Information Sciences 342 (2016) 111–131

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Probabilistic reasoning with graphical security models[☆]

Barbara Kordy[a,b,*], Marc Pouly[c], Patrick Schweitzer[b]

[a] INSA Rennes, IRISA, 35708 Rennes, France
[b] University of Luxembourg, SnT, 1359 Luxembourg, Luxembourg
[c] Lucerne University of Applied Sciences and Arts, 6048 Horw, Switzerland

ARTICLE INFO

Article history:
Received 30 July 2014
Revised 10 November 2015
Accepted 1 January 2016
Available online 12 January 2016

Keywords:
Attack–defense trees
Attack trees
Bayesian networks
Dependent actions
Probabilistic analysis of security
Semiring theory

ABSTRACT

This work provides a computational framework for meaningful probabilistic evaluation of attack–defense scenarios involving dependent actions. We combine the graphical security modeling technique of attack–defense trees with probabilistic information expressed in terms of Bayesian networks. In order to improve the efficiency of probability computations on attack–defense trees, we make use of inference algorithms and encoding techniques from constraint reasoning. The proposed approach is illustrated on a running example and the computations are automated with the help of suitable software tools. We show that the computational routines developed in this paper form a conservative generalization of the attack–defense tree formalism defined previously. We discuss the algebraic theory underlying our framework and point out several generalizations which are possible thanks to the use of semiring theory. Finally, our results apply directly to the analysis of the industrially recognized model of attack trees.

1. Introduction

Attack–defense trees [15] extend the well-known attack tree methodology [24,36], by considering not only actions of an attacker, but also possible countermeasures of a defender. Since the augmented formalism models interactions between an attacker and a defender explicitly and is able to capture evolutionary aspects of attack–defense scenarios, it allows for a more thorough and accurate security assessment process compared to classical attack trees. The necessity for including defensive nodes into the attack tree formalism and the advantages of attack–defense trees over attack trees were discussed in [19]. A large industrial case study presented in [2] validated the usefulness of the attack–defense tree methodology for the analysis of real-world security problems. Furthermore, in [16], we have proven that the analysis using attack–defense trees interpreted in the propositional semantics, i.e., when the trees are formalized with Boolean functions, is computationally not more expensive than the analysis using attack trees. These results show that attack–defense trees have the potential to become an efficient and practical security modeling and risk assessment tool.

Quantifying probabilistic aspects of attacks is one of the most important issues in security evaluation. Decisions concerning which defensive mechanisms or countermeasures should be implemented are based on the success probability of attacks. Furthermore, estimation of probability is necessary in order to evaluate risk related measures, because they all combine frequency or probability of an attack with its impact or costs [41,42]. Hence, a fully fledged methodology for security

[☆] A preliminary version of this work has been published in [21].

[*] Corresponding author at: INSA Rennes, 20 Avenue des Buttes de Coësmes, F-35708 Rennes, France. Tel.: +33 02 99 84 7527.
E-mail addresses: barbara.kordy@irisa.fr (B. Kordy), marc.pouly@hslu.ch (M. Pouly), patrick.schweitzer@gmail.com (P. Schweitzer).

http://dx.doi.org/10.1016/j.ins.2016.01.010
0020-0255/© 2016 Elsevier Inc. All rights reserved.

analysis needs to contain a mature framework for probabilistic computations. Unfortunately, the standard bottom-up approach for quantitative analysis of attack tree-based formalisms [19,24] can *only* be used for computing probabilities under the assumption that *all considered actions are independent*. This is a very strong assumption which is unrealistic for real-life situations.

The main contribution of this paper is to provide a *complete framework for probability computations on attack–defense trees*. Our approach combines the security methodology of attack–defense trees with the probabilistic framework of Bayesian networks. This allows us to overcome the mentioned limitation of the bottom-up approach and to *perform probabilistic computations in the presence of dependent actions*. The main strengths of our design are

- In our framework, the security model and the probability model are *kept separate*. Hence, they can be created and maintained by different experts. The overlay network is only implicitly constructed when probabilistic results are calculated.
- We show that the proposed computational procedure is compatible with the propositional semantics for attack–defense trees, i.e., that propositionally equivalent attack–defense trees yield the same probability values. Furthermore, in the case of attack–defense scenarios without dependent actions, our framework coincides with the standard bottom-up approach for quantitative evaluation of attack–defense trees. These two results show that the framework developed in this paper is a *sound extension of the attack–defense tree methodology* from [19].
- Since attack–defense trees allow for modeling of interleaved attacks and defenses, the proposed approach *improves upon existing frameworks* that combine AND-OR graphs and Bayesian networks.
- Finally, as attack trees are formally a subclass of attack–defense trees, our framework *applies directly for the analysis of the former model*. Thus, the paper also provides a full-fledged analysis technique for attack trees which are widely accepted and commonly used by industry [20].

We start by describing related work in Section 2. Then, we give a brief overview of the attack–defense tree methodology in Section 3. After recalling basic concepts for Bayesian networks, we present our framework for dependent probability computations on attack–defense trees, in Section 4. Sections 5 and 6 are concerned with methods for improving the efficiency of the proposed framework. In Section 7, we elaborate on practical aspects of the introduced methodology. We conclude the article and give an outline of future work in Section 8.

2. Related work

Attack–defense trees (ADTrees) have been first proposed in [15], where their theoretical foundations, the syntax, and numerous formal semantics have been introduced. A bottom-up procedure for quantitative analysis of ADTrees has been formalized in [19]. In [18], the most popular quantitative measures for ADTrees have been classified and guidelines for their specification have been presented. The authors of [14] have established a relation between ADTrees and game theory, by proving that ADTrees under the propositional semantics are equivalent to two-player, binary, zero-sum games. Finally, to ease the use of the ADTree formalism, a free software tool, called ADTool [17], has been developed. The ADTool supports creation, sharing and management of ADTree models and automates their quantitative bottom-up analysis.

Since classical attack trees cannot model dependencies between involved actions, several improved and alternative models have been proposed to lift this limitation [5,25,28,40]. A recent survey, by Kordy et al. [20] presents a complete state of the art in the field of DAG-based approaches for modeling of attacks and defenses. The paper summarizes existing formalisms, compares their features, and proposes their taxonomy. In the remainder of this section, we concentrate on the most prominent, existing approaches that combine AND-OR graphs with Bayesian networks and make the probabilistic evaluation of security scenarios involving dependent actions possible.

Qin and Lee are one of the pioneers in applying Bayesian networks for security analysis [33]. They propose a conversion of regular attack trees into Bayesian networks, in order to make use of probabilistic inference techniques to evaluate the likelihood of attack goals and predict potential upcoming attacks. Edges representing disjunctive refinements in the tree are also present in the corresponding Bayesian network, because they represent cause-consequence relations between components. Contrary to our interpretation, a conjunction in attack trees is assumed to have an explicit or implicit order in which the actions have to be executed. This allows to convert conjunctions into a directed path in the Bayesian network, starting from the first child, according to the given order, and ending with the parent node. The construction from [33] implies that the Bayesian network and the attack tree contain the same set of nodes. Furthermore the Bayesian network models cause-consequence relationships that correspond to the child-parent connections in the underlying attack tree. In our case, the Bayesian network depicts *additional* dependencies that represent how different basic actions are influenced by each other.

In 2008, Frigault and Wang advance a different model, called *Bayesian attack graphs*, which is used to calculate general security metrics regarding information system networks [8]. They construct a Bayesian network starting from an attack graph model which depicts how multiple vulnerabilities may be combined in an attack. The resulting directed acyclic graph contains all nodes of the original attack graph. The nodes are then populated with probability values based on the Common Vulnerability Scoring System (CVSS) [26]. These values and the conditional relationships represented by the edges of the network are encoded in the conditional probability tables assigned to each node. In a follow-up paper the approach is extended with a dynamic dimension by interpreting multiple copies of the Bayesian network as time slices [9]. Attack graphs may contain cycles, which need to be eliminated in order to construct the corresponding Bayesian network. An intricate procedure for such an elimination is described in [44]. Like in [33], the model of [8] assumes an explicit order on the children of conjunctive nodes.

In 2012, Poolsappasit et al. revisited the framework of Bayesian attack graphs to be able to deal with asset identification, system vulnerability and connectivity analysis, as well as mitigation strategies [29]. Their model extends the work of Frigault and Wang by assigning a disjunctive or a conjunctive identifier to every node that has at least two incoming edges. In addition to the conditional probability tables, that represent the probability with which an attack takes place, they consider edge probabilities expressing how likely a present attack succeeds. Due to these modifications, their probability computations are able to handle conjunctions without an implicit order. As an extension, Poolsappasit et al. augment Bayesian attack graphs with additional nodes and values representing defenses. The augmented structure allows them to solve the multiobjective optimization problem of how to select optimal defenses [29]. Even though this model is similar to ours, it does not cover interleaved attacks and defenses.

