{d}' \neq \mathbf{d}$, such that $\mathbf{d} \preceq \mathbf{d}'$. Since $\mathbf{d}$ is not dominated by any element of B, it follows that $\mathbf{d}' \in A$. But then, since $\preceq$ is a transitive relation, every $\mathbf{d}'' \in \max(A)$ that dominates $\mathbf{d}'$ dominates also $\mathbf{d}$. This contradicts the choice of $\mathbf{d}$. □

Lemma 9. *If A and B are finite subsets of $D_1 \times \ldots \times D_m$, then* $\max(\max(A) \otimes B) = \max(A \otimes B)$.

Proof. For a proof of the inclusion $\max(\max(A) \otimes B) \subseteq \max(A \otimes B)$, let $\mathbf{d} \in \max(\max(A) \otimes B)$. Towards a contradiction, suppose that $\mathbf{d}$ is not a Pareto point in $A \otimes B$. This implies that there exist elements $\mathbf{d}_A \in A$ and $\mathbf{d}_B \in B$ such that $\mathbf{d} \preceq \mathbf{d}_A \otimes \mathbf{d}_B$ and $\mathbf{d} \neq \mathbf{d}_A \otimes \mathbf{d}_B$. Let $\mathbf{d}'_A \in \max(A)$ be such that $\mathbf{d}_A \preceq \mathbf{d}'_A$. Then

$$\mathbf{d} \preceq \mathbf{d}_A \otimes \mathbf{d}_B \preceq \mathbf{d}'_A \otimes \mathbf{d}_B,$$

by Lemma 6. Since $\mathbf{d}'_A \otimes \mathbf{d}_B \in \max(A) \otimes B$, this contradicts the choice of $\mathbf{d}$.

Assume now that $\mathbf{d}$ is a Pareto point in $A \otimes B$. Observe that $\mathbf{d} \in \max(A) \otimes B$, by Lemma 7. Since $\mathbf{d}$ is not dominated by any element of $A \otimes B$, it is in particular not dominated by any element of $\max(A) \otimes B$. Therefore, $\mathbf{d}$ is a Pareto point in $\max(A) \otimes B$. □

We are now ready to prove Theorem 5.

Proof. We begin with proving that $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes})$ is a commutative idempotent semiring. Since a binary associative operation can be modeled with an unranked operator, this immediately implies that $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$ is an attribute domain.

For $A \in P(D_1 \times \ldots \times D_m)$, we have

$$A \hat{\oplus} A = \max(A \cup A) = \max(A) = A,$$

i.e., the operation $\hat{\oplus}$ is idempotent. It is easy to verify that both $\hat{\oplus}$ and $\hat{\otimes}$ are commutative and that $\mathtt{a}_{\hat{\otimes}} = \{(\mathtt{a}_{\otimes_1}, \ldots, \mathtt{a}_{\otimes_m})\}$. Since $\mathtt{a}_{\otimes_i} = \mathtt{e}_{\oplus_i}$ for every $i \in \{1, \ldots, m\}$, together with the definitions of canonical partial orders and Definition 30 this implies that $\mathtt{a}_{\hat{\otimes}}$ is dominated by every other element of $D_1 \times \ldots \times D_m$. Therefore, for any $D \in P(D_1 \times \ldots \times D_m)$, we have that $D \hat{\oplus} \mathtt{a}_{\hat{\otimes}} = \max(D \cup \mathtt{a}_{\hat{\otimes}}) = \max(D) = D$. This proves that $\mathtt{e}_{\hat{\oplus}} = \mathtt{a}_{\hat{\otimes}}$.

The associativity of the two operations follows from Lemma 8 and 9. Namely,

$$\begin{aligned}(A \hat{\oplus} B) \hat{\oplus} C &= \max(\max(A \cup B) \cup C) \\ &\overset{\text{Lemma 8}}{=} \max(A \cup B \cup C) \\ &\overset{\text{Lemma 8}}{=} \max(A \cup \max(B \cup C)) \\ &= A \hat{\oplus} (B \hat{\oplus} C)\end{aligned}$$

and

$$\begin{aligned}(A \hat{\otimes} B) \hat{\otimes} C &= \max(\max(A \otimes B) \otimes C) \\ &\overset{\text{Lemma 9}}{=} \max(A \otimes B \otimes C) \\ &\overset{\text{Lemma 9}}{=} \max(A \otimes \max(B \otimes C)) \\ &= A \hat{\otimes} (B \hat{\otimes} C).\end{aligned}$$

We prove that $\hat{\otimes}$ distributes over $\hat{\oplus}$ in a similar way:

$$\begin{aligned}A \hat{\otimes} (B \hat{\oplus} C) &= \max(A \otimes \max(B \cup C)) \\ &\overset{\text{Lemma 9}}{=} \max(A \otimes (B \cup C)) \\ &= \max(A \otimes B \cup A \otimes C) \\ &\overset{\text{Lemma 8}}{=} \max(\max(A \otimes B) \cup \max(A \otimes C)) \\ &= (A \hat{\otimes} B) \hat{\oplus} (A \hat{\otimes} C).\end{aligned}$$

The above reasoning proves that $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes})$ is a commutative idempotent semiring and that $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$ is an attribute domain.

Assume now that the domains $(D_i, \oplus_i, \otimes_i, \otimes_i, \oplus_i, \otimes_i, \oplus_i)$ are non-increasing, for $i \in \{1, \ldots, m\}$. To prove the second statement of the theorem, it remains to prove that for every $A, B \in P(D_1 \times \ldots \times D_m)$ the equality $A \hat{\oplus} (A \hat{\otimes} B) = A$ holds. Let $A, B \in P(D_1 \times \ldots \times D_m)$. Observe that, since the domains $(D_i, \oplus_i, \otimes_i, \otimes_i, \oplus_i, \otimes_i, \oplus_i)$ are

non-increasing, the second item of Lemma 6 implies that $\max(A \cup (A \otimes B)) = \max(A)$. Furthermore, since A is a Pareto optimal set, the equality $\max(A) = A$ holds. Thus,

$$\begin{aligned} A \hat{\oplus}(A \hat{\otimes} B) &= \max(A \cup \max(A \otimes B)) \\ &\overset{\text{Lemma 8}}{=} \max(A \cup (A \otimes B)) = \max(A) = A. \end{aligned}$$

The proof of Theorem 5 is complete. □

5.2.2 Complexity issues

Theorems 1–5, summarized in Theorem 6, provide a general framework for a convenient multi-objective analysis of scenarios modeled with attack–defense trees. Before illustrating the applicability of the framework, we briefly discuss its complexity.

Recall that even in the simplest case of attack trees with a single *minimal cost* attribute domain, the problem of determining a cheapest strategy is known to be NP-hard [BLWC17], and that this difficulty originates from the presence of repeated basic actions of the proponent (as indicated by Theorems 1 and 2). One could therefore hope for the multi-objective optimization to also be easier in trees with no repeated basic actions. Unfortunately, this is not necessarily the case, due to the number of possible Pareto optimal strategies. The following construction illustrates this issue.

Example 47. *Let $m \geqslant 2$ be an even integer and let $T = \mathtt{AND}^{\mathrm{p}}(\mathtt{OR}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{b}_2), \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_3, \mathtt{b}_4), \ldots, \mathtt{OR}^{\mathrm{p}}(\mathtt{b}_{m-1}, \mathtt{b}_m))$. Consider a Pareto domain induced by m* minimal cost for the proponent *attribute domains and a basic assignment that assigns to the action $\mathtt{b}_i$ a vector assuming 1 on the ith coordinate and 0 on each of the remaining $m-1$ coordinates. Then, every pair of the form $(\{\mathtt{b}_{i_1}, \mathtt{b}_{i_2}, \ldots, \mathtt{b}_{i_{m/2}}\}, \emptyset)$, where $i_j \in \{2j-1, 2j\}$, is a Pareto optimal strategy in T, and the value corresponding to such a strategy is unique. Clearly, the number of such strategies is $2^{m/2}$.*

If the number of domains inducing a Pareto domain is small, then the time and space complexities of the methods for evaluation of attributes depend mostly on two factors: the size of the set semantics and the number k of repeated basic actions in the considered tree. In the case when k is big and the number of strategies is small, it is better to use the computation on the set semantics. This is obviously due to the fact that the time complexity of Algorithm 1 is exponential in k. If k is small and the number of strategies in the tree is big, then Algorithm 1 will perform better. This intuition is supported by the experimental results presented in Section 5.3.2. These results provide also some indications towards making the meaning of the words "big" and "small" more precise for particular use cases.

5.3 Empirical validation

In Section 5.3.1, we validate the practicality of Pareto attribute domains with a small case study. Experimental results illustrating the approach's scalability and the differences between the two methods for attributes evaluation are presented in Section 5.3.2.

5.3.1 Case study

We illustrate the applicability of the developed framework with a "what-if" analysis of the scenario modeled with the attack–defense tree T from Figure 2. For this purpose, we use the Pareto attribute domain induced by the domains for *minimal time for the proponent*, *minimal (technical) skill level of the proponent* and *maximal probability for the proponent* attributes (given in Table 1)[3]. In other words, we use the domain

$$\big(P\big((\mathbb{N}\cup\{+\infty\})^2\times[0,1]\big),\hat{\oplus},\hat{\otimes},\hat{\otimes},\hat{\oplus},\hat{\otimes},\hat{\oplus}\big),$$

where $\hat{\otimes}$ and $\hat{\oplus}$ are given by equations (19)–(21) for $\otimes$ defined by

$$\mathbf{d}\otimes\mathbf{d}' := (d_1+d_1', \max(d_2,d_2'), d_3\cdot d_3')$$

for $\mathbf{d},\mathbf{d}'\in(\mathbb{N}\cup\{+\infty\})^2\times[0,1]\big)$.

Table 7: Assignment of *time*, *skill level*, and *probability* to the basic actions of the proponent.

Basic action b	$\beta_{\mathtt{time}}(\mathtt{b})$	$\beta_{\mathtt{skill}}(\mathtt{b})$	$\beta_{\mathtt{prob}}(\mathtt{b})$
cam	60	2	0.8
eav	360	0	0.5
force	10	0	0.3
card	120	0	0.2
cash	5	1	0.95
phish	100	4	0.6
pwd	300	0	2^{-48}
uname	20	0	2^{-20}
log&trans	5	1	0.95
phone	20	0	0.2

We denote the above domain with A_{Par} and let the values assigned to the basic actions of the proponent to be as specified in Table 7, e.g., the value assigned to the action cam is $\{(60,2,0.8)\}$. We consider three scenarios. In the first of them, scenario S_1, the opponent

[3]A more realistic case study, involving a Pareto attribute domain induced by a greater number of domains, is conducted in Chapter 7

executes none of their actions. In scenario S_2, the only action executed by the opponent is `sms`. Finally, the opponent executes all of their actions in scenario S_3. For each of the scenarios the basic assignment from Table 7 is extended according to Remark 1 with the values presented in Table 8. For instance, the value assigned to each of the opponent's actions in scenario S_1 is $\{(0, 0, 1)\}$.

Table 8: Assignment of time, skill level and probability to the basic actions of the opponent in scenarios S_1, S_2, and S_3.

Basic action	S_1	S_2	S_3
`cover`	$(0, 0, 1)$	$(+\infty, +\infty, 0)$	$(+\infty, +\infty, 0)$
`spwd`	$(+\infty, +\infty, 0)$	$(0, 0, 1)$	$(0, 0, 1)$
`sms`	$(0, 0, 1)$	$(0, 0, 1)$	$(+\infty, +\infty, 0)$

Table 9: The Pareto optimal strategies in scenarios S_1, S_2, and S_3.

Scenario	Pareto optimal values	Corresponding strategies
S_1	$(135, 1, 0.057)$	$(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset)$
	$(485, 1, 0.095)$	$(\{\texttt{eav}, \texttt{card}, \texttt{cash}\}, \{\texttt{cover}\})$
	$(105, 4, 0.57)$	$(\{\texttt{phish}, \texttt{log\&trans}\}, \{\texttt{sms}\})$
S_2	$(135, 1, 0.057)$	$(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset)$
	$(485, 1, 0.095))$	$(\{\texttt{eav}, \texttt{card}, \texttt{cash}\}, \{\texttt{cover}\})$
	$(125, 4, 0.114)$	$(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{sms}\})$
S_3	$(135, 1, 0.057)$	$(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset)$
	$(545, 2, 0.076)$	$(\{\texttt{cam}, \texttt{eav}, \texttt{card}, \texttt{cash}\}, \emptyset)$
	$(125, 4, 0.114)$	$(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$

Using the set semantics of the tree from Figure 2 (given in Example 23 on page 42), it is straightforward to compute the values corresponding to the execution of the strategies in the particular scenarios (see Table 10), as well as the Pareto optimal values. We illustrate the computation in a bit more detail for the case of scenario S_1, denoting with β_{Par} the basic assignment for this scenario. Following Definition 24, we have

$$\begin{aligned}
\mathrm{Par}_{\mathcal{S}}(T, \beta_{\mathrm{Par}}) =& \widehat{\oplus}_{(P,O)\in\mathcal{S}(T)}\big(\widehat{\otimes}_{\mathtt{b}\in P\cup O}\beta_{\mathrm{Par}}(\mathtt{b})\big) \\
=& \max\big(\beta_{\mathrm{Par}}(\mathtt{force})\,\hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{card})\,\hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{cash}) \cup \ldots \\
& \cup \beta_{\mathrm{Par}}(\mathtt{uname})\,\hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{pwd})\,\hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{phone}) \\
& \hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{log\&trans})\,\hat{\otimes}\,\beta_{\mathrm{Par}}(\mathtt{spwd})\big) \\
=& \max\big(\{(135,1,0.057)\} \cup \ldots \cup \{(345,1,0.12\cdot 2^{-68})\}\big) \\
=& \max\big(\{(135,1,0.057),\ldots,(345,1,0.12\cdot 2^{-68})\}\big) \\
=& \{(135,1,0.057),(485,1,0.095),(105,4,0.57)\}.
\end{aligned}$$

The strategies corresponding to the Pareto optimal values obtained above are presented in Table 9, along the results of the evaluation of the Pareto domain on the set semantics in the remaining scenarios. One can draw several corollaries from Table 9. As two of the optimal values and the optimal strategies obtained for scenarios S_1 and S_2 are the same, one could conclude that securing transfer dispositions with two-factor authentication using mobile phone text messages (`sms`) does not increase significantly one's resistance against stealing from the account in the scenario modeled by the tree from Figure 2. Furthermore, the strategy $(\{\mathtt{force}, \mathtt{card}, \mathtt{cash}\}, \emptyset)$ consisting of forcing the victim to reveal their PIN, stealing the payment card and withdrawing cash from an ATM, implementation of which requires relatively low amount of time and very low technical skill level, is an optimal strategy in all of the three scenarios. Knowing the values corresponding to strategies that achieve the root goal in particular scenarios, as well as the capabilities of the attacker and constraints on available resources, might help a security expert in making an informed decision on which security measures should be implemented.

5.3.2 Performance tests

To verify that the quantitative analysis of attack–defense trees using Pareto attribute domains is applicable for trees describing even more complex scenarios than the one from Figure 2, we have tested our implementation on a number of automatically generated trees. Full description of the experimental setup, as well as all the sources necessary to reproduce the results are available at `https://github.com/wwidel/pareto-tests`. The main goal of our experiment was to compare how the two methods perform, depending on the characteristics of the analyzed trees. An excerpt from the obtained results is presented in Table 11.

For a tree T with n nodes and k repeated basic actions of the proponent, two Pareto domains were considered. Each of them is induced by m domains for *minimal cost for the proponent*, one domain for *minimal skill of the proponent*, and one domain for *minimal time for the proponent*. Basic assignments β were constructed under the assumption that the opponent performs all of their actions. Values assigned to the basic actions

of the proponent were generated randomly. For the computation of Pareto frontiers, the naive method, where each element of a set is compared with the other elements, coordinate by coordinate, was used. We have measured the time of the computation of the Pareto optimal values using the evaluation on the set semantics $\mathrm{Par}_{\mathcal{S}}(T, \beta)$ (which includes the time needed for the construction of the set semantics itself) and using the repeated bottom-up evaluation (Algorithm 1) applied to Pareto domains. Each time value presented in Table 11 is an average over twenty measurements.

Table 11 is partitioned into three parts. For the trees from the first part, the performance of the two methods is comparable. For the trees presented in the second part, the evaluation on the set semantics outperforms the repeated bottom-up evaluation, while the opposite is true for the third part of the table.

We would like to point out the following facts.

1. The attack trees from the second part of Table 11 have small set semantics, while having a significant number of repeated basic actions.

2. The trees *tree10* and *tree13* have large set semantics, while having a very low number of repeated basic actions.

3. The running times for trees *tree12* and *tree30* differ significantly, while the two trees have the same number of nodes and repeated basic actions, and small set semantics. However, there are more Pareto optimal values under the basic assignments generated for *tree30*. This illustrates the impact of the actual *values assigned to the basic actions*, which translates into different numbers of Pareto optimal values, on the running time.

5.4 Conclusion and future work

The main objective of the work presented in this chapter was to develop an efficient method for multi-parameter optimization of security based on attack–defense trees. The proposed Pareto attribute domains are suitable for this purpose, and can be used with attack–defense trees containing repeated basic actions. As discussed already in Section 3.2.5, Pareto attribute domains are a viable alternative to many of the existing methods developed for tackling the same problem. Our construction shows that the multi-parameter evaluation can be addressed with techniques existing for the single-parameter evaluation. Additionally, Theorem 5 constitutes a general algebraic result that might be of independent interest on its own.

We focused on optimization from the point of view of the proponent only. However, the optimization from the point of view of the opponent, or both actors at the same time is also worth investigating. As stated in Remark 1, the basic assignments that we consider for the opponent are limited to express whether actions are executed or not,

without taking their actual values, e.g., *cost*, *probability*, etc. into account. Should such values be considered, interesting questions arise. For instance, given the assignment of a number of attributes to the basic actions of the proponent, as well as the *cost* of the basic actions of the opponent, which countermeasures should the opponent (having a fixed budget) implement to make the achievement of the root goal as "difficult" as possible, in the sense of Pareto optimality? And if the actions of the opponent are decorated with several attributes, how to determine a Pareto optimal solution to the above problem?

The Pareto domains defined in this chapter are intentionally crafted in a way ensuring that they behave well when induced by non-increasing attribute domains. Nevertheless, other constructions are possible. For instance, by replacing the tuple $(P(D_1 \times \ldots \times D_m), \hat{\oplus}, \hat{\otimes}, \hat{\otimes}, \hat{\oplus}, \hat{\otimes}, \hat{\oplus})$, in Definition 34 with the tuple $(P(D_1 \times \ldots \times D_m), \hat{\otimes}, \hat{\otimes}, \hat{\otimes}, \hat{\otimes}, \hat{\otimes}, \hat{\otimes})$, one obtains a domain similar in spirit to the ones considered in [AN15]. While the former is suitable for attributes whose evaluation on the set semantics does not depend on the non-minimal strategies, the latter could be used when every set of basic actions of the actors is considered to be a possible realization of the security scenario modeled with a tree.

Table 10: Values of the strategies in scenarios S_1, S_2, and S_3.

Strategy	S_1	S_2	S_3
$(\{\texttt{force}, \texttt{card}, \texttt{cash}\}, \emptyset)$	$(135, 1, 0.057)$	$(135, 1, 0.057)$	$(135, 1, 0.057)$
$(\{\texttt{cam}, \texttt{eav}, \texttt{card}, \texttt{cash}\}, \emptyset)$	$(545, 2, 0.076)$	$(545, 2, 0.076)$	$(545, 2, 0.076)$
$(\{\texttt{eav}, \texttt{card}, \texttt{cash}\}, \{\texttt{cover}\})$	$(485, 1, 0.095)$	$(485, 1, 0.095)$	$(+\infty, +\infty, 0)$
$(\{\texttt{phish}, \texttt{log\&trans}\}, \{\texttt{sms}\})$	$(105, 4, 0.57)$	$(+\infty, +\infty, 0)$	$(+\infty, +\infty, 0)$
$(\{\texttt{phish}, \texttt{uname}, \texttt{log\&trans}\}, \{\texttt{sms}\})$	$(125, 4, 0.57 \cdot 2^{-20})$	$(+\infty, +\infty, 0)$	$(+\infty, +\infty, 0)$
$(\{\texttt{phish}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{spwd}, \texttt{sms}\})$	$(405, 4, 0.57 \cdot 2^{-48}))$	$(+\infty, +\infty, 0)$	$(+\infty, +\infty, 0)$
$(\{\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{spwd}, \texttt{sms}\})$	$(325, 1, 0.95 \cdot 2^{-68})$	$(+\infty, +\infty, 0)$	$(+\infty, +\infty, 0)$
$(\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$	$(125, 4, 0.114)$	$(125, 4, 0.114)$	$(125, 4, 0.114)$
$(\{\texttt{phish}, \texttt{uname}, \texttt{phone}, \texttt{log\&trans}\}, \emptyset)$	$(145, 4, 0.114 \cdot 2^{-20}))$	$(145, 4, 114 \cdot 2^{-20})$	$(145, 4, 114 \cdot 2^{-20})$
$(\{\texttt{phish}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{spwd}\})$	$(425, 4, 0.114 \cdot 2^{-48})$	$(425, 4, 0.114 \cdot 2^{-48})$	$(+\infty, +\infty, 0)$
$(\{\texttt{uname}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{spwd}\})$	$(345, 1, 0.12 \cdot 2^{-68})$	$(345, 1, 0.12 \cdot 2^{-68})$	$(+\infty, +\infty, 0)$

Table 11: Running times of the methods for some trees with n nodes and k repeated basic actions of the proponent.

Parameters							**Time in sec**	
Name of file storing T	n	k	$\lvert\mathcal{S}(T)\rvert$	$\lvert\mathcal{S}(T)\rvert$ bound of Proposition 4	m	Number of Pareto optimal values	$\text{Pareto}_{\mathcal{S}}(T,\beta)$	$\text{Pareto}_{RB}(T,\beta,\mathbb{B}^{\text{OT}})$
tree04	31	7	352	1024	1	3	0.02	0.02
					5	20	0.11	0.05
tree08	37	9	928	4096	1	2	0.07	0.09
					5	30	0.69	0.23
tree12	43	11	2436	16384	1	1	0.27	0.4
					5	72	4.97	3.2
tree20	36	4	832	1024	1	2	0.04	0.01
					5	12	0.07	0.01
tree29	41	10	640	1280	1	2	0.03	0.25
					5	304	13.32	65.05
tree30	43	11	704	1408	1	3	0.03	0.67
					5	184	6.52	67.51
tree31	45	12	768	1536	1	2	0.04	1.12
					5	128	2.92	53.58
tree32	47	13	832	1664	1	4	0.05	3.47
					5	378	27.88	827.92
tree03	31	4	640	1024	1	2	0.04	< 0.01
					5	131	2.77	0.14
tree10	43	2	14336	16384	1	3	9.68	< 0.01
					5	658	2178.31	1.7
tree13	46	0	32768	32768	1	5	81.93	< 0.01
					5	2151	> 3600	5.56
tree24	50	8	9536	16384	1	2	2.9	0.07
					5	15	3.36	0.1

Chapter 6

Selection of countermeasures in attack–defense scenarios

As highlighted in Section 3.3, a somewhat generic approach to the problem of selection of countermeasures in attack–defense scenarios is, not surprisingly, mathematical programming, and in particular linear programming and integer programming [Chv83]. The standard input required for formulating security related programming problems includes a set of *attacks* (or *threats*; these are sets or sequences of vulnerabilities or attack steps), together with a set of *mitigations* (or *countermeasures*) and a description of relations between the attacks and the mitigations, as in, e.g., [RDR12, Saw13, ZALT19]. Existing methods for extracting such information from attack–defense trees have limited applications, as they have been developed for specific attack–defense trees of very limited expressive power (see Section 3.3 for details).

The work described in this chapter is aimed at achieving two goals. The first of them is the extraction of the information described in the previous paragraph from attack–defense trees, under no structural restrictions being imposed on trees. The second goal is to exploit the specific form of the extracted information for formulating integer linear programming problems interesting from the security point of view.

In Chapter 4, to solve the problem of evaluation of attributes on attack–defense trees containing clones, we proceeded by studying properties of an existing semantics for attack–defense trees, and then exploited them to develop Algorithms 1 and 2. Here, we take a different approach. We begin with formalizing our intuition regarding the knowledge that we would like to extract from trees. This results in a novel semantics for attack–defense trees, defined in Section 6.2, that we call *defense semantics*. Only then we proceed with the (non-trivial) task of developing a method for the construction of this semantics, in Section 6.2.1. Section 6.3 is devoted to a number of security-related optimization problems, expressed in terms of mathematical programming and relying on the defense semantics. We conclude in Section 6.4.

6.1 Preliminaries

The framework developed in this chapter relies on some structural elements of attack–defense trees and on properties of the *satisfiability* attribute domain. We introduce them in this section.

Due to the different shapes and colors used for representing nodes of the two actors, the first thing noticed when one looks at a graphically depicted attack–defense tree are its structural components, namely, the maximal rooted subdags whose all nodes belong to one of the actors. We call them *homogeneous subdags*.