Yet another approach that makes use of Bayesian networks for security analysis was described by Sommestad et al. [38,39]. It transforms defense trees [3] (an extension of attack trees with defenses attached to leaf nodes) into extended influence diagrams [22] (an extension of Bayesian networks with conjunctive and disjunctive nodes as well as countermeasures). The relationships between the nodes are encoded in conditional probability tables assigned to each node. The authors state that with this setup, “Bayesian inference can be used to derive values”, however they do not provide detailed computation algorithms. Our paper specifies how the necessary computational steps could be performed.

Contrary to our design, none of the above approaches separates the logical structure (conjunctions and disjunctions) from the probabilistic structure. One advantage of our approach is that we are not transforming one model into another, but we are using them modularly. Merging the two methodologies is only implicitly done during fusion. Unlike our model, all related approaches assume a one-to-one correspondence between the nodes in the original graph and the Bayesian network. Since in our framework the Bayesian network concerns only basic actions, its size is much smaller than the size of Bayesian networks used by the approaches described in this section. Finally, none of the related work treats evaluation of probabilities on equivalent representations of scenarios. This, however, is necessary for ADTrees, whose semantics are based on equivalence relations. Such a setting accommodates possible tree transformations and allows us to work with equivalence classes instead of particular representations of attack–defense scenarios.

3. The attack–defense tree methodology

This section provides background knowledge about attack–defense trees, which is necessary to understand the framework developed in this paper. For a more detailed description, we refer to [19] and [16].

3.1. ADTrees

Attack–defense trees (ADTrees) have been introduced to illustrate and analyze security scenarios that involve two opposing players: an attacker and a defender. Consequently, it is possible to model interleaved attacker and defender actions qualitatively and quantitatively. The root of an ADTree represents the main goal of one of the players called *proponent*. The other player is called *opponent*. When the root is an attack node, the proponent is an attacker and the opponent is a defender. Conversely, when the root is a defense node, the proponent is a defender and the opponent is an attacker. In ADTrees, both types of nodes, attacks and defenses, can be conjunctively as well as disjunctively refined. The meaning of a refinement is the following:

- A goal represented by a conjunctively refined node is reached when *all the subgoals* depicted by its child nodes are reached.
- A goal represented by a disjunctively refined node is reached when *at least one of the subgoals* depicted by its child nodes is reached.

The refinement operation is applied until atomic subgoals are obtained. Subgoals are considered to be atomic if they can be easily understood and quantified. Atomic subgoals are represented by the nodes which do not have any children of the same type. The attack–defense tree formalism allows for each node to have one child of the opposite type. Children of the opposite type represent countermeasures. These countermeasures can be refined and countered again. In ADTrees, attack nodes are modeled by circles, defense nodes by rectangles. A conjunctive refinement can be distinguished from a disjunctive refinement through the arc that connects the refining children of the conjunctive node. Countermeasures are connected to the actions they counteract by a dotted line.

Example 1. Consider a scenario in which an attacker wants to infect a computer with a virus. In order to do this, the attacker needs to ensure that the virus file is accessible from the targeted computer and that it is executed. There are two possibilities to make the file accessible: an attacker can send the virus in an e-mail attachment or distribute an infected USB stick to the computer user. The computer user, on his part, can protect himself against a virus with an anti-virus program. For the anti-virus to be effective, it needs to be installed and it needs to be running. A resourceful attacker, in turn, could attack the anti-virus by using a fake version of an anti-virus program, that disables the real anti-virus from running and only pretends that it is working. Fig. 1 depicts the described attack–defense scenario using an ADTree. In this tree, the proponent is the attacker and the opponent is the defender.

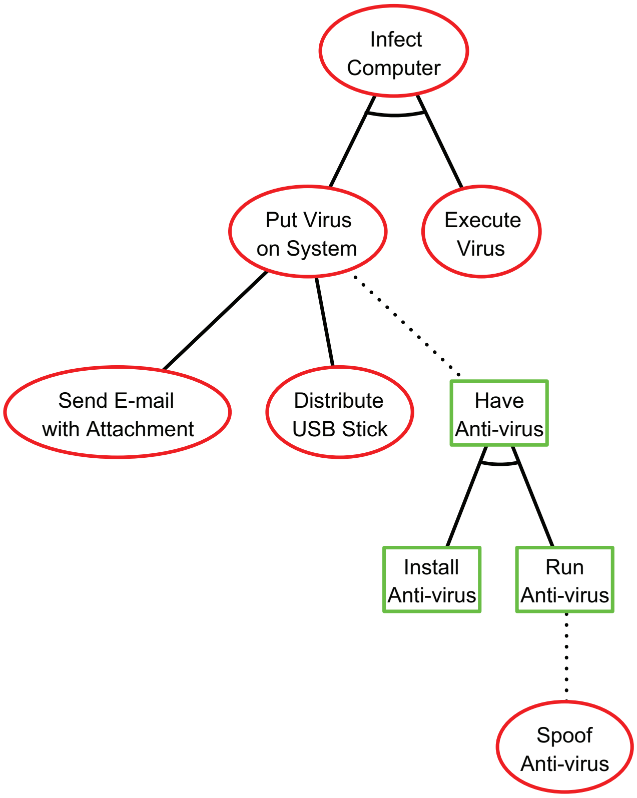


Fig. 1. ADTree for infecting a computer.

The attack–defense scenario described above is used as the running example in this paper. Its main role is to illustrate how the introduced methodology works. We purposely keep the example simple (and incomplete) in order to easily highlight technical aspects of our framework rather than exhaustively describe a complex scenario.

Remark 1. Since the root of the ADTree in Fig. 1 represents an attack goal (infecting a computer), the paper is concerned with the *probability of attacking a system*. In the case of an ADTree having a defensive root node, we would talk about the *probability of defending a system*.

3.2. ADTerms

A term-based representation of ADTrees, called *attack–defense terms*[1] (ADTerms), has been developed to make possible the display of large ADTrees and allow for formal analysis of attack–defense scenarios [19]. We briefly recall the construction of the ADTerms below.

Let $\{p, o\}$ be a set of types representing the proponent and the opponent. Given a player $s \in \{p, o\}$, we write $\bar{s}$ to denote the opposite player. By $\mathbb{B}$ we denote a set of constants called *basic actions*. These constants constitute a formal representation of atomic subgoals from ADTrees. The set of basic actions $\mathbb{B}$ is partitioned into basic actions of the proponent and basic actions of the opponent.

Example 2. For the ADTree from Example 1, the basic actions of the proponent, i.e., of the attacker, are

SE	–	"Send E-mail with Attachment"
DU	–	"Distribute USB Stick"
SA	–	"Spoof Anti-virus"
EV	–	"Execute Virus".

The basic actions of the opponent, i.e., of the defender, are

IA	–	"Install Anti-virus"
RA	–	"Run Anti-virus".

[1] Equivalence between ADTrees and ADTerms has been discussed in [19].

The typed operators $\vee^{\mathrm{p}}, \wedge^{\mathrm{p}}$ are employed for modeling disjunctive and conjunctive refinements of the proponent's actions and the corresponding operators $\vee^{\mathrm{o}}, \wedge^{\mathrm{o}}$ for refining the opponent's actions. Moreover, to connect actions of one player with counteractions of the other player, the countermeasure operators c^{p} and c^{o} are used.

Definition 1. Attack–defense terms (ADTerms) are finite, typed, ground terms recursively constructed from $\mathbb{B}$ using the typed operators $\vee^s, \wedge^s, \mathrm{c}^s$, for $s \in \{\mathrm{p}, \mathrm{o}\}$, according to the following grammar:

$$\mathbb{T}^s : \mathbb{B}^s \mid \vee^s(\mathbb{T}^s, \dots, \mathbb{T}^s) \mid \wedge^s(\mathbb{T}^s, \dots, \mathbb{T}^s) \mid \mathrm{c}^s(\mathbb{T}^s, \mathbb{T}^{\bar{s}}).$$

The set of all ADTerms is denoted by $\mathbb{T}$, i.e., we have $\mathbb{T} = \mathbb{T}^{\mathrm{p}} \cup \mathbb{T}^{\mathrm{o}}$.

Example 3. The ADTerm corresponding to the ADTree from Fig. 1 is

$$t = \wedge^{\mathrm{p}}(\mathrm{c}^{\mathrm{p}}(\vee^{\mathrm{p}}(\mathrm{SE}, \mathrm{DU}), \wedge^{\mathrm{o}}(\mathrm{IA}, \mathrm{c}^{\mathrm{o}}(\mathrm{RA}, \mathrm{SA}))), \mathrm{EV}). \tag{1}$$

3.3. The propositional semantics for ADTerms

Several distinct semantics for ADTerms have been introduced in [19]. They allow us to interpret ADTerms in different formal frameworks, which provides a broad spectrum of analysis methods. In this paper, we consider the propositional semantics for ADTerms, whose construction we recall below.

By r, we denote a countable set of propositional variables. A *configuration* with finite domain $u \subseteq r$ is a function $\mathbf{x}$: $u \to \{0, 1\}$ that associates a value $\mathbf{x}(X) \in \{0, 1\}$ with every variable $X \in u$. Thus, a configuration $\mathbf{x} \in \{0, 1\}^u$ represents an assignment of Boolean values to the variables in u.

Definition 2. A Boolean function f with domain u is a function f: $\{0, 1\}^u \to \{0, 1\}$ that assigns a value $f(\mathbf{x}) \in \{0, 1\}$ to each configuration $\mathbf{x} \in \{0, 1\}^u$.

Given a configuration $\mathbf{x}$ with domain $u \subseteq r$, we denote by $\mathbf{x}^{\downarrow w}$ the projection of $\mathbf{x}$ to a subset $w \subseteq u$. Let f and g be two Boolean functions with domains u and w, respectively. The *disjunction* $(f \vee g)$ and the *conjunction* $(f \wedge g)$ of f and g are Boolean functions with domain $u \cup w$, defined for every $\mathbf{x} \in \{0, 1\}^{u \cup w}$ by

$$(f \vee g)(\mathbf{x}) = \max\{f(\mathbf{x}^{\downarrow u}), g(\mathbf{x}^{\downarrow w})\}, \qquad (f \wedge g)(\mathbf{x}) = f(\mathbf{x}^{\downarrow u}) \times g(\mathbf{x}^{\downarrow w}).$$

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

$$(\neg f)(\mathbf{x}) = 1 - f(\mathbf{x}).$$

Remark 2. We deliberately define operations on Boolean functions with the help of the algebraic structure $\langle\{0, 1\}, \max, \times\rangle$, instead of $\langle\{\text{true}, \text{false}\}, \vee, \wedge\rangle$, because in this paper we develop a framework combining Boolean functions with probabilistic computations over the structures $\langle\mathbb{R}, +, \times\rangle$ and $\langle[0, 1], \max, \times\rangle$. Such a design choice allows us to later employ some well known results from semiring theory in order to improve the computational efficiency of the introduced framework, as presented in Sections 5 and 6.

Now, we explain how the propositional semantics associates ADTerms with Boolean functions. First, for every basic action $B \in \mathbb{B}$, a propositional variable $X_B \in r$ is constructed. We assume that for $B, B' \in \mathbb{B}$, $B \neq B' \Rightarrow X_B \neq X_{B'}$. Next, a Boolean function f_t is associated with every ADTerm t, as follows:

- If $t = B \in \mathbb{B}$, then $f_B : \{0, 1\}^{\{X_B\}} \to \{0, 1\}$ is defined as $f_B(X_B) = X_B$. In other words, the Boolean function associated with B is the identity function. Thus, we often abuse notation and use X_B instead of f_B.
- Boolean functions associated with composed ADTerms are then defined recursively[2] as follows. For $s \in \{\mathrm{p}, \mathrm{o}\}$, $k > 0$,

$$f_{\vee^s(t_1,\dots,t_k)} = \bigvee_{i=1}^{k} f_{t_i}, \qquad f_{\wedge^s(t_1,\dots,t_k)} = \bigwedge_{i=1}^{k} f_{t_i}, \qquad f_{\mathrm{c}^s(t_1,t_2)} = f_{t_1} \wedge \neg f_{t_2}.$$

Example 4. Applying the introduced recursive construction to the ADTerm t from Example 3 results in the following Boolean function:

$$f_t \equiv ((X_{\mathrm{SE}} \vee X_{\mathrm{DU}}) \wedge \neg(X_{\mathrm{IA}} \wedge (X_{\mathrm{RA}} \wedge \neg X_{\mathrm{SA}}))) \wedge X_{\mathrm{EV}}. \tag{2}$$

Given an ADTerm t, we denote by var_t the domain of the Boolean function f_t. In other words, var_t is the set of propositional variables corresponding to the basic actions involved in t. A configuration $\mathbf{x} \in \{0, 1\}^{\mathrm{var}_t}$ represents which actions succeed (the corresponding variables are set to 1) and which do not succeed (the corresponding variables are set to 0). Following our terminology convention from Remark 1, if $f_t(\mathbf{x}) = 1$, then we say that configuration $\mathbf{x}$ is an *attack with respect to t*.

[2] Here, $\bigvee$ and $\bigwedge$ stand for extensions of conjunction and disjunction of two Boolean functions to any finite number of Boolean functions. They are well-defined by associativity of max and ×.

The purpose of a semantics for ADTerms is to define which ADTrees represent the same scenario. In the propositional semantics, this is achieved by using the notion of equivalent Boolean functions. Let $u, w \subseteq \{X_B \mid B \in \mathbb{B}\}$ be two sets of propositional variables. Two Boolean functions f and g, with respective domains u and w, are said to be *equivalent* (denoted by $f \equiv g$) if and only if, for every $\mathbf{x} \in \{0, 1\}^{u \cup w}$, we have $f(\mathbf{x}^{\downarrow u}) = g(\mathbf{x}^{\downarrow w})$.

Definition 3. We say that ADTerms t and t' are equivalent in the propositional semantics if $f_t \equiv f_{t'}$.

In other words, the propositional semantics partitions the set of ADTerms on equivalence classes, such that ADTerms t and t' belong to the same equivalence class if $\forall \mathbf{x} \in \{0, 1\}^r$, configuration $\mathbf{x}^{\downarrow \mathrm{var}_t}$ is an attack with respect to t if and only if configuration $\mathbf{x}^{\downarrow \mathrm{var}_{t'}}$ is an attack with respect to t'.

4. Probabilistic evaluation of scenarios with dependencies

The attack–defense tree formalism can be used not only to represent how to attack a system, but also to quantify related security parameters, such as cost, time, or probability. The most often used computational procedure for quantitative assessment of ADTrees relies on a bottom-up algorithm. It has originally been proposed for attack trees in [24,36] and has then been extended to ADTrees in [19]. In this approach, values are assigned to the nodes representing atomic subgoals in an ADTree. Then, the bottom-up algorithm is used to determine the values of the remaining nodes as a function of the values of their child nodes. The computation stops when the value for the root node has been found. Since the value of a node only depends on the *values* of its children, and *not on their meaning*, the bottom-up procedure cannot take dependencies between actions into account. Thus, this standard technique for quantitative analysis implicitly assumes that all actions are independent. Such an assumption is unrealistic, especially when we are interested in evaluating parameters such as the probability of attacking a system. For instance, in the scenario described in Example 1, the probability that the defender actually runs an anti-virus program, if it is installed, would be quite high, but would be zero, if the anti-virus is not installed.

In order to compute the probability of attacking a system, while taking dependencies between involved actions into account, we propose a framework which combines attack–defense trees with Bayesian networks.

4.1. Bayesian networks – background

A *Bayesian network* [27] is a graphical representation of a *joint probability distribution* over a finite set of variables with finite domains. The network itself is a directed, acyclic graph that reflects the conditional interdependencies between the variables associated with the nodes of the network. A directed edge from the node associated with variable X_1 to the node associated with variable X_2 means that X_2 stochastically depends on X_1. Each node contains a *conditional probability table* that quantifies the influence between the variables. The joint probability distribution p of a Bayesian network over $\{X_1, \ldots, X_n\}$ is given by

$$p(X_1, \ldots, X_n) = \prod_{i=1}^{n} p(X_i \mid \mathrm{par}(X_i)). \tag{3}$$

Here, $\mathrm{par}(X_i)$ denotes the parents of X_i in the network, defined as all nodes that have an outgoing edge that points into X_i. If the set $\mathrm{par}(X_i)$ is empty, the conditional probability becomes an ordinary probability distribution. Nodes (or subgraphs) of the Bayesian network that are unconnected represent stochastically independent (sets of) variables.

In Section 4.3, we develop a framework for computing the success probability of attack–defense scenarios. This framework makes use of a Bayesian network that complements an ADTree with additional information concerning stochastic dependencies between the involved actions. The construction of such a Bayesian network is explained in the next section.

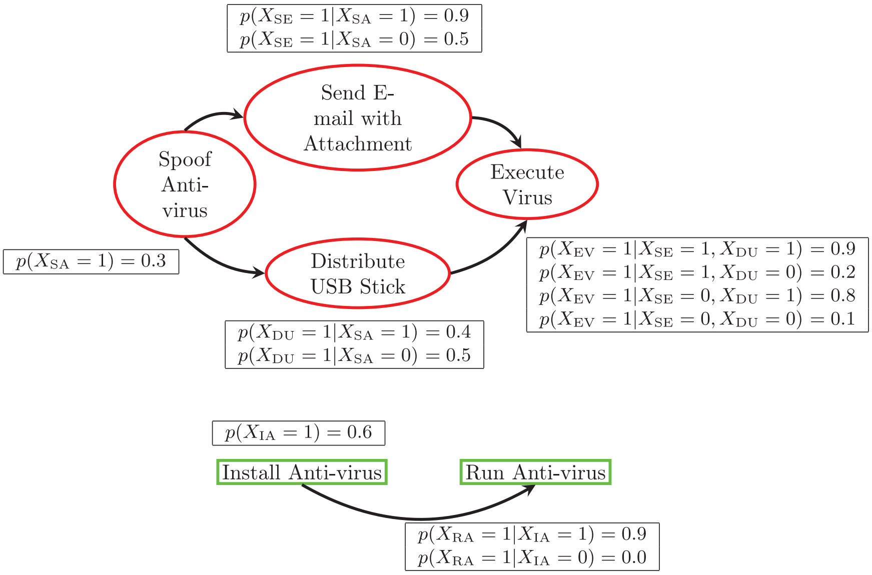
4.2. Bayesian network associated with an ADTerm

In the ADTree methodology, refined nodes do not contain any additional information, other than how the information represented by their children is composed (conjunctively or disjunctively). This means that refined nodes *do not* depict any additional actions. This is why, when constructing a Bayesian network for an ADTerm, we take only basic actions into account.