Definition 35 (Homogeneous subdag)**.** *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree and let* $H = (V_H, A_H)$ *be a rooted DAG such that*

- $V_H \subseteq V$, $A_H = A \cap (V_H \times V_H)$,
- *at least one of the parents of* $\mathrm{root}(H)$ *in* T *belongs to the actor other than* $\mathrm{actor}(\mathrm{root}(H))$ *or else* $\mathrm{root}(H) = \mathrm{root}(T)$,
- $\mathrm{children}_T(v) \subseteq V_H$, *for every* $v \in V_H$,
- $\bar{v} \notin V_H$, *for every* $v \in V_H$.

Moreover, let λ_H, actor_H, *and* ref_H *be restrictions of* λ, actor, *and* ref, *respectively, to* V_H. *If the actor assigned to all the nodes of* V_H *is* p_T *(resp.* o_T*), then the attack–defense tree* $(V_H, A_H, \mathrm{L}, \lambda_H, \mathrm{actor}_H, \mathrm{ref}_H)$ *is called a homogeneous subdag of the proponent (resp. opponent) in* T. *If the only homogeneous subdag of* T *is* T *itself, then* T *is called a homogeneous attack–defense tree*[1].

Example 48. *In the tree* T *from Figure 2, the attacker is the proponent and the defender is the opponent. Each of the nodes of the defender constitutes a homogeneous subdag of the defender in* T. *Each of the nodes labeled* `cam` *and* `phone` *is a homogeneous subdag of the attacker in* T, *and the last homogeneous subdag of the attacker in* T *is the subdag of* T *induced by the remaining nodes of the attacker.*

The next example illustrates the fact that every node can belong to more than one homogeneous subdag of a tree.

Example 49. *There are two homogeneous subdags of the defender in the tree from Figure 5. These are* $\mathtt{AND}^{\mathrm{o}}(\mathtt{d}_1, \mathtt{d}_2, \mathtt{d}_3)$ *and* $\mathtt{OR}^{\mathrm{o}}(\mathtt{d}_3, \mathtt{d}_4)$. *The node labeled* $\mathtt{d}_3$ *belongs to both of them.*

Recall that the for a tree T and a set $B \subseteq \mathbb{B}_T$ the value of $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), B)$ is obtained by evaluating a Boolean function that is positive in the variables corresponding to the basic actions of the proponent, and negative in the remaining variables (cf. Remark 2 and 3). This fact has multiple consequences, some of them intuitively obvious, of which the following will be of use for us.

[1] In such a case, T is in fact an attack tree.

Corollary 3. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree and let* $B \subseteq \mathbb{B}^{\mathrm{s}_T}$*, with* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$*, be a set of actions of one of the actors. If the equality* $\mathrm{achieved}_{\mathrm{T}}(v, B) = 0$ *holds for a node* $v \in V$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, B \backslash \{\mathtt{b}\}) = 0$*, for every* $\mathtt{b} \in \mathbb{B}_T$*.*

Proof. For the proof, we assume that $\mathtt{b} \in B$, since otherwise the statement is obviously true.

Suppose first that $\mathrm{actor}(v) = \mathrm{s}_T$. Then, the value of $\mathrm{achieved}_{\mathrm{T}}(v, \cdot)$ is computed by evaluating a Boolean function that is positive in the variables corresponding to the basic actions of the actor s_T. Together with the equality $\mathrm{achieved}_{\mathrm{T}}(v, B) = 0$ and the fact that the value of $\mathrm{achieved}_{\mathrm{T}}(v, B)$ is computed by substituting the 0 assigned to the variable corresponding to the basic action $\mathtt{b}$ in $\mathrm{achieved}_{\mathrm{T}}(v, B \backslash \{\mathtt{b}\})$ with 1, this implies that $\mathrm{achieved}_{\mathrm{T}}(v, B \backslash \{\mathtt{b}\}) = 0$.

Suppose now that $\mathrm{actor}(v) = \bar{\mathrm{s}}_T$. Observe that Definition 15 and 19 together with the definition of the *satisfiability* attribute domain imply that $\mathrm{achieved}_{\mathrm{T}}(v, \emptyset) = 0$. Together with the fact the the function $\mathrm{achieved}_{\mathrm{T}}(v, \cdot)$ is negative in the variables corresponding to the actions belonging to the set B, this implies that $\mathrm{achieved}_{\mathrm{T}}(v, B \backslash \{\mathtt{b}\}) = 0$. □

Corollary 4. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree,* $B \subseteq \mathbb{B}_T$ *be a set of basic actions of the actors, and* $v \in V$ *be a node with* $\mathrm{actor}(v) = \mathrm{s}_T$*, for* $\mathrm{s} \in \{\mathrm{p}, \mathrm{o}\}$*. Then,*

- *if* $\mathrm{achieved}_{\mathrm{T}}(v, B) = 0$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, B \cup B') = 0$*, for every* $B' \subseteq \mathbb{B}^{\bar{\mathrm{s}}_T}$*, and*
- *if* $\mathrm{achieved}_{\mathrm{T}}(v, B) = 1$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, B \cup B') = 1$*, for every* $B' \subseteq \mathbb{B}^{\mathrm{s}_T}$*.*

Of special usefulness for us will be the contraposition of the first of the two statements given in Corollary 4.

Corollary 5. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree, let* $v \in V$ *be a node satisfying* $\mathrm{actor}(v) = \mathrm{p}_T$ *and let* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *be a set of basic actions of the proponent. If there is a set* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ *of basic actions of the opponent such that* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, P) = 1$*.*

Corollary 4 implies also the following.

Corollary 6. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree, and let* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *and* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ *be sets of basic actions of the actors. If the equalities* $\mathrm{achieved}_{\mathrm{T}}(v, P) = 0$ *and* $\mathrm{achieved}_{\mathrm{T}}(v, O) = 0$ *hold for a node* $v \in V$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$*.*

We rely on Corollary 6 to prove the intuitively obvious statement: if a root goal of an attack–defense tree is achieved by a set of basic actions and an action from this set does not contribute to the goal being achieved, then the goal is still achieved after the action is removed from the set.

Lemma 10. *Let* $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ *be an attack–defense tree, and let* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *and* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ *be sets of basic actions of the actors. If* $v \in V$ *is a node such that*

- $\mathrm{ref}(v) = \mathtt{N}$,
- $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$, *and*
- *on every path from* v *to* $\mathrm{root}(T)$ *there is a node* v' *satisfying* $\mathrm{achieved}_{\mathrm{T}}(v', P) = 0$ *and* $\mathrm{achieved}_{\mathrm{T}}(v', O) = 0$,

then $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \backslash \{\lambda(v)\}) = 1$.

Proof. Let v' be one of the nodes satisfying the last condition of the lemma. Corollary 6 implies that $\mathrm{achieved}_{\mathrm{T}}(v', P \cup O) = 0$. Therefore, when the value of $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O)$ is computed using the bottom-up procedure, the value propagated up to the root from v' is zero. Furthermore, it follows from Corollary 3 that $\mathrm{achieved}_{\mathrm{T}}(v', O \backslash \{\lambda(v)\}) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v', P \backslash \{\lambda(v)\}) = 0$. Thus, by Corollary 6, $\mathrm{achieved}_{\mathrm{T}}(v', P \cup O \backslash \{\lambda(v)\}) = 0$, i.e., the value propagated from v' remains unchanged after the removal of the basic action $\lambda(v)$ from $P \cup O$. Hence,

$$\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \backslash \{\lambda(v)\}) = \mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1.$$

□

6.2 Defense semantics

Our ultimate goal is to extract possible behaviors of rational actors from an attack–defense tree modeling a security scenario, and to exploit this information for optimal selection of countermeasures to be implemented by the opponent. Similarly as in the previous chapters, we will express actors' behavior in terms of sets of their basic actions. While some works consider every subset of basic actions of an actor to model a possible realization of the scenario (e.g., [AN15] or [GHL+16]) such an approach is not only computationally ineffective, but also unnecessary, in the sense that among all the subsets there are inefficient ones that do not correspond to a reasonable behavior. Note that the second condition of Definition 35 implies that the goal of the root node of a homogeneous subdag either counters some goal of the other actor, or else achieving it means success for the proponent. Therefore, in order to succeed, the actors need to achieve the root goals of (some of) their corresponding homogeneous subdags; if a set of actions achieves none of the root goals of the homogeneous subdags, its execution has no impact on the realization of the modeled scenario. Therefore, as the building blocks for our formalization of the behavior of rational actors we use minimal sets of actions that achieve root goals of homogeneous subdags. We call them *proponent's* and *opponent's vectors*.

Definition 36 (Proponent's/opponent's vector)**.** *Let T be an attack–defense tree and let H be a homogenous subdag of the proponent (opponent) in T. A minimal, w.r.t. the inclusion, set of basic actions of the proponent (resp. opponent) achieving the root goal of H is called a* proponent's vector *(resp. an* opponent's vector*)* in H.

Example 50. *Let T' be the subdag of the tree T from Figure 2 induced by the nodes bearing labels from the set $\mathbb{B}^{\mathrm{p}_T} \setminus \{\texttt{cam}, \texttt{phone}\}$. The proponent's vectors in this homogeneous subdag of the proponent in T are*

$$\{\texttt{force}, \texttt{card}, \texttt{cash}\},$$
$$\{\texttt{eav}, \texttt{card}, \texttt{cash}\},$$
$$\{\texttt{phish}, \texttt{log\&trans}\},$$
$$\{\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}\}.$$

We assume that in order to counter the proponent in the best way possible, the opponent might be interested in executing a number of opponent's vectors in a single homogeneous subdag of an attack–defense tree. On the other hand, given a specific behavior of the opponent, we assume that a rational proponent executes only those actions that are *necessary* for achieving the root goal. Thus, we try to capture the behavior of rational actors with the following notion of strategies of the actors.

Definition 37 (Proponent's/opponent's strategy)**.** *Let T be an attack–defense tree.*

- *A set $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ is called an opponent's strategy in T if it is a union of any number of opponent's vectors from some of the homogeneous subdags of T. Note that the empty set is a possible opponent's strategy.*
- *A set $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ is called a proponent's strategy in T if there exists an opponent's strategy O in T for which P is a minimal set satisfying* $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$. *Such a set O is called a* witness *for the proponent's strategy P.*

Note that every proponent's strategy can be witnessed by many opponent's strategies, and that each of the opponent's strategies can be a witness for a number of proponent's strategies.

Example 51. *Consider again the tree T from Figure 2. The opponent's strategies in T are the elements of the set $2^{\mathbb{B}^{\mathrm{o}_T}}$, i.e., the sets*

$$\emptyset,$$
$$\{\texttt{cover}\}, \{\texttt{spwd}\}, \{\texttt{sms}\},$$
$$\{\texttt{cover}, \texttt{spwd}\}, \{\texttt{cover}, \texttt{sms}\}, \{\texttt{sms}, \texttt{spwd}\}, \text{ and}$$
$$\{\texttt{cover}, \texttt{spwd}, \texttt{sms}\}.$$

The proponent's vectors listed in Example 50 are the proponent's strategies witnessed by the empty opponent's strategy $\emptyset$. Intuitively, this means that should the defender perform none of their actions in the scenario modeled with T, the reasonable attacker would achieve the root goal by executing any of the four vectors. The remaining proponent's

strategies in T and their (in this case, unique) witnesses are

$$\{\texttt{cam}, \texttt{eav}, \texttt{card}, \texttt{cash}\}, \text{ witnessed by } \{\texttt{cover}\},$$
$$\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}, \text{ witnessed by } \{\texttt{sms}\},$$
$$\{\texttt{uname}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \text{ witnessed by } \{\texttt{sms}\}.$$

Let P be a proponent's strategy and O be an opponent's strategy in T. We say that O *counters* P, if $\text{achieved}_{\text{T}}(\text{root}(T), P \cup O) = 0$; otherwise P counters O. With the actors' strategies defined by Definition 37, our objective of determining possible behavior of a rational proponent and ways of countering it is accomplished with the notion of *defense semantics*[2].

Definition 38 (Defense semantics)**.** *The defense semantics of an attack–defense tree T, denoted $\mathcal{D}(T)$, is the set of all pairs (P, O), where P is a proponent's strategy in T and O is a minimal (w.r.t. the inclusion) opponent's strategy in T that counters P.*

We would like to stress that the proponent's strategies in an attack–defense tree that cannot be countered *do not* appear in its defense semantics. The proponent's strategies in T that *do* appear in the defense semantics of T, i.e., those that can be countered by an opponent's strategy in T, are called *counterable*. The defense semantics of our running attack–defense tree is given in the following example.

Example 52. *Recall the strategies of the actors in the tree T from Figure 2 given in Example 51. The defense semantics of T is*

$$\begin{aligned}\mathcal{D}(T) = \{&(\{\texttt{eav}, \texttt{card}, \texttt{cash}\}, \{\texttt{cover}\}),\\ &(\{\texttt{phish}, \texttt{log\&trans}\}, \{\texttt{sms}\}),\\ &(\{\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{spwd}\})\\ &(\{\texttt{uname}, \texttt{pwd}, \texttt{log\&trans}\}, \{\texttt{sms}\})\\ &(\{\texttt{uname}, \texttt{pwd}, \texttt{phone}, \texttt{log\&trans}\}, \{\texttt{spwd}\})\}.\end{aligned}$$

The strategies that are not counterable in T are

$$\{\texttt{force}, \texttt{card}, \texttt{cash}\},$$
$$\{\texttt{eav}, \texttt{cam}, \texttt{card}, \texttt{cash}\}, \text{ and}$$
$$\{\texttt{phish}, \texttt{phone}, \texttt{log\&trans}\}.$$

While the concept of the defense semantics is intuitively simple and self-explanatory, constructing this semantics is a complex task. We proceed with describing our method for its construction.

[2]We decided to keep the name under which this semantics was initially introduced in [KW17], as the name seems reasonable no matter which of the actors is the opponent. The semantics aims at helping the opponent to defend against the proponent, regardless of whether the proponent is the attacker or the defender.

6.2.1 Construction of the defense semantics

To construct the defense semantics of an attack–defense tree T, one could consider the following naive approach, which we are going to build upon.

1. Create all the opponent's strategies in T.

2. For every opponent's strategy, determine the proponent's strategies witnessed by it.

3. For every proponent's strategy, identify the minimal opponent's strategies countering it.

The first of the three steps is already very expensive, since every subset of basic actions of the opponent might constitute an opponent's strategy, as illustrated in Example 51. We reduce this step's complexity by creating (if possible) only a subset of the set of all possible opponent's strategies in T, while ensuring that every proponent's strategy is witnessed by at least one element of this subset. Then, we proceed with the remaining two steps. The construction of the defense semantics is summarized in Algorithm 3. The rest of this section is devoted to proving its correctness and completeness.

Algorithm 3 Defense semantics for attack–defense trees

Input: Attack–defense tree T
Output: Defense semantics $\mathcal{D}(T)$ of T

$\mathcal{O} \leftarrow \texttt{SuffWit}_B(T, \beta, \mathrm{root}(T)) \cup \{\emptyset\}$
$\mathcal{P} \leftarrow \emptyset$
for $O \in \mathcal{O}$ **do**
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{P\colon P \text{ is a minimal set in } \texttt{CounterOpp}_B(T, \beta^O, \mathrm{root}(T))\}$
end for
$\mathcal{D}(T) \leftarrow \emptyset$
for $P \in \mathcal{P}$ **do**
    $\mathcal{D}(T) \leftarrow \mathcal{D}(T) \cup \{(P, O)\colon O \text{ is a minimal set in } \texttt{CounterPro}_B(T, \beta^P, \mathrm{root}(T))\}$
end for
return $\mathcal{D}(T)$

We start by introducing four operations on sets of sets that we use to define attribute

domains employed by Algorithm 3. For n sets $\mathcal{A}_1, \ldots, \mathcal{A}_n$ of sets, let

$$\boxtimes_{i=1}^{n} \mathcal{A}_i := \{\bigcup_{i=1}^{n} A_i \mid A_i \in \mathcal{A}_i\}, \tag{22}$$

$$\boxplus_{i=1}^{n} \mathcal{A}_i := \bigcup_{I \subseteq \{1,\ldots,n\}} \boxtimes_{i \in I} \mathcal{A}_i, \tag{23}$$

$$\mathcal{A}_1 \boxslash \mathcal{A}_2 := \begin{cases} \{\emptyset\}, \text{ if } \mathcal{A}_1 = \{\emptyset\} \text{ or } \mathcal{A}_2 = \{\emptyset\}, \\ \mathcal{A}_1 \cup \mathcal{A}_2, \text{ otherwise,} \end{cases} \tag{24}$$

$$\mathcal{A}_1 \boxminus \mathcal{A}_2 := \mathcal{A}_1 \cup (\mathcal{A}_1 \boxtimes \mathcal{A}_2). \tag{25}$$

To construct a set of witnesses sufficient for determining all proponent's strategies, we use the *sufficient witnesses* attribute, abbreviated as `SuffWit`, formalized with the attribute domain $A_{\texttt{SuffWit}} := (2^{2^{\mathbb{B}}}, \boxplus, \boxplus, \boxplus, \boxtimes, \boxplus, \boxminus)$. In Proposition 6, we give an elementary property of the bottom-up evaluation of the `SuffWit` attribute under a specific basic assignment.

Proposition 6. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree and let β be the basic assignment for the* `SuffWit` *attribute defined as*

$$\beta(\mathtt{b}) := \begin{cases} \emptyset, & \textit{if } \mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}, \\ \{\{\mathtt{b}\}\}, & \textit{otherwise.} \end{cases} \tag{26}$$

If $O \in \texttt{SuffWit}_B(T, \beta)$, then O is an opponent's strategy in T.

Proof. We shall prove that, for every $v \in V$, every element of $O \in \texttt{SuffWit}_B(T, \beta, v)$ is a union of opponent's vectors from some of the homogeneous subdags of $T(v)$. The validity of this statement for $v = \mathrm{root}(T)$ completes the proof of Proposition 6.

The proof is by induction on the structure of $T(v)$. For the base case, let v be a non-refined node such that $\bar{v}$ does not exist. Then, the statement is obviously true by the definition of the basic assignment β.

If v is refined or $\bar{v}$ exists, then, since every element of $\texttt{SuffWit}_B(T, \beta, v)$ is a union of some of the sets belonging to

$$\bigcup_{v' \in \mathrm{children}_T(v) \cup \{\bar{v}\}} \texttt{SuffWit}_B(T, \beta, v'),$$

by formulæ (22), (23) and (25), the statement follows from the induction hypothesis. □

The above proof provides some insight into our motivation for the choice of most of the operations of the `SuffWit` attribute domain: they are defined in a way that ensures that the result of the bottom-up evaluation under the basic assignment given by (26) consists of opponent's strategies. There is an additional motivation behind the choice of the $\boxminus$ operation that we will comment on later in this chapter. Nevertheless, being

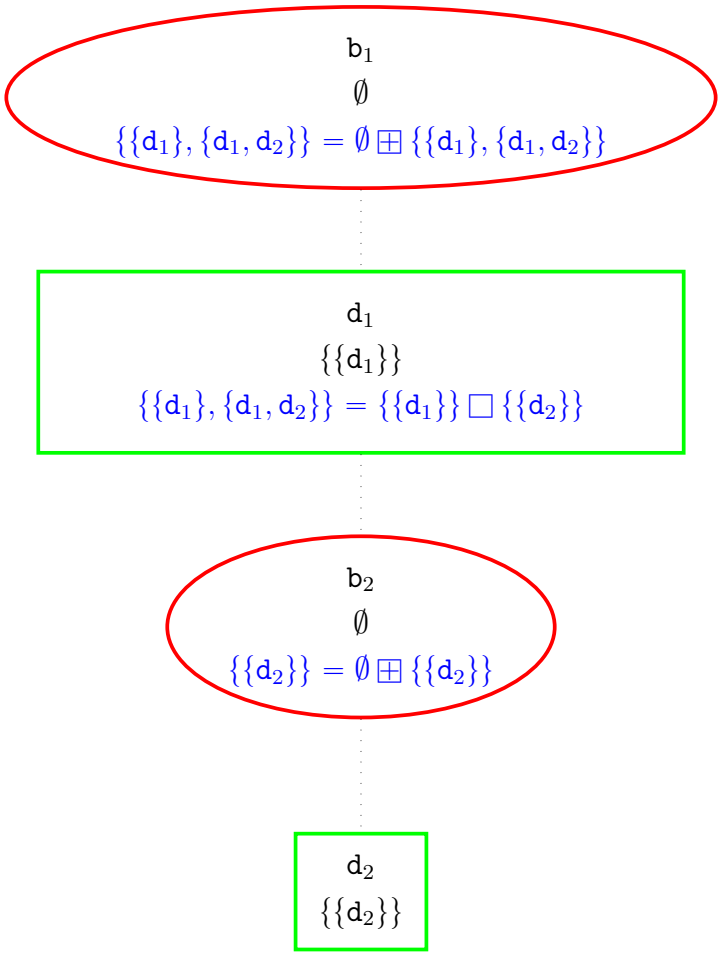


Figure 18: Bottom-up evaluation of the `SuffWit` attribute on the attack–defense tree $\mathtt{C}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{C}^{\mathrm{o}}(\mathtt{d}_1, \mathtt{C}^{\mathrm{p}}(\mathtt{b}_2, \mathtt{d}_2)))$. Values assigned to the basic actions are given in black, values computed at the intermediate nodes — in dark blue.

aware that the definition of the attribute domain $A_{\mathtt{SuffWit}}$ is somewhat non-intuitive, we illustrate its usage with three examples. Throughout the rest of the chapter, whenever we say "bottom-up evaluation of the `SuffWit` attribute", we mean its bottom-up evaluation under the basic assignment given by (26).

Example 53. *Consider the tree* $T = \mathtt{C}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{C}^{\mathrm{o}}(\mathtt{d}_1, \mathtt{C}^{\mathrm{p}}(\mathtt{b}_2, \mathtt{d}_2)))$. *The bottom-up evaluation of the* `SuffWit` *attribute in* T *is depicted in Figure 18. It is easy to verify that the opponent's strategy* $\{\mathtt{d}_1\}$ *is the unique minimal witness for the proponent's strategy* $\{\mathtt{b}_1, \mathtt{b}_2\}$. *The set* $\{\mathtt{d}_1, \mathtt{d}_2\}$ *is an opponent's strategy in* T, *but it is not a witness for any of the proponent's strategies.*

Should a node of the attacker labeled $\mathtt{b}_3$ *be attached as a countermeasure to the node labeled* $\mathtt{d}_2$, *the set obtained with the bottom-up evaluation of* `SuffWit` *in the resulting tree would be the same as in* T. *In this case, however, the opponent's strategy* $\{\mathtt{d}_1, \mathtt{d}_2\}$ *would be the unique minimal witness for the proponent's strategy* $\{\mathtt{b}_1, \mathtt{b}_2, \mathtt{b}_3\}$.

There are attack–defense trees for which the result of the bottom-up evaluation of the `SuffWit` attribute consists of exactly the non-empty witnesses necessary for determining

all proponent's strategies. As illustrated by Example 53, this is the case, for instance, for an attack–defense tree being a path of alternating non-refined nodes of the proponent and the opponent, with the first node on the path belonging to the proponent. We discuss these trees further in the next example.

Example 54. *Let* $T = \mathtt{C}^{\mathrm{p}}(\mathtt{b}_1, \mathtt{C}^{\mathrm{o}}(\mathtt{d}_1, \mathtt{C}^{\mathrm{p}}(\mathtt{b}_2, \mathtt{C}^{\mathrm{o}}(\mathtt{d}_2, \ldots \mathtt{C}^{\mathrm{o}}(\mathtt{d}_n, \mathtt{b}_{n+1}) \ldots))))$ *be an attack–defense tree being a path of alternating non-refined nodes of the proponent and the opponent, with the first node on the path belonging to the proponent, and with* n *nodes of the opponent. The total number of non-empty opponent's strategies in* T *is* $2^n - 1$*, whereas there are only* $n - 1$ *strategies in the result of the bottom-up evaluation of* `SuffWit` *on* T*. Furthermore, each of them is a unique witness for one of the proponent's strategies: the opponent's strategy* $\{\mathtt{d}_1, \ldots, \mathtt{d}_i\}$*, with* $i \in \{1, \ldots, n\}$*, is the unique witness for the proponent's strategy* $\{\mathtt{b}_1, \ldots, \mathtt{b}_{i+1}\}$*.*

Observe the following: if O *is an opponent's strategy belonging to the result of the bottom-up evaluation of* `SuffWit` *on* T *and* $\mathtt{d}_i \in O$*, with* $i \in \{1, \ldots, n\}$*, then* $\mathtt{d}_j \in O$ *for every* $j \in \{1, \ldots, i-1\}$*. Informally speaking, there are no "gaps" in the obtained opponent's strategies. This is intentional: should the above condition be not satisfied by an opponent's strategy* O*, say,* $O = \{\mathtt{d}_1, \ldots, \mathtt{d}_i, \mathtt{d}_{i+k}\}$*, with* $i, i+k \in \{1, \ldots, n\}$*,* $k > 1$*, then* O *is a witness for the same proponent's strategies as* $\{\mathtt{d}_1, \ldots, \mathtt{d}_i\}$*. This example motivates our choice of the operation* $\square$ *as the one to be performed in the bottom-up procedure when traversing countermeasures against goals of the opponent.*

Example 53 illustrates also the fact that, in general, there might be opponent's strategies in the result of the bottom-up evaluation of the `SuffWit` attribute that do not witness any proponent's strategy, or that witness the same proponent's strategies as other elements of the set. This is also the case in our running example.