Definition 4. Let t be an ADTerm. A Bayesian network associated with t, denoted by BN_t, is a Bayesian network over the set of propositional variables var_t, such that there exists a directed edge from $X_A \in \mathrm{var}_t$ to $X_B \in \mathrm{var}_t$ if and only if action B stochastically depends on action A.

The structure of the Bayesian network BN_t is usually constructed manually. This process can however be supported by numerous existing approaches for constructing Bayesian networks [43]. Example 5 illustrates a Bayesian network associated with the ADTerm from our running example.

Example 5. A Bayesian network BN_t associated with ADTerm t given by Eq. (1) is shown in Fig. 2. It depicts stochastic dependencies that link the basic actions involved in infecting a computer with a virus. An attacker who manages to install a fake anti-virus on a victim's computer ($X_{\mathrm{SA}} = 1$) most probably presents computer skills allowing him to successfully



$p(X_{SE}=1|X_{SA}=1)=0.9$
$p(X_{SE}=1|X_{SA}=0)=0.5$
Send E-mail with Attachment
Spoof Anti-virus
Execute Virus
$p(X_{SA}=1)=0.3$
Distribute USB Stick
$p(X_{EV}=1|X_{SE}=1,X_{DU}=1)=0.9$
$p(X_{EV}=1|X_{SE}=1,X_{DU}=0)=0.2$
$p(X_{EV}=1|X_{SE}=0,X_{DU}=1)=0.8$
$p(X_{EV}=1|X_{SE}=0,X_{DU}=0)=0.1$
$p(X_{DU}=1|X_{SA}=1)=0.4$
$p(X_{DU}=1|X_{SA}=0)=0.5$
$p(X_{IA}=1)=0.6$
Install Anti-virus
Run Anti-virus
$p(X_{RA}=1|X_{IA}=1)=0.9$
$p(X_{RA}=1|X_{IA}=0)=0.0$

Fig. 2. Bayesian network BN_t associated with ADTerm t from Example 3.

distribute an e-mail with malicious attachment ($X_{SE}=1$). However, he might not possess strong social engineering skills helpful in distributing infected USB sticks ($X_{DU}=1$). Thus,

$$p(X_{SE}=1|X_{SA}=1) > p(X_{SE}=1|X_{SA}=0)$$

and

$$p(X_{DU}=1|X_{SA}=1) < p(X_{DU}=1|X_{SA}=0).$$

Furthermore, a user who has received an infected e-mail attachment or a USB stick is more likely to execute the virus ($X_{EV}=1$)

$$p(X_{EV}=1|X_{SE}=1,X_{DU}=1) > p(X_{EV}=1|X_{SE}=1,X_{DU}=0)$$

and

$$p(X_{EV}=1|X_{SE}=1,X_{DU}=1) > p(X_{EV}=1|X_{SE}=0,X_{DU}=1).$$

Nevertheless, if he has not received the virus via these two channels, he might still obtain the virus by different means, which explains the non-null probability

$$p(X_{EV}=1|X_{SE}=0,X_{DU}=0)=0.1.$$

Finally, a defender will be able to successfully run an anti-virus program ($X_{RA}=1$) only if the latter has previously been installed with success ($X_{IA}=1$), i.e.,

$$p(X_{RA}=1|X_{IA}=0)=0.0.$$

The corresponding joint probability distributions are, for the attacker

$$p(X_{EV},X_{SE},X_{DU},X_{SA}) = p(X_{EV}|X_{SE},X_{DU})\times p(X_{SE}|X_{SA})\times p(X_{DU}|X_{SA})\times p(X_{SA}) \tag{4}$$

and for the defender

$$p(X_{RA},X_{IA}) = p(X_{RA}|X_{IA})\times p(X_{IA}). \tag{5}$$

Since in our running example variables associated with basic actions of the attacker and those associated with basic actions of the defender are stochastically independent, the joint probability distribution for BN_t is obtained by multiplying expressions (4) and (5)

$$p(X_{EV},X_{SE},X_{DU},X_{SA},X_{RA},X_{IA}) = p(X_{EV}|X_{SE},X_{DU})\times p(X_{SE}|X_{SA})\times p(X_{DU}|X_{SA})\times p(X_{SA})\times p(X_{RA}|X_{IA})\times p(X_{IA}). \tag{6}$$

The conditional probability tables used in Fig. 2 have been created on an intuitive basis. The accuracy of the input values, as well as the actual methods for their estimation are a research topic in itself and are outside the scope of this submission. In the rest of the paper, we assume that the structure of the Bayesian network BN_t and the corresponding conditional probability tables have been constructed and are available. In Section 7.2, we refer to some existing methods that can be helpful in creating Bayesian networks and estimating their input values.

4.3. Probabilistic computations

We now present our framework for probability computations on an ADTerm t, taking the dependencies between the involved actions into account. Our computation makes use of the Boolean function f_t and the Bayesian network BN_t.

Given configuration $\mathbf{x} \in \{0, 1\}^{\mathrm{var}_t}$, we define

$$\psi_t(\mathbf{x}) = f_t(\mathbf{x}) \times p(\mathbf{x}), \tag{7}$$

where p is the joint probability distribution of BN_t. If $\mathbf{x}$ is an attack with respect to t, then $f_t(\mathbf{x}) = 1$ and $\psi_t(\mathbf{x})$ returns the probability value for $\mathbf{x}$ from the Bayesian network, representing the success probability of attack $\mathbf{x}$. If $\mathbf{x}$ is not an attack with respect to t, then $f_t(\mathbf{x}) = 0$ and thus $\psi_t(\mathbf{x}) = 0$.

Example 6. Let us assume that we are interested in the success probability of the situation where the attacker installs a virus file on the system by sending an e-mail with attachment ($X_{\mathrm{SE}} = 1$ and $X_{\mathrm{DU}} = 0$), executes the virus file ($X_{\mathrm{EV}} = 1$), but does not use a fake anti-virus program ($X_{\mathrm{SA}} = 0$). The defender, in turn, installs a real anti-virus ($X_{\mathrm{IA}} = 1$) which however is not running ($X_{\mathrm{RA}} = 0$). The corresponding configuration $\mathbf{x} = (X_{\mathrm{EV}} = 1, X_{\mathrm{SE}} = 1, X_{\mathrm{DU}} = 0, X_{\mathrm{SA}} = 0, X_{\mathrm{RA}} = 0, X_{\mathrm{IA}} = 1)$ is an attack, because

$$\begin{aligned} f_t(\mathbf{x}) &\overset{(2)}{=} (((X_{\mathrm{SE}} \vee X_{\mathrm{DU}}) \wedge \neg(X_{\mathrm{IA}} \wedge (X_{\mathrm{RA}} \wedge \neg X_{\mathrm{SA}}))) \wedge X_{\mathrm{EV}})(\mathbf{x}) \\ &= \max\{1, 0\} \times (1 - (1 \times (0 \times (1 - 0)))) \times 1 \\ &= 1 \times (1 - 0) \times 1 = 1. \end{aligned}$$

By instantiating formula (6) with values from Fig. 2, we obtain that this attack will be successfully executed with the probability

$$\begin{aligned} \psi_t(\mathbf{x}) &= f_t(\mathbf{x}) \times p(\mathbf{x}) \\ &= p(X_{\mathrm{EV}} = 1 | X_{\mathrm{SE}} = 1, X_{\mathrm{DU}} = 0) \times p(X_{\mathrm{SE}} = 1 | X_{\mathrm{SA}} = 0) \\ &\quad \times \; p(X_{\mathrm{DU}} = 0 | X_{\mathrm{SA}} = 0) \times p(X_{\mathrm{SA}} = 0) \times p(X_{\mathrm{RA}} = 0 | X_{\mathrm{IA}} = 1) \times p(X_{\mathrm{IA}} = 1) \\ &= 0.2 \times 0.5 \times (1 - 0.5) \times (1 - 0.3) \times (1 - 0.9) \times 0.6 = 0.0021. \end{aligned}$$

Next, assume that we are not interested in calculating the probability of successfully executing a specific set of basic actions, but more generally in the success probability of attacking a system according to the scenario represented with ADTerm t. This corresponds to the sum of the probabilities of all possible attacks with respect to t. We thus have

$$P(t) = \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} \psi_t(\mathbf{x}) \overset{(7)}{=} \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} f_t(\mathbf{x}) \times p(\mathbf{x}). \tag{8}$$

In the rest of this paper, we refer to the value $P(t)$ as *the probability related to ADTerm t*.