Example 55. *Let* T *be an attack–defense tree from Figure 2. Recall that the operation performed at the nodes of the proponent during the bottom-up evaluation of the* `SuffWit` *attribute is* $\boxplus$*, defined by* (23)*, and the one performed when traversing countermeasures against goals of the opponent is* $\square$*, defined by* (25)*. Observe that for* $\mathcal{A}$ *being a set of sets the equalities*

$$\mathcal{A} \boxplus \emptyset = \mathcal{A},$$
$$\mathcal{A} \boxtimes \emptyset = \emptyset \text{ and}$$
$$\mathcal{A} \square \emptyset = \mathcal{A}$$

hold. Recalling the basic assignment given by (26)*, it is thus easy to see that the results of the bottom-up evaluation of the* `SuffWit` *attribute at the nodes labeled* via ATM*,* via online banking *and* steal from account *are* $\{\{\mathtt{cam}\}\}$*,* $\{\{\mathtt{spwd}\}, \{\mathtt{sms}\}, \{\mathtt{spwd}, \mathtt{sms}\}\}$ *and* $2^{\mathbb{B}^{\mathrm{o}}T} \setminus \emptyset$*, respectively.*

In Proposition 7, we shall prove that the result of the bottom-up evaluation of the `SuffWit` attribute on T contains at least one witness for each of the proponent's strategies

in T. Our proof of Proposition 7 relies on the following property of the attribute domain $A_{\texttt{SuffWit}}$.

Lemma 11. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree, let $v \in V$ and let β be the basic assignment of the* `SuffWit` *attribute defined by* (26)*. Let $T' = (V', A')$ be a rooted subdag of T such that*

- $\mathrm{root}(T') = v$,
- *if $v' \in V'$ and $\mathrm{ref}(v') =$* `AND`*, then $\mathrm{children}_T(v') \subseteq V'$,*
- *if $v' \in V'$ and $\mathrm{ref}(v') =$* `OR`*, then the intersection $\mathrm{children}_T(v') \cap V'$ is not empty,*
- $A' = A \cap (V' \times V')$.

Let

$$\mathbb{B}^{\mathrm{o}_T}_{T'} := \{\lambda(v') \colon v' \in V', \mathrm{actor}(v') = \mathrm{o}_T, \mathrm{ref}(v') = \texttt{N}\} \tag{27}$$

be the set of all basic actions of the opponent in T that appear in T'. If the set $\mathbb{B}^{\mathrm{o}_T}_{T'}$ is non-empty, then it belongs to $\texttt{SuffWit}_B(T, \beta, v)$.

Proof. The proof is by induction on the structure of T'. We consider three cases.

Case 1. The node v is not refined and $\bar{v} \notin V'$.

Since the set $\mathbb{B}^{\mathrm{o}_T}_{T'}$ is not empty, it follows that $\mathrm{actor}(v) = \mathrm{o}_T$, $\mathbb{B}^{\mathrm{o}_T}_{T'} = \{\lambda(v)\}$, and $\texttt{SuffWit}_B(T, \beta, v) = \{\{\lambda(v)\}\}$. Thus, the claim holds.

Case 2. The node v is not refined and $\bar{v} \in V'$.

If $\mathrm{actor}(v) = \mathrm{p}_T$, then $\mathbb{B}^{\mathrm{o}_T}_{T'} = \mathbb{B}^{\mathrm{o}_T}_{T'(\bar{v})}$. Since $\mathbb{B}^{\mathrm{o}_T}_{T'} \neq \emptyset$, it follows that $\mathbb{B}^{\mathrm{o}_T}_{T'(\bar{v})} \neq \emptyset$, and so the subdag $T'(\bar{v})$ of T' rooted at $\bar{v}$ satisfies all of the assumptions of the lemma. Thus, by the induction hypothesis, we have $\mathbb{B}^{\mathrm{o}_T}_{T'(\bar{v})} \in \texttt{SuffWit}_B(T, \beta, \bar{v})$. The definition of the attribute domain for `SuffWit` and the operation $\boxplus$ defined by formula (23) imply that $\texttt{SuffWit}_B(T, \beta, \bar{v}) \subseteq \texttt{SuffWit}_B(T, \beta, v)$. Hence, $\mathbb{B}^{\mathrm{o}_T}_{T'} \in \texttt{SuffWit}_B(T, \beta, v)$.

If $\mathrm{actor}(v) = \mathrm{o}_T$, then $\mathbb{B}^{\mathrm{o}_T}_{T'} = \mathbb{B}^{\mathrm{o}_T}_{T'(\bar{v})} \cup \{\lambda(v)\}$. From the definition of the attribute domain for `SuffWit`, the basic assignment β, and the operation $\square$ defined by formula (25), it follows that both sets $\{\{\lambda(v)\}\}$ and $\texttt{SuffWit}_B(T, \beta, \bar{v}) \boxtimes \{\{\lambda(v)\}\}$ are subsets of $\texttt{SuffWit}_B(T, \beta, v)$. Therefore, regardless of whether the set $\mathbb{B}^{\mathrm{o}_T}_{T'(\bar{v})}$ is empty or not, we have $\mathbb{B}^{\mathrm{o}_T}_{T'} \in \texttt{SuffWit}_B(T, \beta, v)$, as required.

Case 3. The node v is refined.

Let k be the size of the (possibly empty) set $\{v' \in \mathrm{children}_T(v) \cap V' \mid \mathbb{B}^{\mathrm{o}_T}_{T'(v')} \neq \emptyset\}$. If $k \neq 0$, we use $v_1, \ldots, v_k$ to denote the elements of this set. Depending on whether or not $\bar{v} \in V'$, we have

$$\mathbb{B}^{\mathrm{o}_T}_{T'} = \bigcup_{i=1}^{k} \mathbb{B}^{\mathrm{o}_T}_{T'(v_i)}$$

or

$$\mathbb{B}^{\mathrm{o}T}_{T'} = \bigcup_{i=1}^{k} \mathbb{B}^{\mathrm{o}T}_{T'(v_i)} \cup \mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})}.$$

Note that, since $\mathbb{B}^{\mathrm{o}T}_{T'} \neq \emptyset$, this implies that $k \geqslant 1$ or the set $\mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})}$ is not empty. Observe also that, by the induction hypothesis, $\mathbb{B}^{\mathrm{o}T}_{T'(v_i)} \in \mathtt{SuffWit}_B(T, \beta, v_i)$, for every $i \in \{1, \ldots, k\}$. If in addition $\bar{v} \in V'$ and $\mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})} \neq \emptyset$, then also $\mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})} \in \mathtt{SuffWit}_B(T, \beta, \bar{v})$.

We distinguish two subcases, depending on the value of k.

Case 3.1 $k = 0$

The assumption of this case implies that $\mathrm{actor}(v) = \mathrm{p}_T$, $\bar{v} \in V'$, and $\mathbb{B}^{\mathrm{o}T}_{T'} = \mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})} \neq \emptyset$. Similarly to Case 2, now it follows from the definition of the attribute domain for $\mathtt{SuffWit}$ and the operation defined by formula (23) that $\mathtt{SuffWit}_B(T, \beta, \bar{v}) \subseteq \mathtt{SuffWit}_B(T, \beta, v)$. Hence, $\mathbb{B}^{\mathrm{o}T}_{T'} \in \mathtt{SuffWit}_B(T, \beta, v)$.

Case 3.2 $k > 0$

In this case, the definitions of the attribute domain for $\mathtt{SuffWit}$ and the operations given by formulæ (22), (25), and (23) imply that

$$\boxed{\bigtimes_{i=1}^{k}} \mathtt{SuffWit}_B(T, \beta, v_i)$$

as well as

$$\mathtt{SuffWit}_B(T, \beta, \bar{v}) \boxtimes \boxed{\bigtimes_{i=1}^{k}} \mathtt{SuffWit}_B(T, \beta, v_i),$$

if $\bar{v}$ exists, are subsets of $\mathtt{SuffWit}_B(T, \beta, v)$. Thus, $\bigcup_{i=1}^{k} \mathbb{B}^{\mathrm{o}T}_{T'(v_i)} \in \mathtt{SuffWit}_B(T, \beta, v)$, and if $\bar{v}$ exists and the set $\mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})}$ is not empty, then also $\bigcup_{i=1}^{k} \mathbb{B}^{\mathrm{o}T}_{T'(v_i)} \cup \mathbb{B}^{\mathrm{o}T}_{T'(\bar{v})} \in \mathtt{SuffWit}_B(T, \beta, v)$. This completes the proof of Lemma 11. □

We are now ready to state and prove Proposition 7.

Proposition 7. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree, let P be a proponent's strategy in T and let β be the basic assignment defined by* (26)*. If O is a minimal non-empty witness for P in T, then $O \in \mathtt{SuffWit}_B(T, \beta)$.*

Proof. We begin with constructing an appropriate subdag of T to which we then apply Lemma 11. Let

$$\begin{aligned} V' &:= \{v \in V\colon \mathrm{achieved}_{\mathrm{T}}(v, P) = 1 \text{ or } \mathrm{achieved}_{\mathrm{T}}(v, O) = 1\},\\ V'' &:= \{v \in V'\colon \text{ there are nodes } v_1, v_2, \ldots, v_m \in V, \text{ such that } v_1 v_2 \ldots v_m \text{ is a path in } T,\\ &\qquad v_1 = v, v_m = \mathrm{root}(T), \text{ and } v_i \in V', \text{ for } i \in \{1, \ldots, m\}\}. \end{aligned}$$

Let T'' be the subdag of T induced by V''. Observe that, since $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$, it follows from Corollary 5 that the root of T belongs to the set V'. Together

with the definition of V'', this implies that the subdag T'' is connected and rooted at[3] root(T). Furthermore, the choice of V' and V'' implies that T'' satisfies the assumptions of Lemma 11 (as the subdag T'). Therefore, if the set $\mathbb{B}^{\mathrm{o}_T}_{T''}$ defined by (27) is not empty, then $\mathbb{B}^{\mathrm{o}_T}_{T''} \in \mathtt{SuffWit}_B(T, \beta)$. To complete the proof we shall thus prove that $\mathbb{B}^{\mathrm{o}_T}_{T''} = O$.

The inclusion $\mathbb{B}^{\mathrm{o}_T}_{T''} \subseteq O$ follows immediately from the choice of V''. To prove that the two sets are in fact equal, suppose that there is a node $v \in V$ with $\lambda(v) \in O$ which does not belong to V''. Then, since $v \in V'$, it follows that on every path from v to root(T) there is a node other than v, such that $\mathrm{achieved}_{\mathrm{T}}(v', P) = 0$ and $\mathrm{achieved}_{\mathrm{T}}(v', O) = 0$. Since O is a witness for P, we have $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O) = 1$. Therefore, Lemma 10 implies that $\mathrm{achieved}_{\mathrm{T}}(\mathrm{root}(T), P \cup O \backslash \{\lambda(v)\}) = 1$. This contradicts the choice of O as the minimal witness for P. Hence, $\mathbb{B}^{\mathrm{o}_T}_{T''} = O$, completing the proof. □

Proposition 6 and 7 imply that the bottom-up evaluation of the `SuffWit` attribute is a suitable choice for the first step in the process of construction of the defense semantics.

Corollary 7. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree and let β be the basic assignment defined by* (26)*. The set*

$$\begin{aligned}\{P \subseteq \mathbb{B}^{\mathrm{p}_T} \colon\ & \textit{there is } O \in \mathtt{SuffWit}_B(T, \beta, \mathrm{root}(T)) \\ & \textit{such that } P \textit{ is a minimal set countering } O \\ & \textit{or } P \textit{ is a minimal set countering } \emptyset\}\end{aligned}$$

consists of all the proponent's strategies in T.

With a set of witnesses constructed, the next step in our method of creation of the defense semantics is to determine the proponent's strategies. This can be achieved with the help of the attribute `CounterOpp`, formalized with the attribute domain $A_{\mathtt{CounterOpp}} := (2^{2^{\mathbb{B}}}, \cup, \boxtimes, \boxtimes, \cup, \boxtimes, \boxslash)$.

Proposition 8. *Let $T = (V, A, \mathrm{L}, \lambda, \mathrm{actor}, \mathrm{ref})$ be an attack–defense tree, let $v \in V$ and $O \subseteq \mathbb{B}^{\mathrm{o}_T}$. Let β^O be the basic assignment for the* `CounterOpp` *attribute defined by*

$$\beta^O(\lambda(v)) := \begin{cases} \{\{\lambda(v)\}\}, & \textit{if } \mathrm{actor}(v) = \mathrm{p}_T, \\ \emptyset, & \textit{if } \mathrm{actor}(v) = \mathrm{o}_T,\ \lambda(v) \in O, \\ \{\emptyset\}, & \textit{if } \mathrm{actor}(v) = \mathrm{o}_T,\ \lambda(v) \notin O. \end{cases} \tag{28}$$

Let P be a set of basic actions of the proponent such that $P \in \mathtt{CounterOpp}_B(T, \beta^O, v)$. If $\mathrm{actor}(v) = \mathrm{p}_T$*, then* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 1$*; otherwise* $\mathrm{achieved}_{\mathrm{T}}(v, P \cup O) = 0$*.*

[3] In fact, T'' is the component of the subdag of T induced by the set V' that contains the root of T. Intuitively, T'' models the (relevant part of the) particular realization of the scenario modeled with T, when the opponent executes all of the actions in O, and the proponent — all of the actions in P.

Proof. The proof is by induction on the structure of $T(v)$ — the maximal subdag of T rooted at v. We distinguish several cases, depending on the refinement of and the actor assigned to v, as well as on the existence of $\bar{v}$.

For the base case, let v be a non-refined node and assume that $\bar{v}$ does not exist. Since the set $\texttt{CounterOpp}_B(T, \beta^O, v)$ is not empty, the definition of the basic assignment β^O implies that $\text{actor}(v) = \text{p}_T$ and $P = \{\lambda(v)\}$ or $\text{actor}(v) = \text{o}_T$ and $P = \emptyset$. In the former case, the claim follows immediately. In the latter, we have $\lambda(v) \notin O$, implying that

$$\text{achieved}_\text{T}(v, P \cup O) = \text{achieved}_\text{T}(v, O) = 0,$$

as required.

Case 1. The node v is not refined and $\bar{v}$ exists.

Case 1.1. $\text{actor}(v) = \text{p}_T$

Under the assumptions of this case, and since the set $\texttt{CounterOpp}_B(T, \beta^O, v)$ is not empty, formula (22), the definition of the `CounterOpp` domain, and the definition of the basic assignment β^O imply that $\texttt{CounterOpp}_B(T, \beta^O, \bar{v}) \neq \emptyset$ and $P = \overline{P} \cup \{\lambda(v)\}$, for some set $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. By the induction hypothesis, the equality $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O) = 0$ holds, and so the definition of the *satisfiability* attribute domain implies that $\text{achieved}_\text{T}(v, P \cup O) = 1$.

Case 1.2. $\text{actor}(v) = \text{o}_T$

In the case when $\lambda(v) \in O$, we have $\texttt{CounterOpp}_B(T, \beta^O, v) = \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, by formula (24) and the definition of the assignment β^O. Thus, $P \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, which together with the induction hypothesis implies that $\text{achieved}_\text{T}(\bar{v}, P \cup O) = 1$. Now the equality $\text{achieved}_\text{T}(v, P \cup O) = 0$ follows from the definition of the *satisfiability* attribute domain.

If $\lambda(v) \notin O$, then $\beta^O(\lambda(v)) = \{\emptyset\}$, and so $\texttt{CounterOpp}_B(T, \beta^O, v) = \{\emptyset\}$, by formula (24). And indeed, since $\lambda(v) \notin O$, the demanded equality

$$\text{achieved}_\text{T}(v, P \cup O) = \text{achieved}_\text{T}(v, O) = 0$$

follows from the definition of the *satisfiability* attribute domain.

For the cases when v is a refined node, we let $\text{children}_T(v) = \{v_1, \ldots, v_k\}$.

Case 2. The node v is refined and $\text{ref}(v) = \texttt{OR}$.

Case 2.1. $\text{actor}(v) = \text{p}_T$

Depending on whether or not $\bar{v}$ exists, either $P = P_i \cup \overline{P}$ (if $\bar{v}$ does exist) or $P = P_i$ (if $\bar{v}$ does not exist), for some $i \in \{1, \ldots, k\}$, $P_i \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$ and

$\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. Thus, by the induction hypothesis, we have $\text{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 0$ and $\text{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 1$, implying that $\text{achieved}_{\mathrm{T}}(v, P \cup O) = 1$.

Case 2.2. $\text{actor}(v) = \mathrm{o}_T$

Again, depending on the existence of $\bar{v}$, and, if it does exist, on whether or not the set $\texttt{CounterOpp}_B(T, \beta^O, \bar{v})$ is empty, either $P = (P_1 \cup \ldots \cup P_k)$, for some $P_i \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$ or $P = \overline{P}$, for some $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. In the latter case, we have $\text{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 1$, by the induction hypothesis, and the equality $\text{achieved}_{\mathrm{T}}(v, \overline{P} \cup O) = 0$ follows from the definition of the *satisfiability* attribute domain. Suppose now that the former of the two cases occurs. The induction hypothesis implies that $\text{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 0$, for $i \in \{1, \ldots, k\}$. Now, it follows from Corollary 4 that $\text{achieved}_{\mathrm{T}}(v_i, P \cup O) = 0$, for $i \in \{1, \ldots, k\}$. Thus, $\text{achieved}_{\mathrm{T}}(v, P \cup O) = 0$.

Case 3. The node v is refined and $\text{ref}(v) = \texttt{AND}$.

Case 3.1. $\text{actor}(v) = \mathrm{p}_T$

Depending on the existence of the countermeasure $\bar{v}$, it either holds that $P = (P_1 \cup \ldots \cup P_k) \cup \overline{P}$ or $P = (P_1 \cup \ldots \cup P_k)$, for some $P_i \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$ and $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. The induction hypothesis implies that $\text{achieved}_{\mathrm{T}}(v_i, P_i \cup O) = 1$, for $i \in \{1, \ldots, k\}$, and $\text{achieved}_{\mathrm{T}}(\bar{v}, \overline{P} \cup O) = 0$. By applying Corollary 4, we get $\text{achieved}_{\mathrm{T}}(v, P \cup O) = 1$.

Case 3.2. $\text{actor}(v) = \mathrm{o}_T$

If $\bar{v}$ does not exist or $\texttt{CounterOpp}_B(T, \beta^O, \bar{v}) = \emptyset$, then, by formula (24), $P = P_i$, for some $i \in \{1, \ldots, k\}$ and $P_i \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$. Otherwise, it might hold that $P = \overline{P}$, for some $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. In either case, the demanded equality follows from induction hypothesis and the definition of the *satisfiability* attribute domain. □

Proposition 8 states in particular that every set belonging to $\texttt{CounterOpp}_B(T, \beta^O, \text{root}(T))$, with β^O defined by (28), counters the set O of basic actions of the opponent in T. With the next proposition we establish another useful fact: that every *minimal* set countering O also belongs to $\texttt{CounterOpp}_B(T, \beta^O, \text{root}(T))$.

Proposition 9. *Let* $T = (V, A, \mathrm{L}, \lambda, \text{actor}, \text{ref})$ *be an attack–defense tree,* $v \in V$*,* $O \subseteq \mathbb{B}^{\mathrm{o}_T}$ *and let* β^O *be the basic assignment defined by* (28)*. If*

- $\text{actor}(v) = \mathrm{p}_T$ *and* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *is a minimal set such that* $\text{achieved}_{\mathrm{T}}(v, P \cup O) = 1$*, or*
- $\text{actor}(v) = \mathrm{o}_T$ *and* $P \subseteq \mathbb{B}^{\mathrm{p}_T}$ *is a minimal set such that* $\text{achieved}_{\mathrm{T}}(v, P \cup O) = 0$*,*

then $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$*.*

Proof. The proof is again by induction on the structure of $T(v)$, the maximal subdag of T rooted at v. For the base case, assume that v has no children at all, i.e., that

$\text{children}_T(v) = \emptyset$ and that $\bar{v}$ does not exist. If $\text{actor}(v) = \text{p}_T$, then the only set P satisfying the assumptions of the theorem is $P = \{\lambda(v)\}$. If $\text{actor}(v) = \text{o}_T$, then either no such P exists (if $\lambda(v) \in O$) or else $P = \emptyset$ (if $\lambda(v) \notin O$). In either case, the statement holds.

We now proceed with the remaining cases.

Case 1. The node v is not refined and $\bar{v}$ exists.

Case 1.1. $\text{actor}(v) = \text{p}_T$

Since $\text{achieved}_\text{T}(v, P \cup O) = 1$, the assumptions of this case imply that $\text{achieved}_\text{T}(\bar{v}, P \cup O) = 0$. From the minimality of P it follows that P can be represented as $\overline{P} \cup \{\lambda(v)\}$, for some minimal set $\overline{P}$ satisfying $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O) = 0$. By the induction hypothesis, we have $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, and so $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$.

Case 1.2. $\text{actor}(v) = \text{o}_T$

The proof in this case is analogous to that from the previous one. Nevertheless, we include it for completeness. Since $\text{achieved}_\text{T}(v, P \cup O) = 0$, it follows from the definition of the *satisfiability* domain domain that $\text{achieved}_\text{T}(\bar{v}, P \cup O) = 1$. The minimality of P implies that $P = \overline{P}$ for some minimal set $\overline{P}$ satisfying $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O) = 1$. As $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, by the induction hypothesis, the definition of the `CounterOpp` attribute domain now implies that $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$, as required.

For a proof of the remaining cases, when v is a refined node, we let $\text{children}_T(v) = \{v_1, \ldots, v_k\}$ and assume that the node $\bar{v}$ exists. The proof for the cases when $\bar{v}$ does not exist is obtained by skipping the parts related to $\bar{v}$ in what follows.

Case 2. The node v is refined and $\text{ref}(v) = \texttt{OR}$.

Case 2.1. $\text{actor}(v) = \text{p}_T$

We begin with proving that there exists $i \in \{1, \ldots, k\}$, a minimal set P' for which $\text{achieved}_\text{T}(v_i, P' \cup O) = 1$ and a minimal set $\overline{P}$ for which $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O) = 0$, such that $P = P' \cup \overline{P}$. To obtain such sets P' and $\overline{P}$, proceed iteratively as follows. Set $P' := P$, $\overline{P} := P$. As long as there exists a basic action $\texttt{b} \in \overline{P}$ such that $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O \backslash \{\texttt{b}\}) = 0$, set $\overline{P} := \overline{P} \backslash \{\texttt{b}\}$. Similarly, as long as there exists a basic action $\texttt{b} \in P'$ such that $\text{achieved}_\text{T}(v_i, P' \backslash \{\texttt{b}\} \cup O) = 1$, for at least one $i \in \{1, \ldots, k\}$, remove $\texttt{b}$ from P'. Observe that, by the minimality of P, the actions that were removed from $\overline{P}$ belong to P', and those removed from P' belong to $\overline{P}$. In other words, the equality $P = P' \cup \overline{P}$ indeed holds. Furthermore, the sets P' and $\overline{P}$ are minimal sets satisfying $\text{achieved}_\text{T}(\bar{v}, \overline{P} \cup O) = 0$ and $\text{achieved}_\text{T}(v_i, P' \cup O) = 1$, for some $i \in \{1, \ldots, k\}$. Thus, by the induction hypothesis, we have that $P' \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$ and $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. Hence,

$P \in \texttt{CounterOpp}_B(T, \beta^O, v)$.

Case 2.2. $\text{actor}(v) = \text{o}_T$

If $P \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, then the definition of the `CounterOpp` attribute domain and operation $\boxslash$ defined by (24) imply that $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$. Thus, in this case the claimed statement holds.

Assume now that $P \notin \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. Since $\text{achieved}_\text{T}(v, P \cup O) = 0$, it follows from the definition of the *satisfiability* attribute domain that for every $i \in \{1, \ldots, k\}$ the equality $\text{achieved}_\text{T}(v_i, P \cup O) = 0$ holds. Furthermore, P being a *minimal* set satisfying $\text{achieved}_\text{T}(v, P \cup O) = 0$ implies that it can be represented as

$$P = P_1 \cup \ldots \cup P_k \tag{29}$$

for some minimal sets $P_1, \ldots, P_k$ satisfying $\text{achieved}_\text{T}(v_i, P_i \cup O) = 0$. To see that this is indeed the case, suppose for a proof by contradiction that it is not. Then, in every representation (29) of P there is a set P_i satisfying $\text{achieved}_\text{T}(v_i, P_i \cup O) = 0$, for some $i \in \{1, \ldots, k\}$, that is not a minimal set having this property. Let $P'_1 \cup \ldots \cup P'_k$ be a representation (29) of P that minimizes the number of such non-minimal sets, and let P'_j be such a non-minimal set. Then, there is a set $P''_j \subset P'_j$ that is a minimal set for which $\text{achieved}_\text{T}(v_j, P''_j \cup O) = 0$ holds. The first statement of Corollary 4 implies that

$$\text{achieved}_\text{T}(v_i, P'_1 \cup \ldots \cup P''_j \cup \ldots \cup P'_k \cup O) = 0$$

for every $i \in \{1, \ldots, k\}$. Thus,

$$\text{achieved}_\text{T}(v, P'_1 \cup \ldots \cup P''_j \cup \ldots \cup P'_k \cup O) = 0.$$

But from the choice of $P'_1 \cup \ldots \cup P'_k$ it follows that

$$P \supset P'_1 \cup \ldots \cup P''_j \cup \ldots \cup P'_k.$$

This contradicts the minimality of P. Thus, the set P admits the representation (29).