Example 7. In our running example, there are 21 possible attacks. They are listed in Table 1. Making use of Eq. (8) and the probability values from the last column of Table 1, we obtain

$$P(t) = \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} f_t(\mathbf{x}) \times p(\mathbf{x}) = \sum_{i=1}^{21} p(\mathbf{x}_i) = 0.29215.$$

We note that the computation presented in this example is exponential in the number of involved variables. It is therefore unsuitable for the analysis of large ADTrees. Later in this paper, we develop techniques to overcome this problem.

Finally, the success probability of the most probable attack with respect to term t is computed as

$$P_{\max}(t) = \max_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} \psi_t(\mathbf{x}) \overset{(7)}{=} \max_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} f_t(\mathbf{x}) \times p(\mathbf{x}). \tag{9}$$

Example 8. By investigating the values from the last column of Table 1, we conclude that, for our running example, the attack represented by configuration $\mathbf{x}_1$ would succeed with the highest probability and that the corresponding probability is higher than 6%. Attack $\mathbf{x}_1$ consists of sending a malicious e-mail, distributing an infected USB stick, and executing the virus file. In this case, the infected computer is not protected by an anti-virus program and a fake anti-virus is not used.

We point out that formula (9) only computes the probability value of the most probable attack with respect to t, but it is not returning the corresponding configuration. In Section 6, we discuss the fusion algorithm which can be used to obtain both, the probability value and the corresponding configuration.

We know that the number of configurations grows exponentially with the number of involved basic actions. For large systems, it is therefore not feasible to explicitly write (i.e., enumerate all values of) the Boolean function and the joint probability distribution associated with an ADTerm. In Section 5, we introduce a factorized representation for Boolean functions, which allows us to increase the efficiency of the computation of $P(t)$ and $P_{\max}(t)$.

In this section, we have focused on the success probability. However, if we modify the interpretation that we use for the propositional variables X_B, for $B \in \mathbb{B}$, the introduced methodology also applies to the computation of the occurrence

Table 1
Possible attacks according to ADTerm t given by Eq. (1) and the corresponding success probabilities.

$\mathbf{x}$	X_{SE}	X_{DU}	X_{EV}	X_{IA}	X_{RA}	X_{SA}	$p(\mathbf{x})$
$\mathbf{x}_1$	1	1	1	0	0	0	0.063
$\mathbf{x}_2$	1	1	1	0	0	1	0.03888
$\mathbf{x}_3$	1	1	1	0	1	0	0
$\mathbf{x}_4$	1	1	1	0	1	1	0
$\mathbf{x}_5$	1	1	1	1	0	0	0.00945
$\mathbf{x}_6$	1	1	1	1	0	1	0.005832
$\mathbf{x}_7$	1	1	1	1	1	1	0.052488
$\mathbf{x}_8$	1	0	1	0	0	0	0.014
$\mathbf{x}_9$	1	0	1	0	0	1	0.01296
$\mathbf{x}_{10}$	1	0	1	0	1	0	0
$\mathbf{x}_{11}$	1	0	1	0	1	1	0
$\mathbf{x}_{12}$	1	0	1	1	0	0	0.0021
$\mathbf{x}_{13}$	1	0	1	1	0	1	0.001944
$\mathbf{x}_{14}$	1	0	1	1	1	1	0.017496
$\mathbf{x}_{15}$	0	1	1	0	0	0	0.056
$\mathbf{x}_{16}$	0	1	1	0	0	1	0.00384
$\mathbf{x}_{17}$	0	1	1	0	1	0	0
$\mathbf{x}_{18}$	0	1	1	0	1	1	0
$\mathbf{x}_{19}$	0	1	1	1	0	0	0.0084
$\mathbf{x}_{20}$	0	1	1	1	0	1	0.000576
$\mathbf{x}_{21}$	0	1	1	1	1	1	0.005184

probability. If by $X_B = 1$, we mean that action B is executed and by $X_B = 0$ that it is not executed, then formula (7) expresses the probability that a specific attack vector is attempted, formula (9) represents the probability of the most probable attack vector, and formula (8) computes the probability of an attempt to attack a system according to the scenario represented with an ADTree t.

4.4. Compliance of the proposed framework with the ADTree methodology

We now show that the framework proposed in this paper is a sound extension of the ADTree methodology. We first make a link to the *compatibility* notion defined in [19] and prove that propositionally equivalent ADTerms yield the same probability values. Then, we compare our novel computation method with the standard bottom-up evaluation procedure for ADTrees.

Definition 5. We say that a computational procedure for ADTerms is compatible with a semantics for ADTerms, if and only if, computations performed on equivalent ADTerms result in the same numerical value.

The following theorem shows that the computational framework developed in this paper is compatible with the propositional semantics.

Theorem 1. *Let t and t' be two ADTerms and BN_t and $\mathrm{BN}_{t'}$ be the corresponding Bayesian networks with the joint probability distributions p and p', respectively. If t and t' are equivalent in the propositional semantics and the marginalized probability distributions over the set of common variables of BN_t and $\mathrm{BN}_{t'}$ are equal, then $P(t) = P(t')$ as well as $P_{\max}(t) = P_{\max}(t')$.*

Proof. First, we remark that the assumption on the marginalized probability distributions can be expressed as $\forall \mathbf{z} \in \{0,1\}^{\mathrm{var}_t \cap \mathrm{var}_{t'}}$,

$$\sum_{\mathbf{u} \in \mathrm{var}_t \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'})} p(\mathbf{z}, \mathbf{u}) = \sum_{\mathbf{u}' \in \mathrm{var}_{t'} \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'})} p'(\mathbf{z}, \mathbf{u}'). \tag{10}$$

Since $f_t \equiv f_{t'}$, we know that variables from $\mathrm{var}_t \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'})$ do not influence the value of f_t. Thus, for every $\mathbf{z} \in \{0,1\}^{\mathrm{var}_t \cap \mathrm{var}_{t'}}$ and every $\mathbf{u} \in \{0,1\}^{\mathrm{var}_t \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'})}$, we have $f_t(\mathbf{z}, \mathbf{u}) = f_t(\mathbf{z}, \mathbf{0})$, where $\mathbf{0}$ is a configuration assigning 0 to every variable from its domain[3]. An analogous property holds for $f_{t'}$ and we have $f_t(\mathbf{z}, \mathbf{0}) = f_{t'}(\mathbf{z}, \mathbf{0})$, for all $\mathbf{z} \in \{0,1\}^{\mathrm{var}_t \cap \mathrm{var}_{t'}}$. The domains of the two configurations $\mathbf{0}$ in the preceding equation have been omitted for the sake of readability. We obtain

$$\begin{aligned} P(t) &\overset{(8)}{=} \sum_{\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}} (f_t(\mathbf{x}) \times p(\mathbf{x})) \\ &= \sum_{\mathbf{z} \in \{0,1\}^{\mathrm{var}_t \cap \mathrm{var}_{t'}}} \sum_{\mathbf{u} \in \{0,1\}^{\mathrm{var}_t \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'})}} (f_t(\mathbf{z}, \mathbf{u}) \times p(\mathbf{z}, \mathbf{u})) \end{aligned}$$

[3] Any configuration can be used here instead of $\mathbf{0}$. We use $\mathbf{0}$ to ease the presentation.

$$
\begin{aligned}
&= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}_t\cap\mathrm{var}_{t'}}} \left(f_t(\mathbf{z},\mathbf{0}) \times \sum_{\mathbf{u}\in\{0,1\}^{\mathrm{var}_t\setminus(\mathrm{var}_t\cap\mathrm{var}_{t'})}} p(\mathbf{z},\mathbf{u}) \right)\\
&\overset{f_t\equiv f_{t'},(10)}{=} \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}_t\cap\mathrm{var}_{t'}}} \left(f_{t'}(\mathbf{z},\mathbf{0}) \times \sum_{\mathbf{u}'\in\{0,1\}^{\mathrm{var}_{t'}\setminus(\mathrm{var}_t\cap\mathrm{var}_{t'})}} p'(\mathbf{z},\mathbf{u}') \right)\\
&= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}_t\cap\mathrm{var}_{t'}}} \sum_{\mathbf{u}'\in\{0,1\}^{\mathrm{var}_{t'}\setminus(\mathrm{var}_t\cap\mathrm{var}_{t'})}} \left(f_{t'}(\mathbf{z},\mathbf{u}') \times p'(\mathbf{z},\mathbf{u}') \right)\\
&= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_{t'}}} \left(f_{t'}(\mathbf{x}) \times p'(\mathbf{x}) \right) \overset{(8)}{=} P(t').
\end{aligned}
$$

The second and the sixth equality is due to the fact that

$$(\mathrm{var}_t \cap \mathrm{var}_{t'}) \text{ and } (\mathrm{var}_t \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'}))$$

are disjoint sets, and so are $(\mathrm{var}_t \cap \mathrm{var}_{t'})$ and $(\mathrm{var}_{t'} \setminus (\mathrm{var}_t \cap \mathrm{var}_{t'}))$.