Now it follows from the induction hypothesis that for every $i \in \{1, \ldots, k\}$, the set P_i from (29) belongs to $\texttt{CounterOpp}_B(T, \beta^O, v_i)$. Together with the definition of the `CounterOpp` attribute domain and operation $\boxtimes$ defined by (22) this fact implies that $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$, completing the proof of this case.

Case 3. The node v is refined and $\text{ref}(v) = \texttt{AND}$.

Case 3.1. $\text{actor}(v) = \text{p}_T$

The assumptions of this case and the fact that P is a minimal set for which the equality $\text{achieved}_\text{T}(v, P \cup O) = 1$ holds imply that P can be represented as $P = P_1 \cup \ldots \cup P_k \cup \overline{P}$, for some minimal sets $P_1, \ldots, P_k$ and $\overline{P}$ satisfying $\text{achieved}_\text{T}(v_i, P_i \cup$

$O) = 1$ and $\text{achieved}_{\text{T}}(\bar{v}, \overline{P} \cup O) = 0$. By the induction hypothesis, we have $P_i \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$, for $i \in \{1, \ldots, k\}$, and $\overline{P} \in \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$. Thus, $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$, by the definition of the `CounterOpp` attribute domain and operation $\boxtimes$ defined by (22).

Case 3.2. $\text{actor}(v) = \text{o}_T$

Similarly as in Case 2.1, we assume that $P \notin \texttt{CounterOpp}_B(T, \beta^O, \bar{v})$, since otherwise the claimed statement follows immediately. In this case, the definition of the *satisfiability* domain and the fact that P is a minimal set satisfying $\text{achieved}_{\text{T}}(v, P \cup O) = 0$ imply that there is $i \in \{1, \ldots, k\}$ for which P is a minimal set satisfying $\text{achieved}_{\text{T}}(v_i, P \cup O) = 0$. Thus, by the induction hypothesis, $P \in \texttt{CounterOpp}_B(T, \beta^O, v_i)$. Now it follows immediately from the definition of the `CounterOpp` attribute domain that $P \in \texttt{CounterOpp}_B(T, \beta^O, v)$. □

Proposition 8 and 9 yield immediately the following result.

Corollary 8. *Let T be an attack–defense tree and O be an opponent's strategy in T. With β^O being the basic assignment defined by* (28)*, the minimal (w.r.t. the inclusion) sets from $\texttt{CounterOpp}_B(T, \beta^O, \text{root}(T))$ are the proponent's strategies in T witnessed by O.*

The final ingredient of our algorithm for creation of the defense semantics is a method for determining minimal opponent's strategies countering a given proponent's strategy. Conceptually, this task is the same as the one achieved by the domain of the `CounterOpp` attribute, but it requires, informally speaking, switching of the actors. What we mean by this, is the following: for an attack–defense tree T, let T' be the tree obtained by attaching the root of T as a countermeasure to a new node belonging to o_T. Assume that the the new node bears a unique label, say x. Then, $\text{p}_{T'} = \text{o}_T$, $\text{o}_{T'} = \text{p}_T$ and for every proponent's strategy P in T there is a set $O' = P$ of basic actions of the opponent in T'. Thus, when creating proponent's strategies countering O' in T', one in fact creates the opponent's strategies countering P in T. That is, every opponent's strategy countering O' in T' is of the form $P' \cup \{x\}$, where $P' = O$, for some opponent's strategy O in T countering P.

Thus, we define the domain $A_{\texttt{CounterPro}} := (2^{2^{\mathbb{B}}}, \boxtimes, \cup, \cup, \boxtimes, \boxslash, \boxtimes)$, with the operations performed by the bottom-up evaluation at the nodes of the proponent being the ones performed at the nodes of the opponent in the attribute domain $A_{\texttt{CounterOpp}}$, and vice versa. Finally, for an attack–defense tree $T = (V, A, \text{L}, \lambda, \text{actor}, \text{ref})$ and a set $P \subseteq \mathbb{B}^{\text{p}_T}$ of basic actions of the proponent let

$$\beta^P(\lambda(v)) := \begin{cases} \{\{\lambda(v)\}\}, \text{ if } \text{actor}(v) = \text{o}_T, \\ \emptyset, \text{ if } \text{actor}(v) = \text{p}_T, \lambda(v) \in P, \\ \{\emptyset\}, \text{ if } \text{actor}(v) = \text{p}_T, \lambda(v) \notin P. \end{cases} \tag{30}$$

The above reasoning implies the following.

Corollary 9. *Let T be an attack–defense tree and P be a proponent's strategy in T. With β^P being the basic assignment defined by* (30)*, the minimal (w.r.t. the inclusion) sets from* $\texttt{CounterPro}_B(T, \beta^P, \mathrm{root}(T))$ *are the minimal opponent's strategies in T countering P.*

The considerations of this section, in particular Corollary 7, 8 and 9, imply that the procedure described in Algorithm 3 is indeed suitable for creating a defense semantics of an attack–defense tree.

Corollary 10. *On input attack–defense tree T, Algorithm 3 outputs the defense semantics $\mathcal{D}(T)$ of T.*

Regarding the complexity of Algorithm 3, we note that

- in the worst case, the number of the opponent's strategies created using the `SuffWit` attribute domain is exponential in both the number of basic actions of the opponent and the number of the opponent's nodes in the tree (see, e.g., Example 55),
- the number of proponent's strategies witnessed by a given opponent's strategy can be exponential in both the number of basic actions of the proponent and the number of the proponent's nodes in the tree (e.g., in a tree obtained by attaching to the node labeled `d` in the tree $\texttt{C}^{\texttt{p}}(\texttt{b}, \texttt{d})$ the root node of a tree belonging to the family described in Example 35),
- the number of minimal opponent's strategies countering a given proponent's strategy can be exponential in both the number of basic actions of the opponent and the number of the opponent's nodes in the tree (e.g., in a tree T obtained by attaching as a countermeasure to the root node of the tree `b` the root of a tree T' belonging to the family described in Example 35, with the nodes of T' belonging to the opponent in T).

The above examples imply that for a tree T with n nodes, the time needed for execution of each of the lines 1, 4 and 8 is in $\mathcal{O}(2^n)$, implying that Algorithm 3 returns the defense semantics of T in time $\mathcal{O}(2^{2n})$. We note that it seems impossible to construct a tree in which each of the three lines would indeed require a number of operations exponential in the number of basic actions in the tree.

6.3 Optimal selection of countermeasures

We will now demonstrate how the information stored in the defense semantics of an attack–defense tree can be exploited for the purpose of optimal selection of countermeasures to be implemented by the opponent. We provide a generic framework for solving

optimization problems expressed in terms of integer linear programming. We consider single parameter and multi-parameter cases, we deal with proponent and opponent-related parameters, and we show how to proceed in a stochastic case. The contents of this section are inspired by and based on [ZALT19].

6.3.1 The mathematical model

We first present the mathematical model that we use to address the optimization problems. It relies on a number of variables modeling behavior of the two actors. Given an attack–defense tree T and its defense semantics $\mathcal{D}(T)$, let

- $\mathtt{b}_1, \ldots, \mathtt{b}_p$ be the basic actions of the opponent present in T,
- $P_1, \ldots, P_n$ be the distinct proponent's strategies that appear in $\mathcal{D}(T)$,
- $O_1, \ldots, O_m$ be the distinct opponent's strategy that appear in $\mathcal{D}(T)$.

Furthermore, for $k \in \{1, \ldots, p\}$, $i \in \{1, \ldots, n\}$, and $j \in \{1, \ldots, m\}$, we set

$$y_{kj} = \begin{cases} 1, \text{ if } \mathtt{b}_k \in O_j, \\ 0, \text{ otherwise,} \end{cases} \qquad P_{ij} = \begin{cases} 1, \text{ if } (P_i, O_j) \in \mathcal{D}(T), \\ 0, \text{ otherwise.} \end{cases}$$

Every basic action $\mathtt{b}$ of the opponent is assumed to be assigned a non-negative integer cost value $\mathtt{cost}(\mathtt{b})$. The budget available to the opponent is denoted by $\mathcal{B}$. The ways in which execution of particular actions contributes to the implementation of opponent's strategies, which, in turn, results in some proponent's strategies being countered, are modeled with inequalities involving Boolean variables:

- x_k, for $k \in \{1, \ldots, p\}$: $x_k = 1$ if and only if the opponent executes action $\mathtt{b}_k$,
- z_i, for $i \in \{1, \ldots, n\}$: $z_i = 1$ if and only if the proponent's strategy P_i achieves the root node of T in the presence of currently deployed countermeasures,
- f_j, for $j \in \{1, \ldots, m\}$: $f_j = 1$ if and only if the opponent does not execute at least one of the basic actions from the opponent's strategy O_j.

6.3.2 Optimization problems in the deterministic case

We begin with the deterministic case, where there is no uncertainty about the outcome of the actions of the opponent, i.e., we assume that every action executed by the opponent succeeds, and that every countermeasure contributes fully to all the goals that depend on it.

General integer linear programming problem

The goal of the opponent is to select countermeasures to be implemented, in a way that optimizes a linear function F dependent on variables x_k, f_j, and z_i. The total cost of the countermeasures cannot exceed the budget $\mathcal{B}$ available to the opponent. The general form of such optimization problem is given in Figure 19.

Constraint (32) ensures that the opponent's investment cannot exceed their budget. The next two families of constraints model the meaning of the variables f_j: constraints (33) ensure that if the opponent does not execute some of the actions from O_j, then $f_j = 1$; constraints (34) ensures that if $f_j = 1$, then the opponent does not execute some action from O_j. Next, we model the meaning of the variables z_i: constraints (35) ensure that if the opponent does not execute some action in any of the sets countering P_i (i.e., $f_j = 1$ for every j, such that $P_{ij} = 1$), then $z_i = 1$; and constraints (36) ensure that if the opponent executes all the actions from at least one of the sets O_j countering the proponent's strategy P_i (i.e., there exists j, such that $P_{ij} = 1$ and $f_j = 0$), then $z_i = 0$[4].

Remark 4. *Observe that the number of elements in the opponent's strategy O_j can be expressed as $|O_j| = \sum_{k=1}^{p} y_{kj}$. Thus, the opponent executes all of the actions from O_j if and only if*

$$\sum_{k=1}^{p} y_{kj} = \sum_{k=1}^{p} x_k y_{kj},$$

which is equivalent to

$$\sum_{k=1}^{p} (1 - x_k) y_{kj} = 0.$$

In consequence, if there is j for which the above equality holds and $P_{ij} = 1$, then the proponent cannot succeed by employing the proponent's strategy P_i. Conversely, if for all j with $P_{ij} = 1$ the above equality does not hold, then the proponent can achieve the root goal with P_i. This explains the form of inequalities (33) *and* (34).

Let us have a look at specific instances of this problem.

Coverage problem. Setting $F := -\sum_{i=1}^{n} z_i$ results in so called *coverage problem*, where the goal is to maximize the number of proponent's strategies countered by the opponent.

Countering the most appealing proponent's strategies

For an attack–defense tree T, let $S\colon 2^{\mathbb{B}^{\mathrm{P}_T}} \to \mathbb{Z}^+$ be a *score function* used for comparing proponent's strategies. The higher the value of the score function of an proponent's strategy, the less appealing the strategy is for the proponent. If the opponent cannot fully protect the system, they can at least implement a set of countermeasures that maximizes the minimal value of the score function, among the proponent's strategies

[4]Note that the denominator in the right hand side of constraints (35) is a constant, i.e., the constraints (35) do not introduce any non-linearity into the programming problem.

$$\textbf{Optimization goal:} \qquad \text{maximize } F(x_1, \ldots, x_p, f_1, \ldots, f_m, z_1, \ldots, z_n) \tag{31}$$

$$\textbf{Subject to:} \qquad \sum_{k=1}^{p} \texttt{cost}(\texttt{b}_k) x_k \leqslant \mathcal{B} \tag{32}$$

$$f_j \geqslant \frac{\sum_{k=1}^{p} y_{kj}(1 - x_k)}{p},\ 1 \leqslant j \leqslant m \tag{33}$$

$$f_j \leqslant \sum_{k=1}^{p} y_{kj}(1 - x_k),\ 1 \leqslant j \leqslant m \tag{34}$$

$$z_i \geqslant 1 + \sum_{j=1}^{m} P_{ij}(f_j - 1),\ 1 \leqslant i \leqslant n \tag{35}$$

$$z_i \leqslant \frac{\sum_{j=1}^{m} P_{ij} f_j}{\sum_{j=1}^{m} P_{ij}},\ 1 \leqslant i \leqslant n \tag{36}$$

$$x_k \in \{0,1\},\ 1 \leqslant k \leqslant p, \qquad f_j \in \{0,1\},\ 1 \leqslant j \leqslant m, \quad z_i \in \{0,1\},\ 1 \leqslant i \leqslant n.$$

Figure 19: General integer linear programming problem for optimal selection of countermeasures

successful in the presence of this set. This is achieved by setting $F := C_S$, where $C_S \in \mathbb{Z}^+$ is a new variable, and by introducing constraints

$$C_S \leqslant z_i S(P_i) + 2(1 - z_i) \max_{P \in \{P_1, \ldots, P_n\}} S(P),\ \text{for } i \in \{1, \ldots, n\}. \tag{37}$$

Constraints (37) relate the value of C_S to the values of the score function attained by proponent's strategies not countered by the considered set of countermeasures. They ensure that C_S is always bounded from above by the minimum of these values, i.e., that maximizing C_S is beneficial for the opponent. Should all the proponent's strategies be countered by the opponent under some configuration of variables, then the optimal solution to the optimization problem will be

$$2 \max_{P \in \{P_1, \ldots, P_n\}} S(P).$$

The constant multiplier is a technical trick allowing for distinguishing the case when all the proponent's strategies can be countered (result exceeds the maximal of the scores of the proponent's strategies) from the case when they cannot (the result will correspond to the minimal of the scores among the proponent's strategies that are not countered).

Below, we present two instances of this problem that are of practical interest in risk analysis.

Countering the cheapest proponent's strategies. Typical example of score function S is the *cost of execution of a strategy*. Assume that the cost of the proponent's actions is modeled with non-negative integers, i.e., that there is a function `cost`()

defined on $\mathbb{B}^{\mathrm{p}}$ such that $\mathtt{cost}(\mathtt{b}) \in \mathbb{Z}^+$, for $\mathtt{b} \in \mathbb{B}^{\mathrm{p}_T}$. By solving the problem from Figure 19 extended with constraints (37) for $S(P) := \sum_{\mathtt{b} \in P} \mathtt{cost}(\mathtt{b})$, one obtains a set of countermeasures that maximizes the minimal necessary investment of the proponent into achieving the root goal.

Countering Pareto optimal proponent's strategies. Let us start with a generic mathematical setting. Suppose that there is a partial order $\preceq$ defined on the set of proponent's strategies $\mathcal{P} = \{P_1, \ldots, P_n\}$, and that maximal elements w.r.t. this order correspond to the strategies most appealing to the proponent. For a given $P \in \mathcal{P}$, denote by $\#P_{\preceq} \in \mathbb{Z}^+$ the number of elements of a largest totally ordered subset of $\mathcal{P}$, in which P is the minimal element.[5] The smaller the value of $\#P_{\preceq}$, the more appealing the proponent's strategy P is for the proponent, because there are not many strategies that are better than P. The opponent's objective is thus to first counter the proponent's strategies P for which the value of $\#P_{\preceq}$ is small. By applying the model from Figure 19 extended with constraints (37) for $S(P) := \#P_{\preceq}$, one identifies a set of countermeasures which maximizes the minimal number $\#P_{\preceq}$ over all proponent's strategies that are not countered, i.e., a set for which the uncountered strategies are as unattractive to the proponent as possible.

The setting described above applies to any partial order on the set of proponent's strategies. In particular, it can be used for countering Pareto optimal proponent's strategies. That is, should each of the proponent's strategies be assigned a vector of values originating from partially ordered sets, one could introduce a partial order $\preceq$ on the set of strategies, were the maximal elements are the strategies that are Pareto optimal w.r.t. all the considered parameters. By instantiating the above generic setting with this order, one selects a set of countermeasures that focuses on countering the proponent's strategies that are Pareto optimal in the scenario modeled with the tree.

Optimizing the opponent's investment without jeopardizing the system

Assume now that the opponent's budget is not limited, but they do not want to spend on security more than necessary. Suppose that there exists a solution to the coverage problem, in which all counterable proponent's strategies are countered. The opponent can identify a cheapest set of countermeasures countering all counterable proponent's strategies by solving the problem from Figure 19, for $F := -\sum_{k=1}^{p} \mathtt{cost}(\mathtt{b}_k) x_k$, with the constraints (32) and (36) being removed, and with additional n constraints $z_i \leqslant 0$, for $i \in \{1, \ldots, n\}$.

In the case of an attack–defense tree T in which all of the proponent's strategies can be countered, the optimization of the opponent's investment can be done using the methods presented in Chapter 4 and 5. This can be achieved with the trick performed when introducing the $A_{\mathtt{CounterPro}}$ attribute domain on page 140: by creating an attack–defense tree T' by attaching the root of T as a countermeasure to a new node belonging to o_T,

[5] Notice that $\#P_{\preceq}$ induces a total order on $\mathcal{P}$.

bearing a unique label. Then, $\mathrm{p}_{T'} = \mathrm{o}_T$, and so the value obtained using evaluation of the *minimal cost for the proponent* attribute on T' under the assumption that the opponent in T' (who is the proponent in T) performs all of their actions is in fact the minimal investment of the opponent in T needed for countering all proponent's strategies in T. The same maneuver can be employed for other attribute domains induced by semirings.

Optimization goal:

$$\text{maximize } G(x_1, \ldots, x_p) + \mathbb{E}[\mathcal{H}(x_1, \ldots, x_p, \xi)]$$

Subject to:

$$\sum_{k=1}^{p} \mathtt{cost}(\mathtt{b}_k) x_k \leqslant \mathcal{B}$$
$$x_k \in \{0, 1\},\ 1 \leqslant k \leqslant p$$

where $\mathcal{H}(x_1, \ldots, x_p, \xi)$ is the optimal value of the problem

$$\text{maximize } H(f_1, \ldots, f_m, z_1, \ldots, z_n)$$

Subject to:

$$f_j \geqslant \frac{\sum_{k=1}^{p} A_{kj}(1 - \xi_k x_k)}{p},\ 1 \leqslant j \leqslant m$$
$$f_j \leqslant \sum_{k=1}^{p} A_{kj}(1 - \xi_k x_k),\ 1 \leqslant j \leqslant m$$
$$z_i \geqslant 1 + \sum_{j=1}^{m} B_{ij}(f_j - 1),\ 1 \leqslant i \leqslant n$$
$$z_i \leqslant \frac{\sum_{j=1}^{m} B_{ij} f_j}{\sum_{j=1}^{m} B_{ij}},\ 1 \leqslant i \leqslant n$$
$$f_j \in \{0, 1\},\ 1 \leqslant j \leqslant m, \quad z_i \in \{0, 1\},\ 1 \leqslant i \leqslant n.$$

Figure 20: General stochastic integer programming problem

6.3.3 Stochastic model

All the problems considered in Section 6.3.2, assume that the opponent always perform their actions successfully. However, in practice, this is almost never the case. We sketch briefly a non–deterministic mode, where the countermeasures may fail. Formally, we associate with every basic action $\mathtt{b}_k \in \mathbb{B}^{\mathrm{o}_T}$ of the opponent a random variable ξ_k that is equal to 1 if $\mathtt{b}_k$ has been implemented successfully, and 0 otherwise (according to the Bernoulli distribution). After splitting the function F into two parts $F = G + H$,

with $G = G(x_1, \ldots, x_p)$, $H = H(f_1, \ldots, f_m, z_1, \ldots, z_n)$, the optimization problem from Figure 19 becomes then a stochastic programming problem (see, e.g., [SDR14]) given in Figure 20, where $\xi := (\xi_1, \ldots, \xi_p)$. This general problem can be instantiated similarly as the deterministic one. It can be solved using variants of the well-studied *sampling average approximation* approach [KSHdM02], or other efficient heuristics, e.g., as in [ZALT19]. An interested reader is referred to [ZALT19] for details.

6.4 Conclusion and future work

The main goal of the work presented in this chapter was to tackle the issue of determining optimal sets of countermeasures in attack–defense scenarios modeled with attack–defense trees. To this end, we developed a novel method for extracting rational behaviors of the actors from attack–defense trees possibly containing clones and countermeasures against countermeasures. We illustrated how the information stored in the resulting defense semantics can be employed for formulating numerous optimization problems in terms of (stochastic) integer linear programming. Some of the optimization problems formalized in this work have been implemented in the `OSEAD` tool. The practical evaluation of the framework developed in this chapter will be performed in Chapter 7.

The bottleneck of our approach is the defense semantic itself. It would be worthwhile to study possible ways of approximating the defense semantics, i.e., creating its smaller variants without significant loss in the information stored. One way of doing this could be to develop a procedure similar in the spirit to Algorithm 2 for the set semantics. Another possible approach would be to relax the definition of opponent's strategy, for instance by limiting the number of opponent's vectors originating from the same homogeneous subdag contained in the same opponent's strategy. Under this new definition, it seems that to create the set of sufficient witnesses it would suffice to replace the $\mathtt{OR}^{\mathrm{o}}_{witnesses} = \boxplus$ operation with the sets union, thus achieving a significant speedup.

Chapter 7

Tool support and a case study

To validate the theoretical developments of the previous three chapters in practice, and to make them easily usable by a wider public, we have created a tool that we call `OSEAD`– *Optimal Strategies Extractor for Attack–Defense trees*. At its core lies the `adtrees` Python package [Wid19] that we developed. While `OSEAD` is intended to be an easy-to-use tool supporting security analysis, the `adtrees` package is targeted at the scientific community, as it can serve as a convenient basis for implementing and testing new analysis methods for attack–defense trees. The `OSEAD` tool is described in Section 7.1.

In Section 7.2, we present a case study of an electricity theft scenario that we conducted using `OSEAD`. Attack–defense trees have been used in the past to perform practical studies of security scenarios. In [FFG+16], the security of ATM machines was analyzed. The main difference between [FFG+16] and the current study is that the former focuses on the modeling aspects only, i.e., it does not involve any quantitative analysis. In [BKMS12], an RFID-based management system has been analyzed. This work resulted in a list of guidelines describing how to carry out a case study involving the attack–defense tree modeling and its quantitative analysis. These guidelines were respected in our electricity theft study. However, [BKMS12] concentrates on analysis w.r.t. single parameter, and it uses only the bottom-up evaluation of attributes, which is not well-suited for trees with clones.

7.1 The `OSEAD` tool

The `OSEAD` tool from the user's perspective `OSEAD` aims at allowing its users to analyze trees in a simple and intuitive way, using methods described in Chapter 4–6. Users operate the tool in a step-by-step manner, via a graphical interface illustrated in Figure 21. The first step is to provide a file storing the structure of the attack–defense tree of interest, which is an XML file produced by ADTool [GJK+16], well-known software for creating attack–defense trees. Furthermore, should the user want to analyze an attack tree created with the help of ATCalc [ABvdB+13] or ATE [Asl16b], the output files of these tools can be easily transformed into an ADTool-like XML file with the help of

ATTOP [KSR+18].

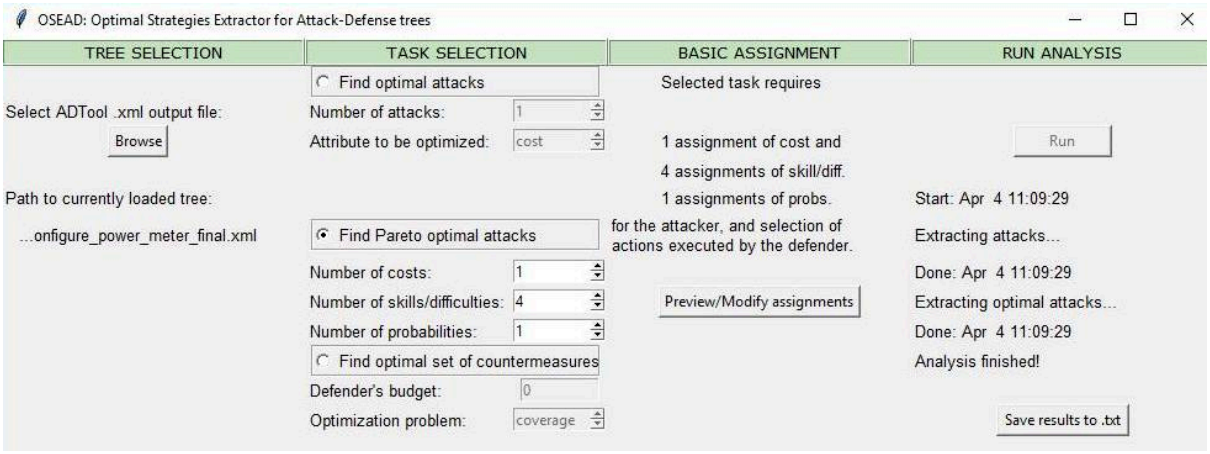


Figure 21: OSEAD's main user interface

Once the tree is provided, users select the problem of interest, which can be

- extraction of attacks[1] that optimize a single parameter (tab *Find optimal attacks* in Figure 21),
- extraction of attacks that are Pareto optimal (tab *Find Pareto optimal attacks*), or
- extraction of an optimal strategy of the defender (tab *Find optimal set of countermeasures*).