Note that the condition on the marginalized probability distributions also implies that, $\forall \mathbf{z} \in \{0,1\}^{\mathrm{var}_t \cap \mathrm{var}_{t'}}$,

$$\max_{\mathbf{u}\in\mathrm{var}_t\setminus(\mathrm{var}_t\cap\mathrm{var}_{t'})} p(\mathbf{z},\mathbf{u}) = \max_{\mathbf{u}'\in\mathrm{var}_{t'}\setminus(\mathrm{var}_t\cap\mathrm{var}_{t'})} p'(\mathbf{z},\mathbf{u}').$$

Thus, by replacing all sum operators in the above reasoning with max operators, we obtain a proof for $P_{\max}(t) = P_{\max}(t')$. □

Theorem 1 implies that the choice of the representative of the equivalence class is not relevant for the probability computation. This is of great value for efficiency considerations, as it allows us to choose the Boolean function we use. For instance, while evaluating the probability for ADTerm $t = B \wedge^{\mathrm{p}} (C \vee^{\mathrm{p}} B)$, where B and C are basic actions, instead of using $f_t(X_B, X_C) = X_B \times \max\{X_C, X_B\}$, we can employ $f_{t'}(X_B) = X_B$. This is because ADTerms $t = B \wedge^{\mathrm{p}} (C \vee^{\mathrm{p}} B)$ and $t' = B$ are equivalent in the propositional semantics[4]. Since $f_{t'}$ has a smaller domain than f_t, using $f_{t'}$ is more efficient.

We conclude this section by showing that, in the case of scenarios without dependent actions, the probability related to an ADTerm coincides with the success probability computed using the standard bottom-up procedure for ADTrees. For that, we first recall how to formalize the probability computation using the bottom-up algorithm and then we compare the two approaches.

In order to make use of the bottom-up algorithm, we assign the success probability value $\mathrm{Prob}(B)$ to each basic action B and specify how the probability of a composed ADTerm is computed from the probability values of its subterms. Let $t_1, t_2, \ldots, t_k \in \mathbb{T}$ and $s \in \{\mathrm{p}, \mathrm{o}\}$. The bottom-up procedure computes the probability as follows:

$$
\begin{aligned}
&\mathrm{Prob}(\wedge^s(t_1,\ldots,t_k)) = \prod_{i=1}^{k} \mathrm{Prob}(t_i)\\
&\mathrm{Prob}(\vee^s(t_1,\ldots,t_k)) = 1 - \prod_{i=1}^{k} (1 - \mathrm{Prob}(t_i))\\
&\mathrm{Prob}(\mathrm{c}^s(t_1,t_2)) = \mathrm{Prob}(t_1) \times (1 - \mathrm{Prob}(t_2)).
\end{aligned}
$$

Since the above formulas use the product operation to compute the probability of a conjunction of subgoals, they can only be applied under the assumption that all actions involved in the scenario are independent.

Theorem 2. *Let t be an ADTerm which does not contain any dependent actions. We have $P(t) = \mathrm{Prob}(t)$.*

Proof. Let us denote by p the probability distribution for the Bayesian network BN_t associated with t. Since all actions involved in t are independent, we have, $\mathrm{Prob}(B) = p(X_B = 1)$, for every $B \in \mathbb{B}$. Thus, for all $\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}$ we get

$$p(\mathbf{x}) = \prod_{X_B\in\mathrm{var}_t} p(\mathbf{x}^{\downarrow X_B}) = \prod_{X_B\in\mathrm{var}_t} \mathrm{Prob}(B).$$

We prove the theorem by using structural induction on t.

- If $t = B \in \mathbb{B}$, then

$$
\begin{aligned}
P(t) &= \sum_{\mathbf{x}\in\{0,1\}^{X_B}} f_B(\mathbf{x}) \times p(\mathbf{x})\\
&= \sum_{v\in\{0,1\}} f_B(X_B = v) \times p(X_B = v)
\end{aligned}
$$

[4] Methods for checking whether two ADTerms are equivalent in the propositional semantics have extensively been discussed in [19].

$$
\begin{aligned}
&= 0 \times p(X_B = 0) + 1 \times p(X_B = 1)\\
&= 0 \times (1 - \mathrm{Prob}(B)) + 1 \times \mathrm{Prob}(B)\\
&= \mathrm{Prob}(B) = \mathrm{Prob}(t).
\end{aligned}
$$

- Let us now assume that t is a composed ADTerm and that $P(t') = \mathrm{Prob}(t')$, for every subterm t' of t.
  - If $t = \wedge^s(t_1, \dots, t_k)$, we have

$$
\begin{aligned}
P(t) &= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} f_{\wedge^s(t_1,\dots,t_k)}(\mathbf{x}) \times p(\mathbf{x})\\
&= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} \left(\prod_{i=1}^{k} f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right) \times p(\mathbf{x}).
\end{aligned}
$$

Since by assumption t does not contain any dependent actions, it holds that, for all $i, j \in \{1, \dots, k\}$, such that $i \neq j$, the variables occurring in t_i do not occur in t_j. This means that configuration $\mathbf{x} \in \{0, 1\}^{\mathrm{var}_t}$ can be expressed as $(\mathbf{x}_1, \dots, \mathbf{x}_k)$, where $\mathbf{x}_i \in \{0, 1\}^{\mathrm{var}_{t_i}}$, $\mathrm{var}_t = \mathrm{var}_{t_1} \cup \dots \cup \mathrm{var}_{t_k}$, and domains var_{t_i} are mutually disjoint. Thus

$$
\begin{aligned}
&\sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} \left(\prod_{i=1}^{k} f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right) \times p(\mathbf{x})\\
&= \sum_{(\mathbf{x}_1,\dots,\mathbf{x}_k)\in\{0,1\}^{\mathrm{var}_t}} \left(\prod_{i=1}^{k} f_{t_i}(\mathbf{x}_i)\right) \times p(\mathbf{x}_1, \dots, \mathbf{x}_k)\\
&= \sum_{(\mathbf{x}_1,\dots,\mathbf{x}_k)\in\{0,1\}^{\mathrm{var}_t}} \left(\prod_{i=1}^{k} f_{t_i}(\mathbf{x}_i)\right) \times \prod_{i=1}^{k} p(\mathbf{x}_i)\\
&= \sum_{(\mathbf{x}_1,\dots,\mathbf{x}_k)\in\{0,1\}^{\mathrm{var}_t}} \prod_{i=1}^{k} (f_{t_i}(\mathbf{x}_i) \times p(\mathbf{x}_i))\\
&= \prod_{i=1}^{k} \sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} f_{t_i}(\mathbf{x}_i) \times p(\mathbf{x}_i)\\
&= \prod_{i=1}^{k} P(t_i) = \prod_{i=1}^{k} \mathrm{Prob}(t_i) = \mathrm{Prob}(t).
\end{aligned}
$$

  - If $t = \vee^s(t_1, \dots, t_k)$, we have

$$
\begin{aligned}
\mathrm{Prob}(t) &= 1 - \prod_{i=1}^{k} (1 - \mathrm{Prob}(t_i))\\
&= 1 - \prod_{i=1}^{k} (1 - P(t_i))\\
&= 1 - \prod_{i=1}^{k} \left[1 - \sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} f_{t_i}(\mathbf{x}_i) \times p(\mathbf{x}_i)\right]\\
&= 1 - \prod_{i=1}^{k} \left[\sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} p(\mathbf{x}_i) - \sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} f_{t_i}(\mathbf{x}_i) \times p(\mathbf{x}_i)\right]\\
&= 1 - \prod_{i=1}^{k} \sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} (p(\mathbf{x}_i) - f_{t_i}(\mathbf{x}_i) \times p(\mathbf{x}_i))\\
&= 1 - \prod_{i=1}^{k} \sum_{\mathbf{x}_i\in\{0,1\}^{\mathrm{var}_{t_i}}} (p(\mathbf{x}_i) \times (1 - f_{t_i}(\mathbf{x}_i)))\\
&= 1 - \sum_{(\mathbf{x}_1,\dots,\mathbf{x}_k)\in\{0,1\}^{\mathrm{var}_{t_i}}} \prod_{i=1}^{k} (p(\mathbf{x}_i) \times (1 - f_{t_i}(\mathbf{x}_i)))
\end{aligned}
$$

$$= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} p(\mathbf{x}) - \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_{t_i}}} p(\mathbf{x}) \times \prod_{i=1}^{k}\left(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right)$$

$$= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} p(\mathbf{x}) \times \left[1 - \prod_{i=1}^{k}\left(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right)\right].$$

Now it suffices to notice that $1 - \prod_{i=1}^{k}(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})) = 1$ if and only if $\prod_{i=1}^{k}(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})) = 0$. This holds if and only if there exists $i \in \{1, \dots, k\}$, such that $f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}}) = 1$. Thus, we conclude that

$$1 - \prod_{i=1}^{k}\left(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right) = \max_{i=1}^{k}\{f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\}.$$

Therefore,

$$\sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} p(\mathbf{x}) \times \left[1 - \prod_{i=1}^{k}\left(1 - f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\right)\right]$$
$$= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} p(\mathbf{x}) \times \max_{i=1}^{k}\{f_{t_i}(\mathbf{x}^{\downarrow \mathrm{var}_{t_i}})\}$$
$$= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} p(\mathbf{x}) \times f_{\vee^s(t_1,\dots,t_k)}(\mathbf{x})$$
$$= P(t).$$

- Finally, for $t = \mathrm{c}^s(t_1, t_2)$, we have

$$P(t) = \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} f_{\mathrm{c}^s(t_1,t_2)}(\mathbf{x}) \times p(\mathbf{x})$$
$$= \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} f_{t_1}(\mathbf{x}^{\downarrow \mathrm{var}_{t_1}}) \times \left(1 - f_{t_2}(\mathbf{x}^{\downarrow \mathrm{var}_{t_2}})\right) \times p(\mathbf{x})$$
$$= \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in\{0,1\}^{\mathrm{var}_t}} f_{t_1}(\mathbf{x}_1) \times (1 - f_{t_2}(\mathbf{x}_2)) \times p(\mathbf{x}_1) \times p(\mathbf{x}_2)$$
$$= \sum_{(\mathbf{x}_1,\mathbf{x}_2)\in\{0,1\}^{\mathrm{var}_t}} f_{t_1}(\mathbf{x}_1) \times p(\mathbf{x}_1) \times (1 - f_{t_2}(\mathbf{x}_2)) \times p(\mathbf{x}_2)$$
$$= \sum_{\mathbf{x}_1\in\{0,1\}^{\mathrm{var}_{t_1}}} f_{t_1}(\mathbf{x}_1) \times p(\mathbf{x}_1) \times \sum_{\mathbf{x}_2\in\{0,1\}^{\mathrm{var}_{t_2}}} (1 - f_{t_2}(\mathbf{x}_2)) \times p(\mathbf{x}_2)$$
$$= P(t_1) \times \sum_{\mathbf{x}_2\in\{0,1\}^{\mathrm{var}_{t_2}}} (p(\mathbf{x}_2) - f_{t_2}(\mathbf{x}_2) \times p(\mathbf{x}_2))$$
$$= P(t_1) \times \left(1 - \sum_{\mathbf{x}_2\in\{0,1\}^{\mathrm{var}_{t_2}}} f_{t_2}(\mathbf{x}_2) \times p(\mathbf{x}_2)\right)$$
$$= P(t_1) \times (1 - P(t_2))$$
$$= \mathrm{Prob}(t_1) \times (1 - \mathrm{Prob}(t_2)) = \mathrm{Prob}(t).$$

□

5. ADTerms as constraint systems

The rest of this paper is concerned with improving the efficiency of computing $P(t)$ and $P_{\max}(t)$, introduced in Section 4.3. Since the number of possible configurations is exponential with respect to the number of basic actions, the brute force computation of $P(t)$ and $P_{\max}(t)$, as illustrated in Examples 7 and 8, is no longer possible in the case of larger, real-life ADTrees.

In this section we present methods allowing us to represent formula (8) for $P(t)$ and formula (9) for $P_{\max}(t)$ in a factorized form. In Section 6, we discuss algorithms which make use of this factorized form in order to compute $P(t)$ and $P_{\max}(t)$ in a more efficient way.

5.1. Indicator functions for ADTerms

We propose to employ an encoding technique from constraint reasoning and transform an ADTerm t into its factorized *indicator function* ϕ_t. This function maps to 1 if and only if its arguments represent a valid assignment with respect to the Boolean function f_t.

In order to obtain a factorization of the global indicator function for the Boolean function interpreting an ADTerm in the propositional semantics, we first show how to construct *local indicators* for ADTerms. Local indicator functions make use of additional variables, called *inner variables*, to represent composed subterms. The construction of local indicators is the following:

1. If $t = \vee^{s}(t_1, \ldots, t_k)$, then the propositional variables $Y, Y_1, \ldots, Y_k$ are associated with $t, t_1, \ldots, t_k$, respectively, and the local indicator function for t is defined as: $\phi(Y, Y_1, \ldots, Y_k) = 1$ if $Y = \max\{Y_1, \ldots, Y_k\}$ and $\phi(Y, Y_1, \ldots, Y_k) = 0$ otherwise.
2. If $t = \wedge^{s}(t_1, \ldots, t_k)$, then the propositional variables $Y, Y_1, \ldots, Y_k$ are associated with $t, t_1, \ldots, t_k$, respectively, and the local indicator function for t is defined as: $\phi(Y, Y_1, \ldots, Y_k) = 1$ if $Y = Y_1 \times \cdots \times Y_k$ and $\phi(Y, Y_1, \ldots, Y_k) = 0$ otherwise.
3. If $t = c^{s}(t_1, t_2)$, then the propositional variables Y, Y_1 and Y_2 are associated with t, t_1 and t_2 respectively, and the local indicator function for t is defined as: $\phi(Y, Y_1, Y_2) = 1$ if $Y = Y_1 \times (1 - Y_2)$ and $\phi(Y, Y_1, Y_2) = 0$ otherwise.

Example 9. A step-wise construction of the local indicator functions for the ADTerm given in Example 3 proceeds as follows:

$$t = \underbrace{\wedge^{p}(\underbrace{c^{p}(\underbrace{\vee^{p}(\mathrm{SE}, \mathrm{DU})}_{Y_1}, \underbrace{\wedge^{o}(\mathrm{IA}, \underbrace{c^{o}(\mathrm{RA}, \mathrm{SA})}_{Y_2})}_{Y_3})}_{Y_4}, \mathrm{EV})}_{Y_t}$$

$$\begin{array}{lll}
t_1 = \vee^{p}(\mathrm{SE}, \mathrm{DU}) & \text{thus} & \phi_1(Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}) = 1 \text{ exactly if } Y_1 = \max(X_{\mathrm{SE}}, X_{\mathrm{DU}}); \\
t_2 = c^{o}(\mathrm{RA}, \mathrm{SA}) & \text{thus} & \phi_2(Y_2, X_{\mathrm{RA}}, X_{\mathrm{SA}}) = 1 \text{ exactly if } Y_2 = X_{\mathrm{RA}} \times (1 - X_{\mathrm{SA}}); \\
t_3 = \wedge^{o}(\mathrm{IA}, Y_2) & \text{thus} & \phi_3(Y_3, X_{\mathrm{IA}}, Y_2) = 1 \text{ exactly if } Y_3 = X_{\mathrm{IA}} \times Y_2; \\
t_4 = c^{p}(Y_1, Y_3) & \text{thus} & \phi_4(Y_4, Y_1, Y_3) = 1 \text{ exactly if } Y_4 = Y_1 \times (1 - Y_3); \\
t_5 = \wedge^{p}(Y_4, \mathrm{EV}) & \text{thus} & \phi_5(Y_t, Y_4, X_{\mathrm{EV}}) = 1 \text{ exactly if } Y_t = Y_4 \times X_{\mathrm{EV}}.
\end{array}$$

In this case, the inner variables are Y_1, Y_2, Y_3, Y_4 and Y_t. We illustrate the meaning of an indicator for an ADTerm using function ϕ_2 corresponding to subterm $c^{o}(\mathrm{RA}, \mathrm{SA})$. The defense represented by this subterm only works ($Y_2 = 1$) if an anti-virus program is running ($X_{\mathrm{RA}} = 1$) and there is no fake anti-virus in use ($X_{\mathrm{SA}} = 0$).

- The assignment $\mathbf{z} = (Y_2 = 1, X_{\mathrm{RA}} = 1, X_{\mathrm{SA}} = 0)$ is therefore logically valid and thus $\phi_2(\mathbf{z}) = 1$.
- The same holds for the assignment

  $$\mathbf{z}' = (Y_2 = 0, X_{\mathrm{RA}} = 1, X_{\mathrm{SA}} = 1),$$

  which represents that the defense does not work ($Y_2 = 0$) when the anti-virus is running ($X_{\mathrm{RA}} = 1$) but a fake anti-virus is used ($X_{\mathrm{SA}} = 1$). Thus, we also have $\phi_2(\mathbf{z}') = 1$.
- However, assignment

  $$\mathbf{z}'' = (Y_2 = 1, X_{\mathrm{RA}} = 1, X_{\mathrm{SA}} = 1)$$

  expresses that the defense is working ($Y_2 = 1$), while the anti-virus is running ($X_{\mathrm{RA}} = 1$) and a fake anti-virus is used ($X_{\mathrm{SA}} = 1$). Since assignment $\mathbf{z}''$ is logically invalid, we have $\phi_2(\mathbf{z}'') = 0$.