The last step preceding the actual analysis is the assignment of values of parameters of interest to the basic actions present in the tree. The values can be entered manually, imported from an XML file generated by ADTOOL, or loaded from a TXT file produced by OSEAD, as visualized in Figure 22. With all the inputs provided, OSEAD solves the optimization problem specified by the user. The results obtained can be exported to a TXT file (see Figure 23).

Implementation details OSEAD's computation engine and its user interface have been implemented in Python. Its architecture is depicted in Figure 24. The implementation model consists of the *Tree Model* (storing the tree structure), the *Attribute Domain* (object representing an attribute domain for attack–defense trees), the *ILP Problem* (derived from the *Tree Model*, using defense semantics, and storing the matrix of the selected optimization problem) and the *Basic Assignment* (storing values of parameters assigned to the basic actions).

The extraction of optimal attacks (tabs *Find optimal attacks* and *Find Pareto optimal attacks* in Figure 21) consists of two steps. In the first step, the evaluation of the selected

[1]The word *attack* is used here, since the default proponent in the trees created with ADTOOL is the attacker, and the opponent is the defender.

OSEAD: Optimal Strategies Extractor for Attack-Defense trees

Import Export Save & Exit

ATTACKER	cost_1	skill/diff_1	skill/diff_2	skill/diff_3	skill/diff_4	prob_1
	load from .xml	load from .xml	load from .xml	load from .xml	load from .xml	load from .xml
acquire information from dumpster diving	0.0	0.0	0.0	0.0	1000.0	0.2
acquire information from public Internet source	0.0	0.0	1.0	0.0	100.0	0.79
bribe technician to reconfigure the power meter	500.0	0.0	0.0	3.0	10.0	0.52
bribe technician to reveal power meter credentials	300.0	0.0	0.0	2.0	10.0	0.5
buy optical probe	71.2	0.0	1.0	0.0	100.0	1.0
coerce technician into reconfiguring the power meter	0.0	0.0	0.0	3.0	100.0	0.3
coerce technician into revealing power meter credentials	0.0	0.0	0.0	3.0	100.0	0.33
collect information by exchanging gossips with employees	0.0	0.0	0.0	1.0	1000.0	0.46
enter power meter credentials	0.0	0.0	0.0	0.0	0.0	0.99
extract credentials	0.0	0.0	1.0	0.0	10.0	0.56
find and download software for hacking power meters	0.0	1.0	1.0	0.0	10.0	0.9
get employed as field technician	0.0	0.0	2.0	1.0	1000.0	0.48
get employed as intern by the energy provider	0.0	0.0	1.0	1.0	1000.0	0.52
have physical access to the power meter	0.0	0.0	0.0	0.0	0.0	1.0
intercept credentials	0.0	2.0	1.0	0.0	0.0	0.62
locate encrypted credentials in the dump	0.0	2.0	2.0	0.0	100.0	0.6
make optical probe	14.0	0.0	2.0	0.0	100.0	0.41
make the data dump from hardware component	0.0	1.0	3.0	0.0	100.0	0.73
monitor communication between hardware components	0.0	1.0	2.0	0.0	100.0	0.5
perform brute force attack	0.0	1.0	2.0	0.0	100.0	0.65
reconfigure power meter using authorized software/tools	0.0	0.0	1.0	0.0	10.0	0.94
reconfigure power meter using unauthorized software	0.0	0.0	2.0	0.0	10.0	0.75
select technician for obtaining power meter credentials	0.0	0.0	0.0	0.0	100.0	1.0
select technician for reconfiguring power meter	0.0	0.0	0.0	0.0	100.0	1.0
technician reconfigures power meter using authorized software/tools	0.0	0.0	0.0	0.0	10.0	1.0
trick technician into revealing power meter credentials	0.0	0.0	0.0	2.0	100.0	0.24
use optical probe to establish connection to the meter via the optical port	0.0	0.0	1.0	0.0	10.0	0.95

DEFENDER	performed
enforce policy of using strong passwords	☐
enforce policy to minimize Internet disclosure	☑
enforce policy to minimize leakage of physical artefacts	☐
limit the allowed number of invalid authentication attempts	☐
password authentication for establishing connection	☐
require authentication for introducing changes in power consumption configuration	☑
thorough background check before hiring new employees	☑
track popular social engineering tricks and warn personnel	☐

Figure 22: Input management in OSEAD

```
Tree: ***/reconfigure_power_meter_final.xml

Optimization goal: determining Pareto optimal attacks wrt 1 costs, 4 skills/difficulties and 1 probabilities

Countermeasures implemented by the defender:
    enforce policy to minimize Internet disclosure
    password authentication for establishing connection
    track popular social engineering tricks and warn personnel

7 optimal attacks ('value: actions constituting the attack')
    [0.0, 0.0, 2.0, 1.0, 1000.0, 0.45119999999999993]: get employed as field technician, have physical access to the

    [71.2, 1.0, 2.0, 0.0, 100.0, 0.412644375]: buy optical probe, enter power meter credentials, find and download s

    [14.0, 1.0, 2.0, 0.0, 100.0, 0.16918419375000002]: enter power meter credentials, find and download software for

    [500.0, 0.0, 0.0, 3.0, 1000.0, 0.10400000000000001]: acquire information from dumpster diving, bribe technician

    [0.0, 0.0, 0.0, 3.0, 1000.0, 0.06]: acquire information from dumpster diving, coerce technician into reconfiguri

    [500.0, 0.0, 1.0, 3.0, 1000.0, 0.12438400000000002]: bribe technician to reconfigure the power meter, collect in

    [0.0, 0.0, 1.0, 3.0, 1000.0, 0.07176]: coerce technician into reconfiguring the power meter, collect information

Analysis performed under the following basic assignment:
    acquire information from dumpster diving, [[0.0, 0.0, 0.0, 0.0, 1000.0, 0.2]]
    acquire information from public Internet source, [[0.0, 0.0, 1.0, 0.0, 100.0, 0.79]]
    bribe technician to reconfigure the power meter, [[500.0, 0.0, 0.0, 3.0, 10.0, 0.52]]
    bribe technician to reveal power meter credentials, [[300.0, 0.0, 0.0, 2.0, 10.0, 0.5]]
    buy optical probe, [[71.2, 0.0, 1.0, 0.0, 100.0, 1.0]]
    coerce technician into reconfiguring the power meter, [[0.0, 0.0, 0.0, 3.0, 100.0, 0.3]]
    coerce technician into revealing power meter credentials, [[0.0, 0.0, 0.0, 3.0, 100.0, 0.33]]
    collect information by exchanging gossips with employees, [[0.0, 0.0, 0.0, 1.0, 1000.0, 0.46]]
    enter power meter credentials, [[0.0, 0.0, 0.0, 0.0, 0.0, 0.99]]
    extract credentials, [[0.0, 0.0, 1.0, 0.0, 10.0, 0.56]]
    find and download software for hacking power meters, [[0.0, 1.0, 1.0, 0.0, 10.0, 0.9]]
    get employed as field technician, [[0.0, 0.0, 2.0, 1.0, 1000.0, 0.48]]
    get employed as intern by the energy provider, [[0.0, 0.0, 1.0, 1.0, 1000.0, 0.52]]
    have physical access to the power meter, [[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]
    intercept credentials, [[0.0, 2.0, 1.0, 0.0, 0.0, 0.62]]
    locate encrypted credentials in the dump, [[0.0, 2.0, 2.0, 0.0, 100.0, 0.6]]
    make optical probe, [[14.0, 0.0, 2.0, 0.0, 100.0, 0.41]]
    make the data dump from hardware component, [[0.0, 1.0, 3.0, 0.0, 100.0, 0.73]]
    monitor communication between hardware components, [[0.0, 1.0, 2.0, 0.0, 100.0, 0.5]]
    perform brute force attack, [[0.0, 1.0, 2.0, 0.0, 100.0, 0.65]]
    reconfigure power meter using authorized software/tools, [[0.0, 0.0, 1.0, 0.0, 10.0, 0.94]]
    reconfigure power meter using unauthorized software, [[0.0, 0.0, 2.0, 0.0, 10.0, 0.75]]
    select technician for obtaining power meter credentials, [[0.0, 0.0, 0.0, 0.0, 100.0, 1.0]]
    select technician for reconfiguring power meter, [[0.0, 0.0, 0.0, 0.0, 100.0, 1.0]]
    technician reconfigures power meter using authorized software/tools, [[0.0, 0.0, 0.0, 0.0, 10.0, 1.0]]
    trick technician into revealing power meter credentials, [[0.0, 0.0, 0.0, 2.0, 100.0, 0.24]]
    use optical probe to establish connection to the meter via the optical port, [[0.0, 0.0, 1.0, 0.0, 10.0, 0.95]]
```

Figure 23: Results generated by `OSEAD`

attribute on the set semantics of the tree is performed, yielding both the set semantics of the tree and the optimal value of the attribute. In the second step, a topological sorting of the strategies is performed, w.r.t. their corresponding values, which allows for returning specified number of the "best" strategies. Depending on the optimization problem selected, the "best" strategies can be the cheapest ones, the ones most likely to succeed, the ones requiring the least level of skill, or the Pareto optimal ones.

The task of *Finding optimal set of countermeasures* requires selecting optimization problem to be solved, which can be either the *coverage problem* (see page 143) or the *attacker's investment problem* (where the defender aims at maximizing the necessary investment of the attacker, as described on page 144). The additional input needed here is the budget available to the defender. The task is tackled by creating the defense semantics of the tree and using it for formulating the corresponding integer linear programming problem. The problem itself is solved with the help of the free linear programming solver LP_SOLVE [BEN05].

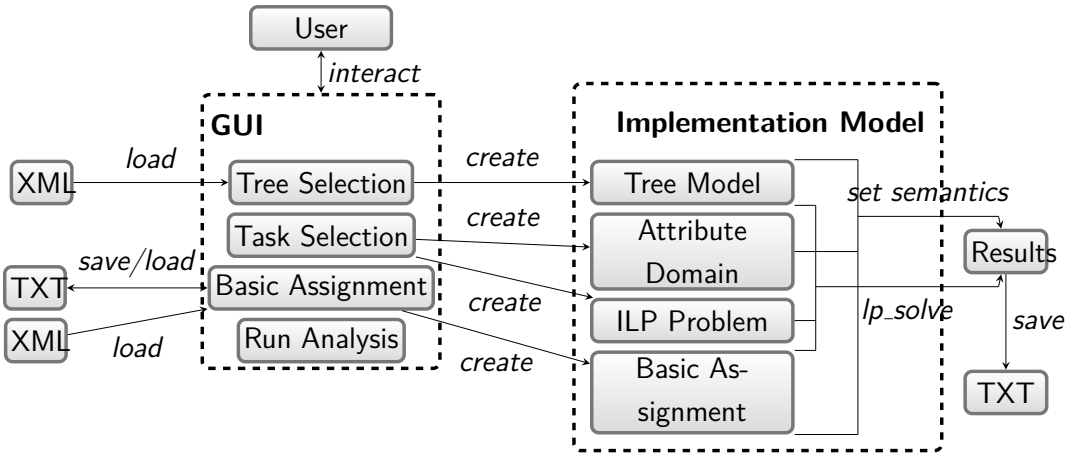


Figure 24: An overview of the OSEAD architecture

OSEAD is open source and it runs on all main platforms. The version for Windows can be downloaded from https://people.irisa.fr/Wojciech.Widel/suftware/osead.zip. Using OSEAD on other platforms requires installing the *adtrees* Python package [Wid19].

OSEAD's performance To solve the optimization problems, OSEAD creates either the set semantics or the defense semantics of the tree provided. In the worst case, the size of each of these semantics is exponential in the number of basic actions in the tree. Another possible bottleneck in the process of determining an optimal strategy for the defender is solving an integer linear programming problem.

In non-extremal cases, OSEAD performs well. Each of the problems considered in the case study described in Section 7.2 was solved in time not exceeding one second. We have also tested OSEAD's performance on trees having structure significantly more complex than the one considered in the case study, i.e., on trees encoding hundreds and thousands of attacks. Using some of the trees considered in Chapter 5, in Table 11, we have measured the time OSEAD needs to determine Pareto optimal attacks[2]. An excerpt from the tests' results is presented in Table 12.

7.2 Case study: electricity theft scenario

Electricity theft is a widespread practice [KD13, Kre12] that generates huge financial losses yearly across the world [Fre19, Kia18, LLC14, T&15], with more than the third of the losses affecting the BRIC countries (Brazil, Russia, India and China) [LLC14]. One of the ways in which electricity is being stolen, is by tampering with power meter in a way that results in the household's or facility's power consumption being under-reported.

[2] The XML files storing the trees are available at https://github.com/wwidel/pareto-tests/tree/master/trees, while the basic assignments used are to be found at https://github.com/wwidel/pareto-tests/tree/master/assignments.

Table 12: OSEAD's runtime for determining Pareto optimal attacks

Name of file storing tree structure	Number of basic actions	Name of file storing basic assignment	Number of attacks	Number of Pareto optimal attacks	Runtime in seconds
tree03	16	*tree03_1_cost*	640	2	1
tree10	26	*tree10_1_cost*	14336	3	438
tree12	17	*tree12_5_costs*	2436	63	11
tree29	22	*tree29_5_costs*	640	304	1
tree30	23	*tree30_5_costs*	704	184	1
tree32	25	*tree32_5_costs*	832	378	1

Modern smart meters make identifying crude power meter tampering attempts easier, but remain vulnerable to (not necessarily sophisticated) hacking attacks [Ms.12].

This study is concerned with the issue of tampering with power meters. We consider a malicious user whose aim is to reconfigure their power meter, in order to lower the recorded electricity consumption of their household. We extend the attack tree-based model of possible behavior of such a user, analyzed by the U.S. Department of Energy in [Nat15], to take possible countermeasures into account.

7.2.1 Description of the scenario

The set-up We consider a fifth year student of an engineering school, whom we will name *Marcel*, who is renting an apartment where he needs to pay for the electricity consumption. Marcel would like to lower his electricity bill and he decided to achieve this by reconfiguring the power meter in his apartment. In this study, Marcel plays a role of an *attacker* and his opponent, i.e., a *defender*, is the electricity provider. The meter under study is equipped with an optical port that allows a user to connect to the meter using an optical probe (see Figure 25 and 26).

The starting point of our analysis was the scenario and the attack tree described in Section 2.3 of [Nat15]. We complemented this tree with additional attacks, and added possible countermeasures that we identified based on [Car09, McC10], and [Web12]. The resulting attack–defense tree contains 68 nodes, 5 repeated basic actions of the attacker and 3 repeated basic actions of the defender. The XML file, compatible with ADTool and OSEAD, containing the entire attack–defense tree for tampering with the power meter is available at `https://people.irisa.fr/Wojciech.Widel/studies/meter_study.zip`.

The scenario In order to reconfigure his power meter via optical port, Marcel has to have physical access to the power meter and reconfigure it using appropriate software tools. Since the power meter is located in the apartment where Marcel lives, we assume

Figure 25: A power meter with an optical port (source: `https://nl.wikipedia.org/wiki/IEC_62056`)

that accessing the power meter is a basic action, i.e., the corresponding node is not refined. In order to reconfigure the power meter with the help of software, we have identified the following three sub-scenarios that Marcel can follow, taking into account his knowledge, capabilities, and financial profile:

The *do it yourself* approach – Marcel reconfigures the meter himself by using unauthorized software tools (Figure 28, 29, 30, 31, 32),

The *social engineering* approach – Marcel social engineers a technician employed by the electricity provider to reconfigure the power meter for him using authorized software tools (Figure 33),

The *get employed* approach – Marcel gets employed by the electricity provider as a field technician to gain access to the authorized tools and to be able to reconfigure the meter himself (Figure 34).

This high-level view of the analyzed scenario is presented by the tree from Figure 27, where the black triangles illustrate subtrees presented in further figures. We now detail the three approaches considered by Marcel.

The *do it yourself* approach

To reconfigure the power meter by himself, Marcel needs to obtain unauthorized software and tools, use optical probe to establish connection with the meter via its optical port, and finally reconfigure the meter using unauthorized software. He can find and download unauthorized software from the Internet. As for the optical probe, he can buy it or make it himself. The corresponding tree is given in Figure 28.

Figure 26: Optical probe connected to the power meter (source: https://www.aliexpress.com/item/China-Manufacturer-DHL-free-Shipping-electricity-optical-meter-reading/32455842504.html?spm=2114.10010108.100009.1.6810cc24soIZC4&gps-id=pcDetailLeftTopSell&scm=1007.13482.95643.0&scm_id=1007.13482.95643.0&scm-url=1007.13482.95643.0&pvid=65873a85-f01b-4876-970d-b58b38041880)

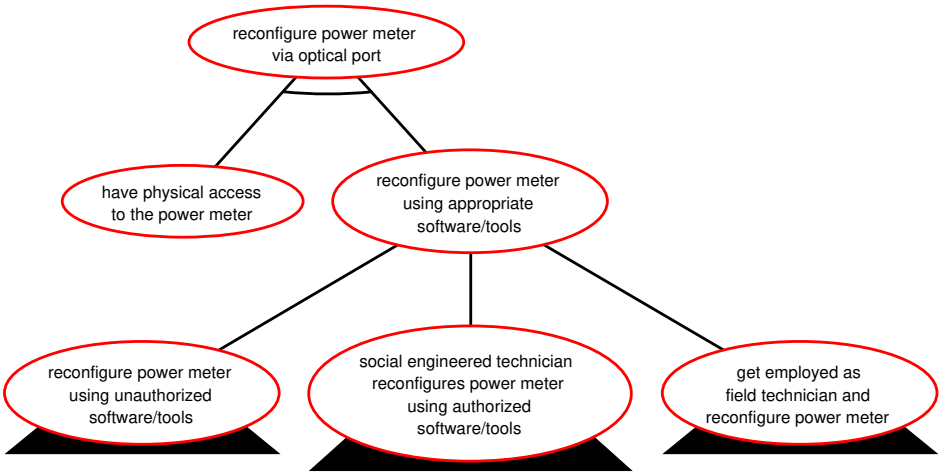


Figure 27: How to reconfigure the power meter – a high level view

Establishing connection to the meter via its optical port might be secured by password authentication. Also, independently of whether a password-based protection is implemented or not, an authentication could be required before the power consumption configuration can be modified. These two possible countermeasures are present in the tree in Figure 28.

If the connection to the power meter was protected by a password, Marcel could still reach his goal if he was able to authenticate using the correct credentials. To do so, he would need to obtain the credentials and enter them to the power meter while authenticating, as visualized in Figure 29. The power meter credentials could be obtained by

- exploiting the hardware components of the power meter (Figure 30),
- performing a brute force attack (Figure 31), or
- social engineering a technician working for the energy provider (Figure 32).

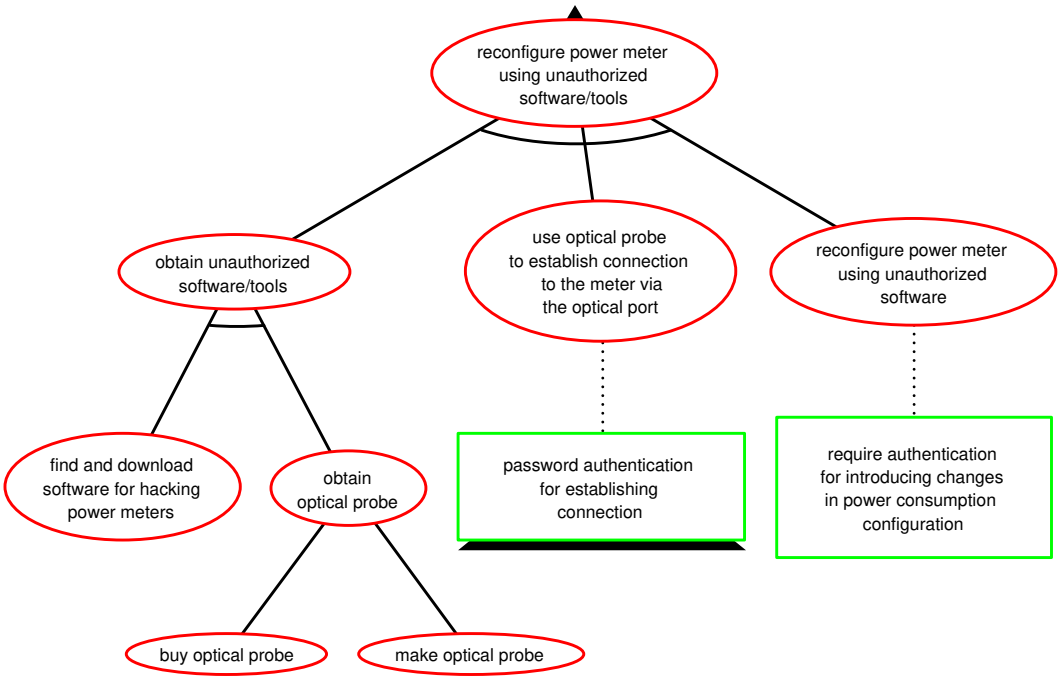


Figure 28: The *do it yourself* approach

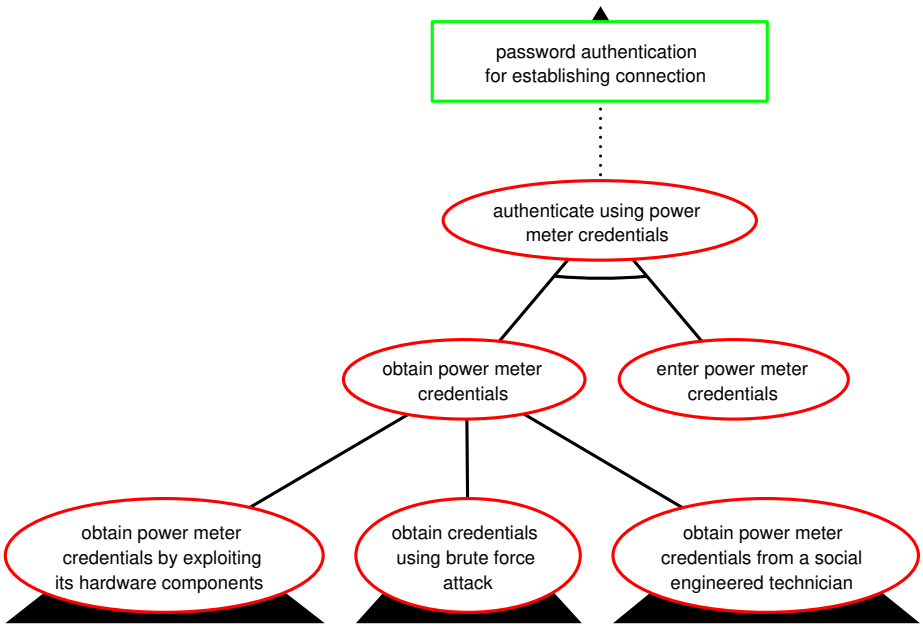


Figure 29: Overcoming the password-based authentication

Extracting credentials from the power meter hardware components, illustrated in Figure 30, can be achieved in two ways: either by extracting them from a data dump or by spying on communication between the hardware components. To extract the credentials from the data dump, the dump needs to be made, the location where encrypted credentials are placed in the dump needs to be identified, and finally the credentials need

to be extracted from the encrypted dump. To extract the credentials from the communication between the hardware components, the communication needs to be monitored and the credentials need to be intercepted. In this study, we assume that during the communication between the hardware components, the data are sent unencrypted.

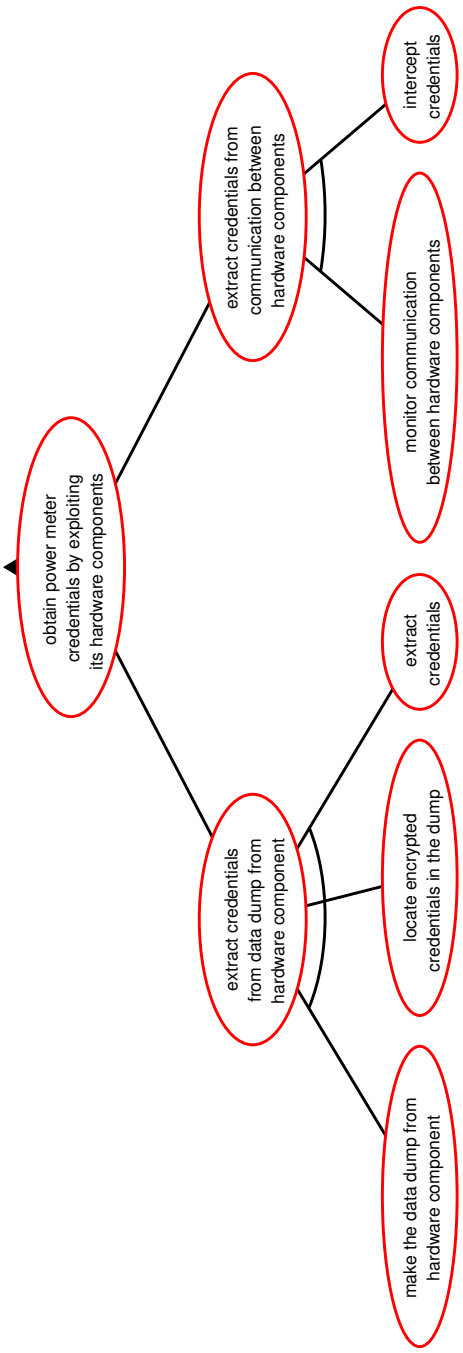


Figure 30: Obtaining power meter credentials from its hardware components

A brute force attack is illustrated in Figure 31. It makes use of software for hacking power meters (in our scenario, this is exactly the same software as the one used by the attacker to reconfigure power meter). An off-line brute force attack using tools like

`Ophcrack` [Oph16], `John the Ripper` [tR16], or `hashcat` [has16], can be prevented if a strong password is used. To make an on-line cracking impossible, the number of possible invalid authentication attempts could be limited.

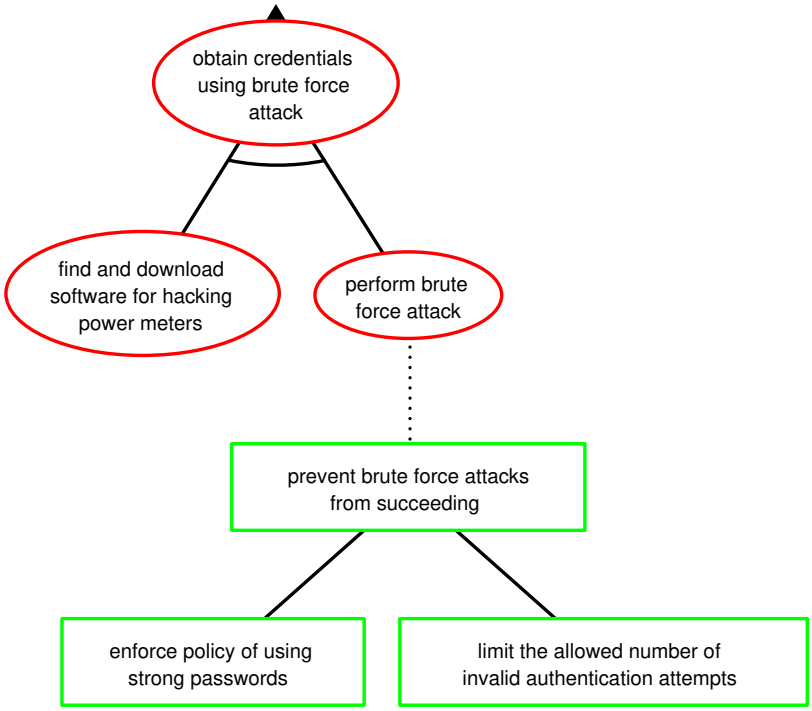


Figure 31: Obtaining credentials by brute force attack

Finally, credentials could also be obtained by social engineering a technician, as depicted in Figure 32. To do so, a suitable technician would need to be selected and social engineered. A social engineering attack would require to assemble background information on employees of the energy provider and to select one who would fall into the social engineering attack to reveal the credentials. Marcel could obtain the background knowledge on employees by searching on the Internet, diving into dumpster and looking for relevant documents and physical artefacts, or by infiltrating the energy provider. To infiltrate the energy provider, Marcel could get hired as an intern student and then collect information by exchanging gossips with the company employees. The following policies could be enforced by the company to prevent access to the background information about its employees:

- a policy to minimize the Internet disclosure,
- a policy to minimize the leakage of physical documents and artefacts,
- a policy of performing thorough background check before hiring new employees.

Once the right social engineering target is selected, the attack itself consists in bribing, coercing or tricking the technician so that they reveal the power meter credentials. The

tricking attack could be prevented by a security training during which the personnel is made aware of popular social engineering tricks.

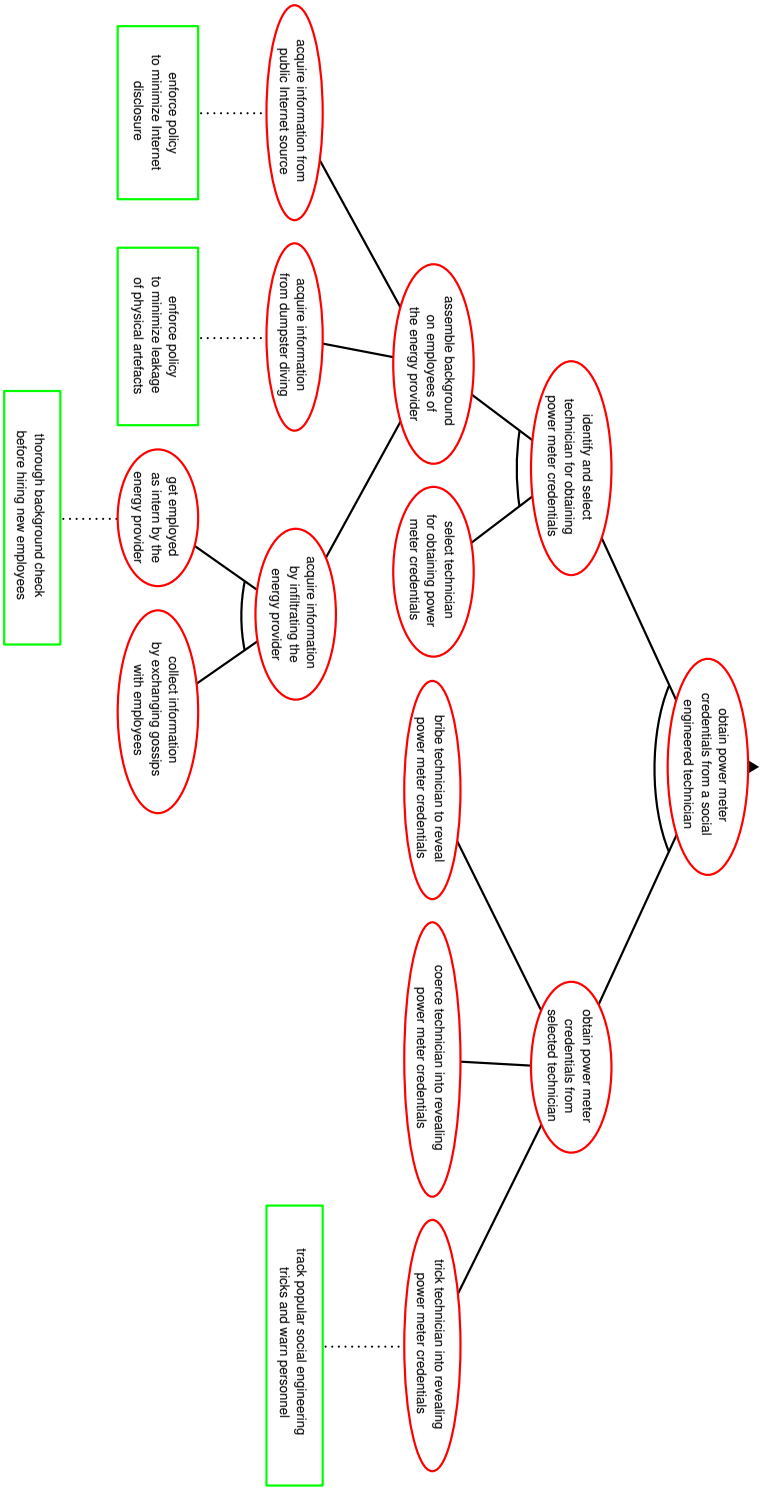


Figure 32: Obtaining credentials by social engineering a technician

The *Social engineering* approach

Instead of attacking by himself, Marcel can social engineer a technician, so that they reconfigure the power meter for him, as modeled in Figure 33.

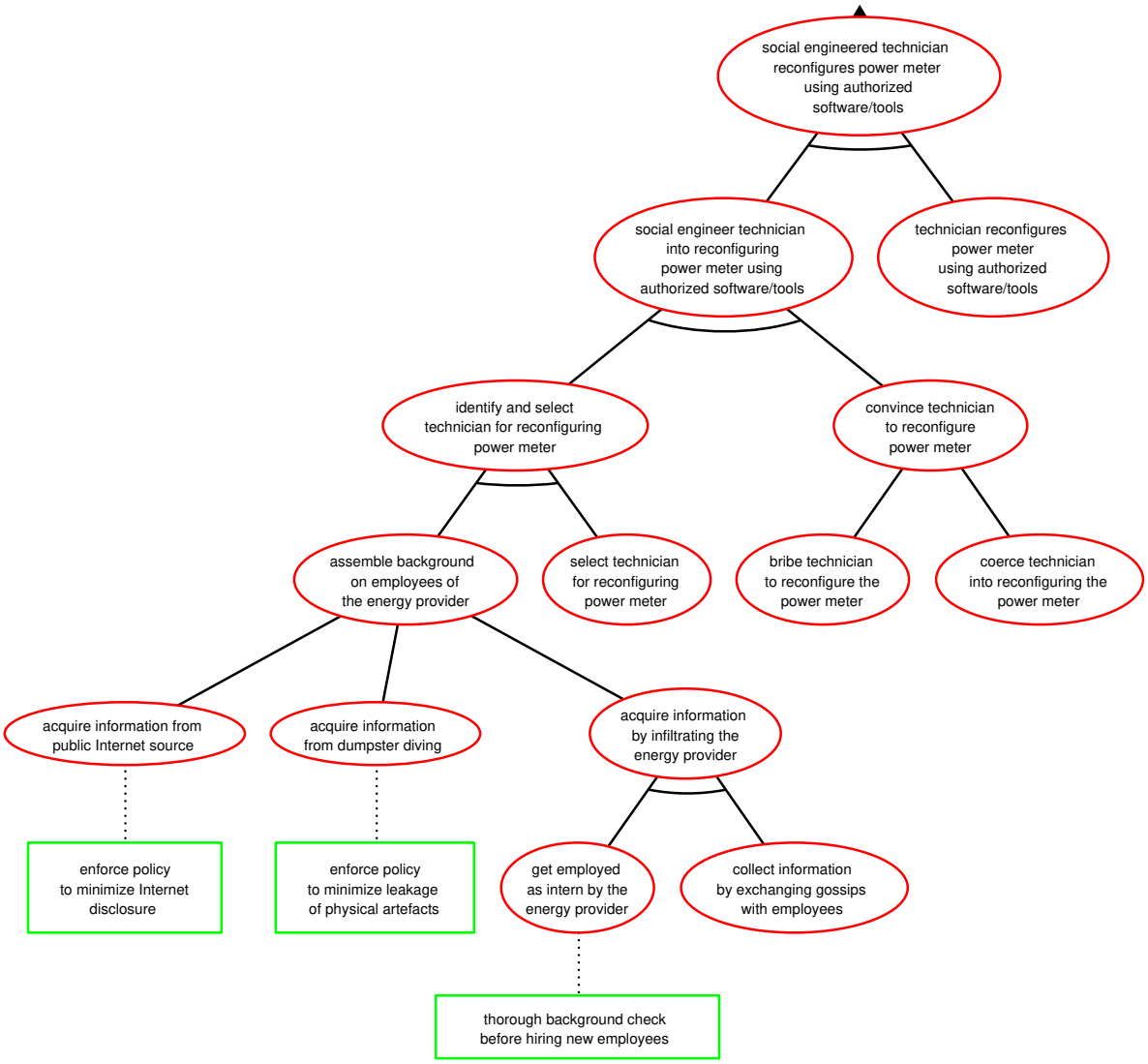


Figure 33: The *social engineering* approach

To perform the social engineering, a suitable technician who would reconfigure the power meter needs to be identified and Marcel needs to convince them to reconfigure the meter. Identification of the suitable social engineering target is performed in exactly the same way as in the *do it yourself* approach, by assembling relevant background knowledge on employees. Once identified, the technician who will reconfigure the power meter is selected. To persuade the technician to reconfigure the power meter, Marcel can bribe or coerce them.

The *get employed* approach

Marcel can also get hired by the power provider company to be officially able to reconfigure power meters. To do so, he needs to get employed as a field technician and then reconfigure his power meter using authorized software provided by the company to its technicians. Performing thorough background check on future employees would mitigate this attack, as it was the case in the two previous approaches. The get employed attack is illustrated in Figure 34.

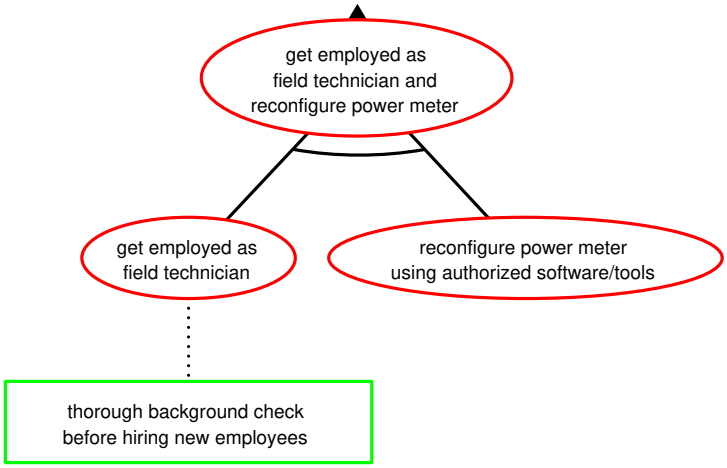


Figure 34: The *get employed* approach

7.2.2 Quantitative analysis of the tampering scenario

The first objective of this case study is to analyze the scenario described in Section 7.2.1. This includes enumeration of all possible attacks, identification of those that are optimal from the point of view of the attacker, as well as pinpointing the countermeasures that offer the best protection to the analyzed system. In what follows, we will use the word *attack* for *a set of basic actions of the proponent that belongs to a minimal strategy in the tree*. By *defender's strategy*, we understand a *set of countermeasures* that the defender can implement to secure the system (a set of basic actions that the defender can execute). The three types of optimization problems that we tackle in this study are:

- selection of attacks optimal w.r.t. one parameter,
- selection of attacks optimal w.r.t. several parameters,
- selection of the defender's strategy optimal from the point of view of their resources and objective.

We begin with describing the attributes of interest for the case study. We give their names, the semirings inducing their corresponding attribute domains, and the values that

they can attain. The process of estimation of the input values, i.e., the basic assignments for the attributes, is then described. Some issues related to the reliability of the input values and the computation methods used are discussed in Section 7.3.

The parameters used

***Cost*, domain induced by $(\mathbb{R}_{\geqslant 0} \cup \{+\infty\}, \mathbf{min}, +)$** The first parameter of interest is the *monetary investment* necessary to implement an attack (or a defender's strategy). To express it, we use non-negative real numbers representing the necessary investment in euro. The actions that are too expensive to be executed are assigned the value of $+\infty$.

***Time*, domain induced by $(\{0, 10, 10^2, 10^3, 10^4, +\infty\}, \mathbf{min}, \mathbf{max})$** Since Marcel would like to lower his electricity bill as soon as possible, the *time that an attack would take* is an important parameter to consider. The following scale is used to express time values:

- *Instantaneous* (0): can be performed by the actor in less than a minute.
- *Quick* (10): can be performed by the actor in less than an hour, but not less than a minute.
- *Slow* (10^2): can be performed by the actor in less than a week, but not less than an hour.
- *Very slow* (10^3): can be performed by the actor in less than six months, but not less than a week.
- *Extremely slow* (10^4): can be performed by the actor within a human lifetime, but not less than six months.
- *Impossible* ($+\infty$): not doable within a human lifetime.

Since this scale is discrete, it is reasonable to assume that the time necessary to perform an attack is the *maximum* value over the time values of its composing actions. As in the case of cost, we are interested in *minimizing* the time necessary to attack the system, thus we select the attack which requires *minimal* time.

***Success probability*, domain induced by $([0, 1], \mathbf{max}, \cdot)$** Attacks that are very cheap or very fast are useless if their probability of succeeding is negligible. Here, we are thus interested in what is the probability that, if executed, an attack will be successful. The probability of successful execution of an action is a value from the interval $[0, 1]$, and the probability of an attack is the product of the probabilities assigned to the actions constituting the attack[3]

[3]Recall that we are working under the assumption of the basic actions being independent.

The remaining three parameters assess the level of special skills – cybersecurity, technical, and social – that is necessary to be able to perform an action successfully. In all three cases, the skill level necessary to perform an attack is defined as the *maximum* among the skill levels necessary to perform its components. By optimal, we mean an attack requiring *minimal* skill level.

***Cybersecurity skills level*, domain induced by $(\{0, 1, 2, 3, +\infty\}, \mathbf{min}, \mathbf{max})$** Some of the actions considered in our scenario may require specific *expertise regarding cybersecurity*. We distinguish five levels of such expertise:

- *None* (0): no cybersecurity-related skills required.
- *Basic* (1): requires basic cybersecurity knowledge and skills.
- *Advanced* (2): requires employing advanced cybersecurity-related skills, e.g., executing a man in the middle attack on a protocol.
- *Expert* (3): requires employing cybersecurity-related skills available to few experts, e.g., return-oriented programming or fault attack on AES.
- *Impossible* ($+\infty$): beyond the known capability of today's human beings.

***Technical skills level*, domain induced by $(\{0, 1, 2, 3, +\infty\}, \mathbf{min}, \mathbf{max})$** Similarly to cybersecurity skills, some actions may require some *technical expertise*. Here again, we distinguish five levels:

- *None* (0): no technical skills required.
- *Basic* (1): requires basic technical skills, e.g., finding information online.
- *Advanced* (2): requires advanced technical skills, available for graduates of technical vocational schools.
- *Expert* (3): requires technical skills available to experienced engineers.
- *Impossible* ($+\infty$): beyond the known capability of today's human beings.

***Social skills level*, domain induced by $(\{0, 1, 2, 3, +\infty\}, \mathbf{min}, \mathbf{max})$** Finally, since some attacks in our scenario rely strongly on social engineering, we are also interested in *social skills* necessary to perform the considered actions. The five levels of social skills are defined as follows:

- *None* (0): does not involve social interactions.
- *Basic* (1): requires basic social interactions, e.g., obtaining information via a conversation.

- *Advanced* (2): requires convincing or tricking someone into doing something they would not do otherwise.
- *Expert* (3): requires convincing or tricking someone into doing something punishable by law.
- *Impossible* ($+\infty$): beyond the known capability of today's human beings.

Estimation of input values

The analysis methods employed in our case study require numerical inputs, including the basic assignments of attributes to the basic actions. We now provide these values, and explain how they have been obtained.

The values of basic actions of the attacker that we have used in this study are given in Table 13. They represent a consensus reached as a result of the following procedure. Seven independent participants, whose profiles correspond to the expertise of Marcel, were involved in the values' estimation. The participants were given a document describing the scenario and the attack–defense tree from Section 7.2.1. They had access to the Internet and relevant materials, including [Car09, Nat15] and [Web12]. Each participant estimated the values for all six parameters for every basic action present in the tree. Unsurprisingly, some of the values were not consistent among different participants. A semi-automatic procedure has thus been used to extract a single value for each parameter at every basic action:

- for the parameters different than *probability*: if all (but one) among the seven values were the same, this value was retained,
- for the *probability* parameter, a simple average over seven values was computed,
- for the cases that do not fall into any of the above items, the retained value is the result of a discussion between the author of this thesis and Barbara Fila (Kordy),
- finally, in the case of strong disagreement, the author of the analyzed attack–defense tree who, among the seven participants, knows the best the optical meter technology, had the decisive power.

The estimation of values took one hour to each participant, on average. The consensus discussion lasted for 3 hours.

Table 14 gathers the basic actions of the defender and gives their cost. The values of the defender's cost represent the investment that the electricity provider needs to make to hire security experts who will advise the company on potential threats and suitable countermeasures against them, organize meetings where the decisions on policies to be implemented will be taken, put in place improved software or hardware solutions, for instance those allowing more secure authentication, and remunerate its personnel for performing specific activities, such as background checks before hiring new employees.

Table 13: Parameter values for basic actions of the attacker

Basic action	***Cost***	***Time***	***Prob***	***Cyber***	***Tech***	***Social***
acquire information from dumpster diving	0	1000	0.2	0	0	0
acquire information from public Internet source	0	100	0.79	0	1	0
bribe technician to reconfigure the power meter	500	10	0.52	0	0	3
bribe technician to reveal power meter credentials	300	10	0.5	0	0	2
buy optical probe	71.2	100	1	0	1	0
coerce technician into reconfiguring the power meter	0	100	0.3	0	0	3
coerce technician into revealing power meter credentials	0	100	0.33	0	0	3
collect information by exchanging gossips with employees	0	1000	0.46	0	0	1
enter power meter credentials	0	0	0.99	0	0	0
extract credentials	0	10	0.56	0	1	0
make the data dump from hardware component	0	100	0.73	1	3	0
find and download software for hacking power meters	0	10	0.9	1	1	0
get employed as field technician	0	1000	0.48	0	2	1
get employed as intern by the energy provider	0	1000	0.52	0	1	1
have physical access to the power meter	0	0	1	0	0	0
intercept credentials	0	0	0.62	2	1	0
locate encrypted credentials in the dump	0	100	0.6	2	2	0
make optical probe	14	100	0.41	0	2	0
monitor communication between hardware components	0	100	0.5	1	2	0
perform brute force attack	0	100	0.65	1	2	0
provide power meter credentials	0	0	1	0	0	0
reconfigure power meter using authorized software/tools	0	10	0.94	0	1	0
reconfigure power meter using unauthorized software	0	10	0.75	0	2	0
select technician for obtaining power meter credentials	0	100	1	0	0	0
select technician for reconfiguring power meter	0	100	1	0	0	0
technician reconfigures power meter using authorized software/tools	0	10	1	0	0	0
trick technician into revealing power meter credentials	0	100	0.24	0	0	2
use optical probe to establish connection to the meter via the optical port	0	10	0.95	0	1	0

Table 14: Cost of basic actions of the defender

Basic action	***Cost***
$\mathtt{d}_1$ = enforce policy of using strong passwords	11600
$\mathtt{d}_2$ = enforce policy to minimize Internet disclosure	9600
$\mathtt{d}_3$ = enforce policy to minimize leakage of physical artefacts	9600
$\mathtt{d}_4$ = limit the number of possible invalid authentication attempts	11600
$\mathtt{d}_5$ = password authentication for establishing connection	13600
$\mathtt{d}_6$ = require authentication for introducing changes in power consumption configuration	13600
$\mathtt{d}_7$ = thorough background check before hiring new employees	320
$\mathtt{d}_8$ = track popular social engineering attacks and warn personnel	1500

7.2.3 Optimal strategies for the attacker and the defender

We now present the results of the power meter tampering scenario analysis. We begin, in Section 7.2.4, by determining sets of countermeasures that the defender can implement under specified budget and that are optimal w.r.t. a given criterion (coverage or attacker's investment). For some of these sets, we then perform a *what-if* analysis: if a given strategy of the defender is implemented, what are the attacks optimal w.r.t. one (Section 7.2.5) or many (Section 7.2.6) parameters? Our objective is to verify whether an attacker having a profile of Marcel would be able to launch a successful attack on its power meter.

The analysis has been performed using the `OSEAD` tool. The files containing all the inputs used, as well as all of the obtained results, are available at `https://people.irisa.fr/Wojciech.Widel/studies/meter_study.zip`.

7.2.4 Selection of optimal sets of countermeasures

The choice of an optimal strategy for the defender depends on the budget that they have at their disposal, and on the optimization problem of interest. In our study, we consider a small, local electricity provider, and we thus analyze three possible values for the defender's budget: 20000, 30000, and 40000 euros. Table 15 presents optimal strategies for a defender interested in maximizing the number of prevented attacks (coverage problem) and another one focused on maximizing the necessary investment of the attacker necessary to achieve his objective (investment problem).

Requiring authentication for introducing changes in power consumption configuration ($\mathtt{d}_6$) and *performing thorough background check before hiring new employees* ($\mathtt{d}_7$) is an optimal strategy for a defender interested in covering a maximal number of possible attacks and having the budget of 20000 euros. We denote this strategy by D_1. Under the same budget, but with the goal of maximizing the necessary investment of the attacker in mind, the optimal behavior of the defender would be to *enforce policy to minimize*

Internet disclosure ($\mathtt{d}_2$), *enforce policy to minimize leakage of physical artefacts* ($\mathtt{d}_3$) and *perform thorough background check before hiring new employees* ($\mathtt{d}_7$). This ensures that the minimal necessary investment of the attacker into achieving the root goal is 14. This means, in particular, that the execution of the three actions prevents all the attacks having the cost of 0 euros.

The other two strategies that we consider are D_2 which corresponds to D_1 extended with the action of *enforcing policy to minimize Internet disclosure* ($\mathtt{d}_2$), and D_3 consisting of *enforcing policy to minimize Internet disclosure* ($\mathtt{d}_2$), *enforcing policy to minimize leakage of physical artefacts* ($\mathtt{d}_3$), *performing thorough background check before hiring new employees* ($\mathtt{d}_7$), and *tracking popular social engineering attacks and warning personnel* ($\mathtt{d}_8$). The strategies D_2 and D_3 are optimal for a defender having 30000 euros, and interested in the coverage problem and the attacker's investment problem, respectively.

Finally, a defender having 40000 euros is able to fully secure the analyzed system, by implementing the countermeasures $\mathtt{d}_2, \mathtt{d}_3, \mathtt{d}_6$, and $\mathtt{d}_7$. Due to space restrictions, we refer the reader to Table 14 for their meaning.

Table 15: Optimal strategies of the defender

	Coverage problem		**Investment problem**	
Defender's budget	Optimal strategy	Prevented /preventable	Optimal strategy	Necessary attacker's investment
20000	$D_1 = \{\mathtt{d}_6, \mathtt{d}_7\}$	29/33	$\{\mathtt{d}_2, \mathtt{d}_3, \mathtt{d}_7\}$	14
30000	$D_2 = \{\mathtt{d}_2, \mathtt{d}_6, \mathtt{d}_7\}$	31/33	$D_3 = \{\mathtt{d}_2, \mathtt{d}_3, \mathtt{d}_7, \mathtt{d}_8\}$	14
40000	$\{\mathtt{d}_2, \mathtt{d}_3, \mathtt{d}_6, \mathtt{d}_7\}$	33/33	$\{\mathtt{d}_2, \mathtt{d}_3, \mathtt{d}_6, \mathtt{d}_7\}$	$+\infty$

For the rest of our study, we retain the strategies D_1, D_2, and D_3 and look for optimal attacks in the case when one of these strategies is implemented by the defender.

7.2.5 Attacks optimizing single parameter

In total, there are 33 attacks[4] in the studied scenario. Their list is available at `https://people.irisa.fr/Wojciech.Widel/studies/meter_attacks.txt`. The attacks of interest for us are those that are not countered by at least one of the three defender's strategies D_1, D_2 or D_3. There are twelve such attacks, and they are presented in Table 16.

By analyzing Table 16, one notices that if the defender decides to implement one of the strategies D_1 or D_2, Marcel will be able to succeed only by executing some of the attacks from the *social engineering* approach. If the strategy D_3 is implemented, then the only possible attacks are those from the *do it yourself* approach.

[4]Recall that, in this chapter, the word *attack* has a meaning specified in the first paragraph of Section 7.2.2.

Table 16: Some of the attacks available to Marcel

Attacking approach: ***do it yourself*** **(Y);** ***social engineering*** **(S);** ***get employed*** **(E)**	S	S	S	S	Y	Y	Y	Y	Y	Y	Y	Y
Basic action	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}
acquire information from dumpster diving		✓		✓								
acquire information from public Internet source	✓		✓									
bribe technician to reconfigure the power meter			✓	✓								
bribe technician to reveal power meter credentials												
buy optical probe									✓	✓	✓	✓
coerce technician into reconfiguring the power meter	✓	✓										
coerce technician into revealing power meter credentials												
collect information by exchanging gossips with employees												
enter power meter credentials					✓		✓	✓		✓	✓	✓
extract credentials							✓	✓				
find and download software for hacking power meters					✓	✓	✓	✓	✓	✓	✓	✓
get employed as field technician												
get employed as intern by the energy provider												
have physical access to the power meter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
intercept credentials								✓		✓		
locate encrypted credentials in the dump							✓			✓		
make optical probe					✓	✓	✓	✓				
make the data dump from hardware component							✓			✓		
monitor communication between hardware components								✓			✓	
perform brute force attack					✓							✓
provide power meter credentials												
reconfigure power meter using authorized software/tools												
reconfigure power meter using unauthorized software					✓	✓	✓	✓	✓	✓	✓	✓
select technician for obtaining power meter credentials												
select technician for reconfiguring power meter	✓	✓	✓	✓								
technician reconfigures power meter using authorized software/tools	✓	✓	✓	✓								
trick technician into revealing power meter credentials												
use optical probe to establish connection to the meter via the optical port					✓	✓	✓	✓	✓	✓	✓	✓
Defender's strategy under which the attack is successful	D_1	D_1, D_2	D_1	D_1, D_2	D_3	D_3	D_3	D_3	D_3	D_3	D_3	D_3

Once the values corresponding to the attacks are obtained, OSEAD returns the optimal ones. We list them in Table 17. This table can be used to check whether an attacker of interest would be able to launch a successful attack. We recall that Marcel is a fifth year student of an engineering school. We assume that he has advanced technical skills, but he has only basic knowledge of cybersecurity. Being a student, he is not rich, but he can manage his time availability freely.

Table 17: Attacks optimal w.r.t. a single parameter and their values

	Attacks optimal w.r.t.					
Defender's strategy	***Cost***	***Time***	***Prob***	***Cyber***	***Tech***	***Social***
D_1	A_1, A_2	A_1, A_3	A_3	A_1, A_2, A_3, A_4	A_2, A_4	A_1, A_2, A_3, A_4
Optimal value	0	100	0.41	0	0	3
D_2	A_2	A_2, A_4	A_4	A_2, A_4	A_2, A_4	A_2, A_4
Optimal value	0	1000	0.10	0	0	3
D_3	A_5, A_6, A_7, A_8	A_5 – A_{12}	A_9	A_5, A_6, A_9, A_{12}	A_5, A_6, A_8, A_9, A_{11}, A_{12}	A_5 – A_{12}
Optimal value	14	100	0.64	1	2	0

Since the cost aspect is of the highest priority for Marcel, we assume that he would analyze the attacks optimal w.r.t. to this parameter first. The preference is given to attack A_2 which consists of *having physical access to the power meter*, *acquiring information from dumpster diving*, *selecting technician for reconfiguring power meter*, *coercing technician into reconfiguring power meter* and the *technician reconfiguring power meter using authorized software/tools*. While this attack is optimal from the point of view of *cost* and all the three *skills levels* under strategies D_1 and D_2, it would require from Marcel to force someone to perform an action punishable by law. Also, A_2 is not prevented by the strategy D_3. Indeed, implementation of D_3 counters all the attacks from the *social engineering* approach.

The strategy D_3 does not secure the meter from any attack in the *do it yourself* approach. An interesting attack within this approach is A_6, consisting of *having physical access to the power meter*, *making optical probe*, *finding and downloading software for hacking power meters*, *using optical probe to establish connection to the meter via the optical port*, and *reconfiguring power meter using unauthorized software*. Note that A_6 corresponds to the profile of Marcel, from the point of view of his resources and skills. Its only drawback is that its probability of success is quite low – only 0.26, as can be seen in Table 18.

Thanks to Table 17, we can also study the impact of the implemented countermeasures

on the attacks available to the attacker. Upgrading the system's protection from D_1 do D_2 (by *enforcing policy to minimize Internet disclosure*) at the cost of 9600 euros (see Table 14) is not worthwhile if the defender considers cheap attacks to be the most tempting for the attacker – the attack A_2 achieves the root goal under both strategies D_1 and D_2. However, if the defender aims at making the attacker less likely to succeed, then this investment is beneficial, as it lowers the attacker's success probability from 0.41 (for attack A_3 which would not work under D_2) to 0.10 (for A_4 that still works when D_2 is implemented).

7.2.6 Attacks optimizing several parameters

Unfortunately, for every attack listed in Table 17, i.e., optimal w.r.t. to one of the parameters, there is always another one that is better from the point of view of another parameter. To overcome this problem, we are now looking for Pareto optimal attacks, i.e., attacks that are not dominated by another one, while taking all six parameters into account simultaneously.

Table 18: Pareto optimal attacks and their values for: *cost* (c), *time* (t), *prob* (pb), *cyber skills* (cs), *tech. skills* (ts), and *social skills* (ss)

Defender's strategy	**Pareto optimal attacks**	**Values** (c, t, pb, cs, ts, ss)
D_1	A_1	(0, 100, 0.24, 0, 1, 3)
	A_2	(0, 1000, 0.06, 0, 0, 3)
	A_3	(500, 100, 0.41, 0, 1, 3)
	A_4	(500, 1000, 0.10, 0, 0, 3)
D_2	A_2	(0, 1000, 0.06, 0, 0, 3)
	A_4	(500, 1000, 0.10, 0, 0, 3)
D_3	A_6	(14.0, 100, 0.26, 1, 2, 0)
	A_9	(71.2, 100, 0.64, 1, 2, 0)

The Pareto optimal attacks are presented in Table 18, along with the values corresponding to their execution. Observe that under strategies D_1 or D_2, all of the attacks available to Marcel are Pareto optimal, including the attack A_2 discussed in the previous section. If the strategy D_3 is implemented by the defender, there exist eight possible attacks that achieve the root goal, but only two of them are Pareto optimal, namely A_6 and A_9. Observe that A_9 is a very interesting attack. It is almost the same as A_6, except that it involves *buying optical probe* instead of *making it*. Attack A_9 is optimal w.r.t. to all parameters, except *cost*. However, when checking its *cost* value, one realizes that the investment necessary to perform it (71.2 euros) would probably be acceptable for Marcel. The greatest advantage of A_9 is that its success probability (0.64) is significantly higher than that of A_6 (0.26).

The importance of the multi-parameter analysis is further illustrated by two facts. First, securing the system in a way that maximizes the necessary investment of the attacker, by implementing D_3, not only leaves the system vulnerable to more attacks than it is the case for the coverage problem (eight attacks versus two or four, see last row of Table 16), but also allows the attacker to execute attack A_9, which has a high probability of succeeding. Second, when the defender implements strategy D_3, the attack A_6 is among the cheapest ones, and the attack A_9 is the optimal one w.r.t. the probability. When we analyze the scenario taking only one of these parameters into consideration, we overlook one of these two attacks. But both of them are Pareto optimal, and as such, both can be considered equally appealing for the attacker.

7.3 On the reliability of the computation framework

Quantifying security is a highly disputable exercise. The reliability of the obtained results depends on the quality of the employed input values and on the suitability of the functions used to perform computations. Despite a great effort of the academic and the industrial communities, numerous underlying issues still remain unsolved. In this section, we debate on drawbacks that we met while performing this study, some of which we have not necessarily managed to overcome.

The quantitative analysis of graphical security models relies on numerical inputs whose exact values can almost never be provided. Their estimation is a difficult task that requires a thorough understanding of

- the parameters employed,
- the meaning of the basic actions present in the tree,
- the attacker's and defender's profiles and knowledge.

In practice, this estimation is very subjective, as it relies to a great extent on the modeler's expertise. In real-life, input values are usually based on historical data, statistics, information gathered from surveys or open sources, e.g., Internet. Such inputs inevitably carry some uncertainty about the values, and this uncertainty propagates during the computations and is accumulated in the final result of the analysis. While there is no established methodology for determining the best approximations of the actual values of the parameters under consideration, we believe that a reasonable estimates can still be obtained, if provided in collaboration with experts in the respective domains. Several industry practitioners performing security and risk analysis on a daily basis, that we had an opportunity to work with, suggest to follow a couple of simple rules.

- Finding a consensus through a discussion usually results in numbers that are more accurate than standard composite values, e.g., the average. People providing inputs might have misunderstood the significance of a parameter or the meaning of an

action, thus their values might be inconsistent. Computing a simple average over such values is meaningless. A discussion allows to identify such misunderstandings and results in a more reliable estimate.

– If a discrete scale is used, an odd number of possible values, such as *low-medium-high*, should be avoided. People having problems with deciding on the most suitable value, for instance due to the lack of knowledge, often tend to chose the middle value, because it seems to be the most neutral alternative. However, if numerous attacks get the same value, their ranking and thus a selection of the optimal ones become impossible.

– A way of taking the knowledge of the value providers into account is to complement the parameter value with the information on how certain the provider is about this value. Such an approach has, for instance, been used in the case study described in [BKMS12], where a *confidence* level was used in addition to the actual values of the parameters of interest. The confidence level plays a role of a weight, allowing to give more importance to values with high confidence (usually provided by experts) compared to those with low confidence (probably coming from less knowledgeable participants).

Note that in our study we decided not to use the confidence level, because our value providers had exactly the same profile as our potential attacker Marcel. We thus assumed that their estimates would be consistent with the estimates (and thus indirectly with the decisions) that Marcel would make.

Another factor possibly undermining the pertinence of the quantitative analysis of security are the computations performed on the input values during the analysis. We illustrate this issue on the examples of *probability* and *risk* metrics. An arguable but commonly used operator in the context of attack tree analysis is the multiplication employed to propagate the probability values at `AND` nodes in a bottom-up fashion. Using multiplication implies that attack components are considered to be independent, which is rarely the case in reality. This means that, even if the input values are correct, the probability computation might introduce some error or inaccuracy to the final result. To overcome this known drawback of the classical bottom-up propagation, some more advanced methods for computing attacks' probability have been proposed in the literature. Their weakness however lies in the fact that they often require sophisticated inputs, such as conditional probability tables [KPS16] or probability distributions [AHPS14], instead of simply probability points. An interested reader is referred to Section 7 of [WAFP19] for a description of some of the probabilistic frameworks for attack tree-based analysis. Another example highlighting both the importance and the difficulty of quantifying security is the *risk* metrics. Various formulas for risk exist. In [RSS15], the authors state that the standard way of defining risk is "the likelihood of an incident and its consequences for an asset", with all the words used having some specified meaning. This definition is used

for instance in the French risk analysis method *EBIOS* [ANS18]. It relies on two factors only, but other definitions are possible. In [EDRM06], risk has been defined in terms of cost, probability, and impact. For a discussion on possible three-factor and many-factor risk measure definitions see Chapter 11 of [RSS15] and references therein. On the one hand, the fact that there are many risk metrics definitions can be seen as a positive thing, because it allows the expert to select the one that is most suitable in a specific analysis context or w.r.t. the available input values. On the other hand, however, different risk formulas will provide different results, so it might be unclear which risk formalization should be used in which case.

To conclude this discussion section, we would like to stress that graphical security models are not the silver bullet for the risk assessment process, and that their role is to accompany other threat and risk analysis approaches, such as penetration testing, red teaming, standardized ISO 27XXX-compatible methods, e.g., [ANS18, LSS11], etc. Each of these methods focuses on different types of attacks and different security problems, so it is worthwhile to combine them in order to get the most complete and full-fledged results.

7.4 Conclusion and future work

In this chapter, we used attack–defense trees to analyze a realistic security scenario of tampering with a power meter. The study allowed us to validate the quantitative analysis methods discussed in Chapter 4-6. To facilitate and automate their usage, we have implemented the `OSEAD` tool described in Section 7.1.

We took great care so that our model and analysis are as unbiased and impartial as possible. The tree was created by crossing several industrial and academic sources, and the input values estimation was performed by independent participants with various cultural background, from Estonia, France, Poland, and Russia.

As discussed in Section 7.2.6, we were able to confirm the intuitive conjecture about the practical importance of the multi-parameter analysis. We note that, despite the fact that the algorithms implemented in `OSEAD` are highly complex, the tool performs extremely well when applied to trees encoding hundreds of attacks, and reasonably well in the case of trees with up to several thousands of attacks.

This study corroborates practical usefulness of attack–defense trees in security and risk analysis. However, solutions for some pragmatic issues still need to be found. The bottleneck of our study was the attribution of parameter values to basic actions. While for some parameters, e.g., *cost*, finding an accurate estimate is easy (nowadays, it suffices to search on the Internet), for some others, e.g., *success probability*, this task is much more difficult, if not impossible. More research and practical investigation is definitely necessary before a reliable methodology for the estimation of values for basic actions can be proposed.

Finally, we would like to emphasize that an attack tree-based analysis, as the one performed in this case study, does not fully cover the entire process of risk analysis. For instance, a practical issue regarding Marcel's return on investment was not discussed in our work. This issue includes the analysis of the actual gain of Marcel versus the necessary expenses related to making the tampering possible, or the estimation of minimal time after which Marcel's investment in attacking the system would start to pay back. Also, one should not forget about a completely separate dimension of risk of being arrested for performing illegal tampering. Although we judged these aspects out of scope of our study, in real life they should be investigated before a truly optimal attack can be identified.

Acknowledgements

We would like to thank the following students and researchers for their (far from being trivial) contribution to the estimation of parameter values used in this study: Jean-Loup Hatchikian-Houdot (INSA Rennes, France), Pille Pullonen (Cybernetica AS, Estonia), Artur Riazanov (Saint Petersburg Department of V.A. Steklov Institute of Mathematics of the Russian Academy of Sciences, Russia), Petr Smirnov (Saint Petersburg State University, Russia), and Aivo Toots (Cybernetica AS, Estonia).

Chapter 8

Conclusion

The main focus of this thesis were methods for quantitative analysis of security based on attack–defense trees, under a fixed interpretation of repeated labels. We studied the problem of attributes evaluation on such trees, including attributes suitable for multi-parameter analysis of security. The problem of optimal selection of countermeasures in security scenarios modeled with trees has also been investigated. Finally, we constructed a realistic attack–defense tree and performed a thorough case study of the corresponding scenario using the `OSEAD` tool that we have created.

For the convenience of a reader, we have decided to conclude each of the chapters separately. Here, we would like to only reiterate two important points raised in Chapter 7. The first of them is that attack–defense trees (in particular, attack trees) are just one of many tools available for risk analysts. Their proper usage is not easy, especially due to the process of their creation being error-prone; actions available to the actors might be overlooked and not included in the model, nodes might be labeled in an inappropriate way, giving raise to misleading results, etc. Just for these reasons, attack–defense trees should never be used as the sole device for performing risk analysis. The second issue regarding the practical usability of trees, and in particular the methods presented in this thesis, is the difficulty in obtaining reliable numerical inputs, as discussed in detail in Section 7.3.

There are many paths in the field of attack trees analysis that a curious researcher might pursue, with some of them highlighted in Section 4.8, 5.4 and 6.4. In the light of the difficulties described in the previous paragraph, one could hesitate whether these paths are worth pursuing. We believe so; even if the attack–defense trees itself will never become popular among risk analysts, they will remain closely related to Boolean functions and other modeling frameworks based on `AND`/`OR` trees, such as fault trees. Further theoretical work on attack–defense trees could thus result in new insights into problems arising in other research areas.

Bibliography

[ABvdB+13] Florian Arnold, Axel Belinfante, Freark van der Berg, Dennis Guck, and Mariëlle Stoelinga. Dftcalc: A tool for efficient fault tree analysis. In Friedemann Bitsch, Jérémie Guiochet, and Mohamed Kaâniche, editors, *Computer Safety, Reliability, and Security - 32nd International Conference, SAFECOMP 2013, Toulouse, France, September 24-27, 2013. Proceedings*, volume 8153 of *Lecture Notes in Computer Science*, pages 293–301. Springer, 2013. Available from: `https://doi.org/10.1007/978-3-642-40793-2_27`, `doi:10.1007/978-3-642-40793-2\_27`.

[AD90] Rajeev Alur and David L. Dill. Automata for modeling real-time systems. In Mike Paterson, editor, *Automata, Languages and Programming, 17th International Colloquium, ICALP90, Warwick University, England, UK, July 16-20, 1990, Proceedings*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 1990. Available from: `https://doi.org/10.1007/BFb0032042`, `doi:10.1007/BFb0032042`.

[AHPS14] Florian Arnold, Holger Hermanns, Reza Pulungan, and Mariëlle Stoelinga. Time-Dependent Analysis of Attacks. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 285–305. Springer, 2014. Available from: `https://doi.org/10.1007/978-3-642-54792-8_16`, `doi:10.1007/978-3-642-54792-8\_16`.

[AN15] Zaruhi Aslanyan and Flemming Nielson. Pareto efficient solutions of attack-defence trees. In Riccardo Focardi and Andrew C. Myers, editors, *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*, volume 9036 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2015. Available from: `https://doi.org/10.1007/978-3-662-46666-7_6`, `doi:10.1007/978-3-662-46666-7\_6`.

[AN17] Zaruhi Aslanyan and Flemming Nielson. Model checking exact cost for attack scenarios. In Matteo Maffei and Mark Ryan, editors, *Principles of Security and Trust - 6th International Conference, POST 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10204 of *Lecture Notes in Computer Science*, pages 210–231. Springer, 2017. Available from: https://doi.org/10.1007/978-3-662-54455-6_10, doi:10.1007/978-3-662-54455-6_10.

[ANP16] Zaruhi Aslanyan, Flemming Nielson, and David Parker. Quantitative verification and synthesis of attack-defence scenarios. In *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 105–119. IEEE Computer Society, 2016. Available from: https://doi.org/10.1109/CSF.2016.15, doi:10.1109/CSF.2016.15.

[ANS18] ANSSI. La Méthode EBIOS Risk Manager, 2018. Available from: https://www.ssi.gouv.fr/guide/la-methode-ebios-risk-manager-le-guide/.

[APK17] Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. Is My Attack Tree Correct? In Simon N. Foley, Dieter Gollmann, and Einar Snekkenes, editors, *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part I*, volume 10492 of *Lecture Notes in Computer Science*, pages 83–102. Springer, 2017. Available from: https://doi.org/10.1007/978-3-319-66402-6_7, doi:10.1007/978-3-319-66402-6_7.

[APK18] Maxime Audinot, Sophie Pinchinat, and Barbara Kordy. Guided design of attack trees: A system-based approach. In *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 61–75. IEEE Computer Society, 2018. Available from: https://doi.org/10.1109/CSF.2018.00012, doi:10.1109/CSF.2018.00012.

[APSW18] Maxime Audinot, Sophie Pinchinat, François Schwarzentruber, and Florence Wacheux. Deciding the non-emptiness of attack trees. In George Cybenko, David J. Pym, and Barbara Fila, editors, *5th International Workshop on Graphical Models for Security, held in conjunction with the Federated Logic Conference (FLoC) 2018, GraMSec@FLoC 2018, Oxford, UK, July 8, 2018, Revised Selected Papers*, volume 11086 of *Lecture Notes in Computer Science*, pages 13–30. Springer, 2018. Available from: https://doi.org/10.1007/978-3-030-15465-3_2, doi:10.1007/978-3-030-15465-3_2.

[Asl16a] Zaruhi Aslanyan. *Stochastic Model Checking of Socio-Technical Models*. PhD thesis, Terchnical University of Denmark, Denmark, 2016.

[Asl16b] Zaruhi Aslanyan. TREsPASS toolbox: Attack Tree Evaluator, 2016. presentation of a tool developed for the EU project TREsPASS. Available from: `https://vimeo.com/145070436`.

[ats18] ATSyRA Studio, 2018. Available from: `http://atsyra2.irisa.fr/`.

[BCSW06] Gerald G. Brown, W. Matthew Carlyle, Javier Salmerón, and R. Kevin Wood. Defending critical infrastructure. *Interfaces*, 36(6):530–544, 2006. Available from: `https://doi.org/10.1287/inte.1060.0252`, `doi:10.1287/inte.1060.0252`.

[BDL04] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. A tutorial on uppaal. In Marco Bernardo and Flavio Corradini, editors, *Formal Methods for the Design of Real-Time Systems, International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM-RT 2004, Bertinoro, Italy, September 13-18, 2004, Revised Lectures*, volume 3185 of *Lecture Notes in Computer Science*, pages 200–236. Springer, 2004. Available from: `https://doi.org/10.1007/978-3-540-30080-9_7`, `doi:10.1007/978-3-540-30080-9\_7`.

[BEN05] Michel Berkelaar, Kjell Eikland, and Peter Notebaert. lp_solve: Open source (Mixed-Integer) Linear Programming system, 2005. Version 5.5.2.5, dated September 24, 2016. Available from: `http://lpsolve.sourceforge.net/5.5/`.

[BK17] Angèle Bossuat and Barbara Kordy. Evil twins: Handling repetitions in attack-defense trees - A survival guide. In Liu et al. [LMS18], pages 17–37. Available from: `https://doi.org/10.1007/978-3-319-74860-3_2`, `doi:10.1007/978-3-319-74860-3\_2`.

[BKMS12] Alessandra Bagnato, Barbara Kordy, Per Håkon Meland, and Patrick Schweitzer. Attribute decoration of attack-defense trees. *IJSSE*, 3(2):1–35, 2012. Available from: `https://doi.org/10.4018/jsse.2012040101`, `doi:10.4018/jsse.2012040101`.

[BLWC17] Ahto Buldas, Aleksandr Lenin, Jan Willemson, and Anton Charnamord. Simple Infeasibility Certificates for Attack Trees. In Satoshi Obana and Koji Chida, editors, *Advances in Information and Computer Security - 12th International Workshop on Security, IWSEC 2017, Hiroshima, Japan, August 30 - September 1, 2017, Proceedings*, volume 10418 of *Lecture Notes in Computer Science*, pages 39–55. Springer, 2017. Available from: `https://doi.org/10.1007/978-3-319-64200-0_3`, `doi:10.1007/978-3-319-64200-0\_3`.

[BM08] John Adrian Bondy and Uppaluri S. R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008. Available from: https://doi.org/10.1007/978-1-84628-970-5, doi:10.1007/978-1-84628-970-5.

[BN98] Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge University Press, 1998.

[Car09] Matthew Carpenter. Advanced Metering Infrastructure Attack Methodology, 2009. Accessed: 2019-02-20. Available from: http://docshare.tips/ami-attack-methodology_5849023fb6d87fd2bb8b4806.html.

[CFK+13a] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013. Available from: https://doi.org/10.1007/s10703-013-0183-7, doi:10.1007/s10703-013-0183-7.

[CFK+13b] Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In Krishnendu Chatterjee and Jirí Sgall, editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013. Available from: https://doi.org/10.1007/978-3-642-40313-2_25, doi:10.1007/978-3-642-40313-2_25.

[Chv83] Vašek Chvátal. *Linear Programming*. W. H. Freeman, 1983.

[Cod06] D. Codetta-Raiteri. Bdd based analysis of parametric fault trees. In *RAMS '06. Annual Reliability and Maintainability Symposium, 2006.*, pages 442–449, Jan 2006. doi:10.1109/RAMS.2006.1677414.

[DBL19] *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*. IEEE, 2019. Available from: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=8804915.

[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 *MILCOM*, pages 1–7. IEEE, 2006.

[EK19] Julia Eisentraut and Jan Kretinsky. Expected Cost Analysis of Attack-Defense Trees, 2019. To appear in QEST'19.

[FFG+16] Marlon Fraile, Margaret Ford, Olga Gadyatskaya, Rajesh Kumar, Mariëlle Stoelinga, and Rolando Trujillo-Rasua. Using attack-defense trees to analyze threats and countermeasures in an ATM: A case study. In Jennifer Horkoff, Manfred A. Jeusfeld, and Anne Persson, editors, *The Practice of Enterprise Modeling - 9th IFIP WG 8.1. Working Conference, PoEM 2016, Skövde, Sweden, November 8-10, 2016, Proceedings*, volume 267 of *Lecture Notes in Business Information Processing*, pages 326–334. Springer, 2016. Available from: `https://doi.org/10.1007/978-3-319-48393-1_24`, `doi:10.1007/978-3-319-48393-1\_24`.

[Fre19] Frederic Byumvuhore. FEATURED: REG steps up crackdown on electricity theft , 2019. Accessed on: 2019-04-05. Available from: `https://www.newtimes.co.rw/news/featured-reg-steps-crackdown-electricity-theft`.

[FW19a] Barbara Fila and Wojciech Wideł. Attack–defense trees for abusing optical power meters: A case study and the OSEAD tool experience report. In *Graphical Security Modeling (GraMSec)*, volume 11720 of *LNCS*. Springer, 2019. (To appear).

[FW19b] Barbara Fila and Wojciech Wideł. Efficient Attack–Defense Tree Analysis using Pareto Attribute Domains. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019* [DBL19], pages 200–215. Available from: `https://doi.org/10.1109/CSF.2019.00021`, `doi:10.1109/CSF.2019.00021`.

[GBTO07] Marc Geilen, Twan Basten, Bart D. Theelen, and Ralph Otten. An Algebra of Pareto Points. *Fundam. Inform.*, 78(1):35–74, 2007. Available from: `http://content.iospress.com/articles/fundamenta-informaticae/fi78-1-03`.

[GHL+16] Olga Gadyatskaya, René Rydhof Hansen, Kim Guldstrand Larsen, Axel Legay, Mads Chr. Olesen, and Danny Bøgsted Poulsen. Modelling attack-defense trees using timed automata. In Martin Fränzle and Nicolas Markey, editors, *Formal Modeling and Analysis of Timed Systems - 14th International Conference, FORMATS 2016, Quebec, QC, Canada, August 24-26, 2016, Proceedings*, volume 9884 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2016. Available from: `https://doi.org/10.1007/978-3-319-44878-7_3`, `doi:10.1007/978-3-319-44878-7\_3`.

[GJK+16] Olga Gadyatskaya, Ravi Jhawar, Piotr Kordy, Karim Lounis, Sjouke Mauw, and Rolando Trujillo-Rasua. Attack Trees for Practical Security Assess-

ment: Ranking of Attack Scenarios with ADTool 2.0. In *QEST*, volume 9826 of *LNCS*, pages 159–162. Springer, 2016.

[GWH+18] W. Guo, J. Wang, M. He, X. Ren, Q. Wang, and W. Tian. An efficient method to transform a sat problem to a mixed integer linear programming problem. In *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, pages 1992–1996, Dec 2018. doi:10.1109/CompComm.2018.8780844.

[has16] hashcat. https://hashcat.net/hashcat/, 2016. Accessed on: 2019-03-27.

[HJL+17] René Rydhof Hansen, Peter Gjøl Jensen, Kim Guldstrand Larsen, Axel Legay, and Danny Bøgsted Poulsen. Quantitative Evaluation of Attack Defense Trees Using Stochastic Timed Automata. In Liu et al. [LMS18], pages 75–90. Available from: https://doi.org/10.1007/978-3-319-74860-3_5, doi:10.1007/978-3-319-74860-3_5.

[HKCH17] Jin B. Hong, Dong Seong Kim, Chun-Jen Chung, and Dijiang Huang. A survey on the usability and practical applications of graphical security models. *Computer Science Review*, 26:1–16, 2017. Available from: https://doi.org/10.1016/j.cosrev.2017.09.001, doi:10.1016/j.cosrev.2017.09.001.

[HMP91] Thomas A. Henzinger, Zohar Manna, and Amir Pnueli. Timed transition systems. In J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, volume 600 of *Lecture Notes in Computer Science*, pages 226–251. Springer, 1991. Available from: https://doi.org/10.1007/BFb0031995, doi:10.1007/BFb0031995.

[HMT17] Ross Horne, Sjouke Mauw, and Alwen Tiu. Semantics for specialising attack trees based on linear logic. *Fundam. Inform.*, 153(1-2):57–86, 2017. Available from: https://doi.org/10.3233/FI-2017-1531, doi:10.3233/FI-2017-1531.

[Hoo88] J.N. Hooker. A quantitative approach to logical inference. *Decision Support Systems*, 4(1):45 – 69, 1988. Available from: http://www.sciencedirect.com/science/article/pii/0167923688900978, doi:https://doi.org/10.1016/0167-9236(88)90097-8.

[HRVG81] David F. Haasl, Norman H. Roberts, William E. Veselay, and Francine F. Goldberg. Fault tree handbook. Technical report, Systems and Reliability

Research, Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Comission, 1981.

[IPHK15] Marieta Georgieva Ivanova, Christian W. Probst, René Rydhof Hansen, and Florian Kammüller. Attack tree generation by policy invalidation. In Raja Naeem Akram and Sushil Jajodia, editors, *Information Security Theory and Practice - 9th IFIP WG 11.2 International Conference, WISTP 2015 Heraklion, Crete, Greece, August 24-25, 2015 Proceedings*, volume 9311 of *Lecture Notes in Computer Science*, pages 249–259. Springer, 2015. Available from: `https://doi.org/10.1007/978-3-319-24018-3_16`, `doi:10.1007/978-3-319-24018-3\_16`.

[ISO18] Risk management – Guideliness. Standard, International Organization for Standardization, February 2018.

[JKM+15] Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer, 2015. Available from: `https://doi.org/10.1007/978-3-319-18467-8_23`, `doi:10.1007/978-3-319-18467-8\_23`.

[JW08] Aivo Jürgenson and Jan Willemson. Computing exact outcomes of multi-parameter attack trees. In Robert Meersman and Zahir Tari, editors, *On the Move to Meaningful Internet Systems: OTM 2008, OTM 2008 Confederated International Conferences, CoopIS, DOA, GADA, IS, and ODBASE 2008, Monterrey, Mexico, November 9-14, 2008, Proceedings, Part II*, volume 5332 of *Lecture Notes in Computer Science*, pages 1036–1051. Springer, 2008. Available from: `https://doi.org/10.1007/978-3-540-88873-4_8`, `doi:10.1007/978-3-540-88873-4\_8`.

[JW09] Aivo Jürgenson and Jan Willemson. Serial model for attack tree computations. In Dong Hoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, volume 5984 of *Lecture Notes in Computer Science*, pages 118–128. Springer, 2009. Available from: `https://doi.org/10.1007/978-3-642-14423-3_9`, `doi:10.1007/978-3-642-14423-3\_9`.

[KD13] Peter Kelly-Detwiler. Electricity Theft: A Bigger Issue Than You Think, 2013. Accessed: 2019-02-20. Available from: `https:`

//www.forbes.com/sites/peterdetwiler/2013/04/23/electricity-theft-a-bigger-issue-than-you-think/#5475872972ef.

[Kel76] Robert M Keller. Formal verification of parallel programs. *Communications of the ACM*, 19(7):371–384, 1976.

[Kia18] Kiana Wilburg. GPL lost US$450M in 19 years to electricity theft, poor networks, 2018. Accessed on: 2019-04-05. Available from: https://www.kaieteurnewsonline.com/2018/12/10/gpl-lost-us450m-in-19-years-to-electricity-theft-poor-networks/.

[KLM19] M. H. R. Khouzani, Zhengliang Liu, and Pasquale Malacaria. Scalable min-max multi-objective cyber-security optimisation over probabilistic attack graphs. *European Journal of Operational Research*, 278(3):894–903, 2019. Available from: https://doi.org/10.1016/j.ejor.2019.04.035, doi:10.1016/j.ejor.2019.04.035.

[KMRS14] Barbara Kordy, Sjouke Mauw, Sasa Radomirovic, and Patrick Schweitzer. Attack–defense trees. *J. Log. Comput.*, 24(1):55–87, 2014. Available from: https://doi.org/10.1093/logcom/exs029, doi:10.1093/logcom/exs029.

[KMS12] Barbara Kordy, Sjouke Mauw, and Patrick Schweitzer. Quantitative questions on attack-defense trees. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon, editors, *Information Security and Cryptology - ICISC 2012 - 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers*, volume 7839 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2012. Available from: https://doi.org/10.1007/978-3-642-37682-5_5, doi:10.1007/978-3-642-37682-5_5.

[Koz97] Dexter Kozen. *Automata and computability*. Undergraduate texts in computer science. Springer, 1997.

[KPS11] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Computational aspects of attack-defense trees. In Pascal Bouvry, Mieczyslaw A. Klopotek, Franck Leprévost, Malgorzata Marciniak, Agnieszka Mykowiecka, and Henryk Rybinski, editors, *Security and Intelligent Information Systems - International Joint Conferences, SIIS 2011, Warsaw, Poland, June 13-14, 2011, Revised Selected Papers*, volume 7053 of *Lecture Notes in Computer Science*, pages 103–116. Springer, 2011. Available from: https://doi.org/10.1007/978-3-642-25261-7_8, doi:10.1007/978-3-642-25261-7_8.

[KPS14] Barbara Kordy, Ludovic Piètre-Cambacédès, and Patrick Schweitzer. DAG-based attack and defense modeling: Don't miss the forest for

the attack trees. *Computer Science Review*, 13-14:1–38, 2014. Available from: https://doi.org/10.1016/j.cosrev.2014.07.001, doi:10.1016/j.cosrev.2014.07.001.

[KPS16] Barbara Kordy, Marc Pouly, and Patrick Schweitzer. Probabilistic reasoning with graphical security models. *Inf. Sci.*, 342:111–131, 2016. Available from: https://doi.org/10.1016/j.ins.2016.01.010, doi:10.1016/j.ins.2016.01.010.

[KPW16] Marta Kwiatkowska, David Parker, and Clemens Wiltsche. Prism-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 560–566. Springer, 2016. Available from: https://doi.org/10.1007/978-3-662-49674-9_35, doi:10.1007/978-3-662-49674-9_35.

[Kre12] Brian Krebs. FBI: Smart Meter Hacks Likely to Spread, 2012. Accessed: 2019-02-20. Available from: https://krebsonsecurity.com/2012/04/fbi-smart-meter-hacks-likely-to-spread/.

[KRS15] Rajesh Kumar, Enno Ruijters, and Mariëlle Stoelinga. Quantitative attack tree analysis via priced timed automata. In Sriram Sankaranarayanan and Enrico Vicario, editors, *Formal Modeling and Analysis of Timed Systems - 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*, volume 9268 of *Lecture Notes in Computer Science*, pages 156–171. Springer, 2015. Available from: https://doi.org/10.1007/978-3-319-22975-1_11, doi:10.1007/978-3-319-22975-1_11.

[KSHdM02] Anton J. Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. on Optimization*, 12(2):479–502, 2002.

[KSR+18] Rajesh Kumar, Stefano Schivo, Enno Ruijters, Bugra Mehmet Yildiz, David Huistra, Jacco Brandt, Arend Rensink, and Mariëlle Stoelinga. Effective analysis of attack trees: A model-driven approach. In Alessandra Russo and Andy Schürr, editors, *Fundamental Approaches to Software Engineering, 21st International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings.*, volume 10802 of *Lecture Notes in Computer Science*, pages 56–73. Springer,

2018. Available from: https://doi.org/10.1007/978-3-319-89363-1_4, doi:10.1007/978-3-319-89363-1_4.

[Kum18] Rajesh Kumar. *Truth or Dare: Quantitative security risk analysis via attack trees*. PhD thesis, University of Twente, The Netherlands, 2018.

[KW17] Barbara Kordy and Wojciech Widel. How Well Can I Secure My System? In Nadia Polikarpova and Steve Schneider, editors, *Integrated Formal Methods - 13th International Conference, IFM 2017, Turin, Italy, September 20-22, 2017, Proceedings*, volume 10510 of *Lecture Notes in Computer Science*, pages 332–347. Springer, 2017. Available from: https://doi.org/10.1007/978-3-319-66845-1_22, doi:10.1007/978-3-319-66845-1_22.

[KW18] Barbara Kordy and Wojciech Wideł. On Quantitative Analysis of Attack-Defense Trees with Repeated Labels. In Lujo Bauer and Ralf Küsters, editors, *Principles of Security and Trust - 7th International Conference, POST 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10804 of *Lecture Notes in Computer Science*, pages 325–346. Springer, 2018. Available from: https://doi.org/10.1007/978-3-319-89722-6_14, doi:10.1007/978-3-319-89722-6_14.

[LLC14] Northeast Group LLC. World Loses $89.3 Billion to Electricity Theft Annually, $58.7 Billion in Emerging Markets, 2014. Accessed: 2019-02-20. Available from: https://www.prnewswire.com/news-releases/world-loses-893-billion-to-electricity-theft-annually-587-billion-in-emerging-markets-300006515.html.

[LMS18] Peng Liu, Sjouke Mauw, and Ketil Stølen, editors. *Graphical Models for Security - 4th International Workshop, GraMSec 2017, Santa Barbara, CA, USA, August 21, 2017, Revised Selected Papers*, volume 10744 of *Lecture Notes in Computer Science*. Springer, 2018. Available from: https://doi.org/10.1007/978-3-319-74860-3, doi:10.1007/978-3-319-74860-3.

[LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997. Available from: https://doi.org/10.1007/s100090050010, doi:10.1007/s100090050010.

[LSS11] Mass Soldal Lund, Bjørnar Solhaug, and Ketil Stølen. *Model-Driven Risk Analysis - The CORAS Approach*. Springer, 2011. Available from: https://doi.org/10.1007/978-3-642-12323-8, doi:10.1007/978-3-642-12323-8.

[MB90] James Moore and Jonathan Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38:911–921, 10 1990. doi:10.1287/opre.38.5.911.

[McC10] Jeff McCullough. Deterrent and detection of smart grid meter tampering and theft of electricity, water, or gas, 2010. Accessed: 2019-02-20. Available from: https://www.elstersolutions.com/assets/downloads/WP42-1010A.pdf.

[MHM16] Steve Muller, Carlo Harpes, and Cédric Muller. Fast and optimal countermeasure selection for attack defence trees. In Jürgen Großmann, Michael Felderer, and Fredrik Seehusen, editors, *Risk Assessment and Risk-Driven Quality Assurance - 4th International Workshop, RISK 2016, Held in Conjunction with ICTSS 2016, Graz, Austria, October 18, 2016, Revised Selected Papers*, volume 10224 of *Lecture Notes in Computer Science*, pages 53–65, 2016. Available from: https://doi.org/10.1007/978-3-319-57858-3_5, doi:10.1007/978-3-319-57858-3_5.

[MO05] Sjouke Mauw and Martijn Oostdijk. Foundations of Attack Trees. In Dongho Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005, 8th International Conference, Seoul, Korea, December 1-2, 2005, Revised Selected Papers*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer, 2005. Available from: https://doi.org/10.1007/11734727_17, doi:10.1007/11734727_17.

[MP19] Heiko Mantel and Christian W. Probst. On the Meaning and Purpose of Attack Trees. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019* [DBL19], pages 184–199. Available from: https://doi.org/10.1109/CSF.2019.00020, doi:10.1109/CSF.2019.00020.

[Ms.12] Ms. Smith. FBI Warns Smart Meter Hacking May Cost Utility Companies $400 Million A Year, 2012. Accessed on: 2019-04-05. Available from: https://www.csoonline.com/article/2222111/fbi-warns-smart-meter-hacking-may-cost-utility-companies--400-million-a-year.html.

[Nat15] National Electric Sector Cybersecurity Organization Resource (NESCOR). Analysis of selected electric sector high risk failure scenarios, version 2.0, 2015. Available from: http://smartgrid.epri.com/doc/NESCOR%20Detailed%20Failure%20Scenarios%20v2.pdf.

[NPMK18] Pantaleone Nespoli, Dimitrios Papamartzivanos, Félix Gómez Mármol, and Georgios Kambourakis. Optimal countermeasures selection against

cyber attacks: A comprehensive survey on reaction frameworks. *IEEE Communications Surveys and Tutorials*, 20(2):1361–1396, 2018. Available from: https://doi.org/10.1109/COMST.2017.2781126, doi:10.1109/COMST.2017.2781126.

[NS03] Abraham Neyman and Sylvain Sorin. *Stochastic Games and Applications*, volume 570 of *NATO Science Series ASIC*. Kluwer Academic Publishers, 2003.

[Oph16] Ophcrack. http://ophcrack.sourceforge.net/, 2016. Accessed on: 2017-03-17.

[PB10] Ludovic Pièrre-Cambacédès and Marc Bouissou. Beyond attack trees: Dynamic security modeling with boolean logic driven markov processes (BDMP). In *Eighth European Dependable Computing Conference, EDCC-8 2010, Valencia, Spain, 28-30 April 2010*, pages 199–208. IEEE Computer Society, 2010. Available from: https://doi.org/10.1109/EDCC.2010.32, doi:10.1109/EDCC.2010.32.

[Pri13] Nicolas Privault. Discrete-time markov chains. In *Understanding Markov Chains: Examples and Applications*, pages 77–94. Springer, 2013.

[RDR12] Terry R. Rakes, Jason K. Deane, and Loren Paul Rees. It security planning under uncertainty for high-impact events. *Omega*, 40(1):79 – 88, 2012. Available from: http://www.sciencedirect.com/science/article/pii/S0305048311000582, doi:https://doi.org/10.1016/j.omega.2011.03.008.

[RKT12] 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, *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*, pages 1–12. IEEE Computer Society, 2012. Available from: https://doi.org/10.1109/DSN.2012.6263940, doi:10.1109/DSN.2012.6263940.

[RS15] Enno Ruijters and Mariëlle Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015. Available from: https://doi.org/10.1016/j.cosrev.2015.03.001, doi:10.1016/j.cosrev.2015.03.001.

[RSS15] Atle Refsdal, Bjørnar Solhaug, and Ketil Stølen. *Cyber-Risk Management*. Springer Briefs in Computer Science. Springer, 2015. Available

from: https://doi.org/10.1007/978-3-319-23570-7, doi:10.1007/978-3-319-23570-7.

[Saw13] Tadeusz Sawik. Selection of Optimal Countermeasure Portfolio in IT Security Planning. *Decis. Support Syst.*, 55(1):156–164, April 2013.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[SDR14] Alexander Shapiro, Darinka Dentcheva, and Andrzej Ruszczynski. *Lectures on Stochastic Programming - Modeling and Theory, Second Edition*, volume 16 of *MOS-SIAM Series on Optimization*. SIAM, 2014. Available from: http://bookstore.siam.org/mo16/.

[Ste86] K. Stecher. Evaluation of large fault-trees with repeated events using an efficient bottom-up algorithm. *IEEE Transactions on Reliability*, 35(1):51–58, April 1986. doi:10.1109/TR.1986.4335344.

[T&15] T&D World. India To Spend $21.6 Billion On Smart Grid Infrastructure By 2025, 2015. Accessed on: 2019-04-05. Available from: https://www.tdworld.com/smart-grid/india-spend-216-billion-smart-grid-infrastructure-2025.

[TR-08] Improving Common Security Risk Analysis. Technical report, Research and Technology Organisation, North Atlantic Treaty Organisation, September 2008.

[tR16] John the Ripper. https://www.openwall.com/john/, 2016. Accessed on: 2019-03-27.

[VNN14] Roberto Vigo, Flemming Nielson, and Hanne Riis Nielson. Automated generation of attack trees. In *IEEE 27th Computer Security Foundations Symposium, CSF 2014, Vienna, Austria, 19-22 July, 2014*, pages 337–350. IEEE Computer Society, 2014. Available from: https://doi.org/10.1109/CSF.2014.31, doi:10.1109/CSF.2014.31.

[WAFP19] Wojciech Wideł, Maxime Audinot, Barbara Fila, and Sophie Pinchinat. Beyond 2014: Formal methods for attack tree–based security modeling. *ACM Comput. Surv.*, 52(4):75:1–75:36, August 2019. Available from: http://doi.acm.org/10.1145/3331524, doi:10.1145/3331524.

[Web12] Don C. Weber. Optiguard: A Smart Meter Assessment Toolkit, 2012. Accessed: 2019-02-20. Available from: https://media.blackhat.com/bh-us-12/Briefings/Weber/BH_US_12_Weber_Eye_of_the_Meter_WP.pdf.

[Wei91] Jonathan D. Weiss. A system security engineering process. In *14th Annual NCSC/NIST National Computer Security Conference*, pages 572–581, 1991.

[Wid19] Wojciech Wideł. adtrees. https://github.com/wwidel/adtrees, 2019. Accessed on: 2019-08-30.

[WNJ06] Lingyu Wang, Steven Noel, and Sushil Jajodia. Minimum-cost network hardening using attack graphs. *Computer Communications*, 29(18):3812–3824, 2006. Available from: https://doi.org/10.1016/j.comcom.2006.06.018, doi:10.1016/j.comcom.2006.06.018.

[Woo93] R.Kevin Wood. Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2):1 – 18, 1993. Available from: http://www.sciencedirect.com/science/article/pii/089571779390236R, doi:https://doi.org/10.1016/0895-7177(93)90236-R.

[ZALT19] Kaiyue Zheng, Laura A. Albert, James R. Luedtke, and Eli Towle. A budgeted maximum multiple coverage model for cybersecurity planning and management, 2019. IISE Transactions, to appear. doi:10.1080/24725854.2019.1584832.

Index

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Formal Modeling and quantitative analysis of security using attack-defense trees

Nom Prénom de l'auteur : WIDEL WOJCIECH

Membres du jury :
- Monsieur EKSTEDT Mathias
- Madame FILA Barbara
- Monsieur MANTEL Heiko
- Monsieur MAUW Sjouke
- Madame PINCHINAT Sophie
- Monsieur RADOMIROVIC Sasa
- Monsieur AVOINE Gildas

Président du jury : Madame Sophie PINCHINAT

Date de la soutenance : 03 Décembre 2019

Reproduction de la these soutenue

☒ Thèse pouvant être reproduite en l'état
☐ Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 03 Décembre 2019

Signature du président de jury

Le Directeur,

M'hamed DRISSI

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

INSA RENNES

Titre : Modélisation formelle et analyse quantitative de la sécurité à l'aide d'arbres les attaques et de défense

Mots clés : analyse des risque, arbres d'attaque, arbres d'attaque et de défense

Résumé : L'analyse de risque est un processus très complexe. Elle nécessite une représentation rigoureuse et une évaluation approfondie des menaces et de leur contre-mesures. Cette thèse porte sur la modélisation formelle de la sécurité à l'aide d'arbres d'attaque et de défense. Ces derniers servent à représenter et à quantifier les attaques potentielles afin de mieux comprendre les enjeux de sécurité auxquels le système analysé peut être confronté. Ils permettent donc de guider un expert dans le choix des contre-mesures à implémenter pour sécuriser son système.

Les principales contributions de cette thèse sont les suivantes :
- L'enrichissement du modèle des arbres d'attaque et de défense permettant d'analyser des scénarios de sécurité réels. Nous avons notamment développé les fondements théoriques et les algorithmes d'évaluation quantitative pour le modèle où une action de l'attaquant peut contribuer à plusieurs attaques et où une contre-mesure peut prévenir plusieurs menaces.
- Le développement d'une méthodologie basée sur la dominance de Pareto et permettant de prendre en compte plusieurs aspects quantitatifs simultanément (e.g., coût, temps, probabilité, difficulté, etc.) lors d'une analyse de risques.
- La conception d'une technique, utilisant les méthodes de programmation linéaire, pour sélectionner un ensemble de contre-mesures optimal, en tenant compte du budget destiné à la protection du système analysé. C'est une technique générique qui peut être appliquée à plusieurs problèmes d'optimisation, par exemple, la maximisation de la couverture de surface d'attaque, ou encore la maximisation du investissement de l'attaquant.

Pour garantir leur applicabilité pratique, le modèle et les algorithmes mathématiques développés ont été implémentés dans un outil informatique à source ouverte et accès gratuit. Tous les résultats ont également été validés lors d'une étude pratique portant sur un scénario industriel d'altération de compteurs de consommation d'électricité.

Title : Formal modeling and quantitative analysis of security using attack-defense trees

Keywords : risk analysis, attack tree, attack-defense tree

Abstract : Risk analysis is a very complex process. It requires rigorous representation and in-depth assessment of threats and countermeasures. This thesis focuses on the formal modelling of security using attack and defence trees. These are used to represent and quantify potential attacks in order to better understand the security issues that the analyzed system may face. They therefore make it possible to guide an expert in the choice of countermeasures to be implemented to secure their system.

The main contributions of this thesis are as follows:
- The enrichment of the attack and defence tree model allowing the analysis of real security scenarios. In particular, we have developed the theoretical foundations and quantitative evaluation algorithms for the model where an attacker's action can contribute to several attacks and a countermeasure can prevent several threats.
- The development of a methodology based on Pareto dominance and allowing several quantitative aspects to be taken into account simultaneously (e.g., cost, time, probability, difficulty, etc.) during a risk analysis.
- The design of a technique, using linear programming methods, for selecting an optimal set of countermeasures, taking into account the budget available for protecting the analyzed system. It is a generic technique that can be applied to several optimization problems, for example, maximizing the attack surface coverage, or maximizing the attacker's investment.

To ensure their practical applicability, the model and mathematical algorithms developed were implemented in a freely available open source tool. All the results were also validated with a practical study on an industrial scenario of alteration of electricity consumption meters.