Let t be an ADTerm. Having constructed all local indicators for t, we can build the *global indicator function*ϕ_t. The domain of ϕ_t contains all variables used by the local indicator functions associated with the subterms of t, i.e., the inner variables and the variables corresponding to basic actions from t. An assignment over all variables is valid if and only if each local assignment is valid. Hence, we may compute the global indicator function for an ADTerm t by multiplying all its local indicators. For the term discussed in Example 9, we obtain

$$\begin{aligned}
&\phi_t(Y_1, Y_2, Y_3, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}) \\
&= \phi_1(Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}) \times \phi_2(Y_2, X_{\mathrm{RA}}, X_{\mathrm{SA}}) \times \phi_3(Y_3, X_{\mathrm{IA}}, Y_2) \times \phi_4(Y_4, Y_1, Y_3) \times \phi_5(Y_t, Y_4, X_{\mathrm{EV}}). \qquad (11)
\end{aligned}$$

In this paper, we use the following notation: given the global indicator function ϕ_t for t, we denote by Y_t the inner variable corresponding to the entire term t. The set of all inner variables of ϕ_t is denoted by d_t.

Remark 3. The local and global indicator functions are Boolean functions. Furthermore, the construction of the indicators implies that, if t and t' are two ADTerms equivalent in the propositional semantics, then $\phi_t \equiv \phi_{t'}$.

In practice, the global indicator function can still not be computed due its exponential growth. However, its factorization, in terms of local indicators which are bounded in size, introduces additional structure that can be exploited for so-called *local computation* [31]. In Section 6, we show that many interesting computational tasks can be performed on the factorization and therefore without explicit construction of the global indicator function.

The inner variables are important for revealing the factorization of the global indicator function, but they actually contain only redundant information. More precisely, consider an ADTerm $t = \xi^s(t_1, \dots, t_k)$, for some $t_1, \dots, t_k \in \mathbb{T}$, $\xi \in \{c, \vee, \wedge\}$, $s \in \{p, o\}$, $k \in \mathbb{N}$, and the corresponding indicator function $\phi(Y, Y_1, \dots, Y_k)$. Let $\mathbf{z}$ be an assignment of values to the variables $Y_1, \dots, Y_k$. There is, by definition, exactly one value $y \in \{0, 1\}$ for Y, such that $\phi(y, \mathbf{z}) = 1$. Since the global indicator function is obtained by multiplication, we may directly conclude the following theorem.

Theorem 3. *Consider an ADTerm t with basic actions $B_1, \dots, B_n$ and its global indicator function ϕ_t. Given a specific assignment $\mathbf{x}$ of values to the variables $X_{B_1}, \dots, X_{B_n}$ corresponding to basic actions, there is exactly one assignment $\mathbf{y}$ to the inner variables from d_t, such that $\phi_t(\mathbf{y}, \mathbf{x}) = 1$.*

An immediate consequence of Theorem 3 is that, for a specific assignment $\mathbf{x} \in \{0, 1\}^{\mathrm{var}_t}$ of values to the variables associated with basic actions, we have

$$\max_{\mathbf{y} \in \{0,1\}^{d_t}} \phi_t(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{y} \in \{0,1\}^{d_t}} \phi_t(\mathbf{y}, \mathbf{x}) = 1. \tag{12}$$

5.2. Indicators for probability computation

We use the propositional interpretation for ADTerms in order to evaluate which combinations of basic actions correspond to attacks. However, the global indicator function ϕ_t maps to 1 for all valid assignments, including those corresponding to unsuccessful cases. For instance, in the case of the function given by formula (11), we have

$$\phi_t(Y_1 = 1, Y_2 = 0, Y_3 = 0, Y_4 = 1, Y_t = 0, X_{\mathrm{SE}} = 1, X_{\mathrm{DU}} = 1, X_{\mathrm{RA}} = 1, X_{\mathrm{SA}} = 1, X_{\mathrm{IA}} = 1, X_{\mathrm{EV}} = 0) = 1.$$

The considered configuration, although valid, does not correspond to a successful infection of a computer, because $Y_t = 0$. This shows that, when reasoning in terms of global indicator functions, we are only interested in configurations where variable Y_t equals 1. We therefore *condition* ϕ_t on $Y_t = 1$, which means that we invalidate all configurations with $Y_t = 0$. This is achieved by defining a *filter* F_t for ADTerm t, that satisfies

$$F_t(Y_t) = 1 \quad \text{if and only if} \quad Y_t = 1.$$

In other words, $F_t : \{0, 1\}^{\{Y_t\}} \to \{0, 1\}$ is the identity function for variable Y_t. Multiplying filter F_t and global indicator ϕ_t results in a function, denoted by $\phi_{t\,|Y_t=1}$, which maps to 1 if and only if the assignment of values to the variables is valid with respect to f_t *and* it represents an attack according to t. We thus have

$$\forall \mathbf{z} \in \{0, 1\}^{\mathrm{var}_t \cup d_t} : \phi_{t\,|Y_t=1}(\mathbf{z}) = F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_t(\mathbf{z}). \tag{13}$$

For our running example, we obtain

$$\begin{aligned} &\phi_{t\,|Y_t=1}(Y_1, \dots, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}) \\ &\quad = F_t(Y_t) \times \phi_t(Y_1, Y_2, Y_3, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}) \\ &\quad = F_t(Y_t) \times \phi_1(Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}) \times \phi_2(Y_2, X_{\mathrm{RA}}, X_{\mathrm{SA}}) \times \phi_3(Y_3, X_{\mathrm{IA}}, Y_2) \times \phi_4(Y_4, Y_1, Y_3) \times \phi_5(Y_t, Y_4, X_{\mathrm{EV}}). \end{aligned} \tag{14}$$

Remark 4. By construction, we know that, if t and t' are two ADTerms that are equivalent in the propositional semantics, then F_t and $F_{t'}$ are equivalent Boolean functions. Thus, using Remark 3, we have $\phi_{t\,|Y_t=1} \equiv \phi_{t'\,|Y_{t'}=1}$.

Let t be an ADTerm, ϕ_t be its global indicator function and F_t be the filter for t. Assume furthermore that we are given a specific configuration $\mathbf{x} \in \{0, 1\}^{\mathrm{var}_t}$. Configuration $\mathbf{x}$ is an attack with respect to t if and only if, there exists $\mathbf{y} \in \{0, 1\}^{d_t}$, such that $\phi_{t\,|Y_t=1}(\mathbf{y}, \mathbf{x})$ maps to 1. From formula (12), it follows that

$$\begin{aligned} \max_{\mathbf{y} \in \{0,1\}^{d_t}} \phi_{t\,|Y_t=1}(\mathbf{y}, \mathbf{x}) &= \sum_{\mathbf{y} \in \{0,1\}^{d_t}} \phi_{t\,|Y_t=1}(\mathbf{y}, \mathbf{x}) \\ &= f_t(\mathbf{x}) = \begin{cases} 1 \text{ if } \mathbf{x} \text{ is an attack} \\ 0 \text{ if } \mathbf{x} \text{ is not an attack.} \end{cases} \end{aligned} \tag{15}$$

Making use of the property described by (15), the procedure for the probabilistic computations developed in Section 4.3 can be redefined as follows.

Let t be an ADTerm and $\mathbf{x} \in \{0,1\}^{\mathrm{var}_t}$ be an assignment of Boolean values to the variables corresponding to the basic actions involved in t. If $\mathbf{x}$ is an attack with respect to t, then its probability is computed as

$$\psi_t(\mathbf{x}) \overset{(7)}{=} f_t(\mathbf{x}) \times p(\mathbf{x}) \overset{(15),\text{distrib.}}{=} \sum_{\mathbf{y}\in\{0,1\}^{d_t}} \big(\phi_{t\,|Y_t=1}(\mathbf{y},\mathbf{x}) \times p(\mathbf{x})\big) \tag{16}$$

$$\overset{(15),\text{distrib.}}{=} \max_{\mathbf{y}\in\{0,1\}^{d_t}} \big(\phi_{t\,|Y_t=1}(\mathbf{y},\mathbf{x}) \times p(\mathbf{x})\big). \tag{17}$$

The probability related to ADTerm t is expressed as

$$\begin{aligned} P(t) &\overset{(8)}{=} \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} \psi_t(\mathbf{x}) \\ &\overset{(16)}{=} \sum_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} \sum_{\mathbf{y}\in\{0,1\}^{d_t}} \big(\phi_{t\,|Y_t=1}(\mathbf{y},\mathbf{x}) \times p(\mathbf{x})\big) \\ &= \sum_{\mathbf{z}\in\{0,1\}^{\mathrm{var}_t\cup d_t}} \big(\phi_{t\,|Y_t=1}(\mathbf{z}) \times p(\mathbf{z}^{\downarrow \mathrm{var}_t})\big). \end{aligned} \tag{18}$$

Similarly, the probability of the most probable attack with respect to t is

$$\begin{aligned} P_{\max}(t) &\overset{(9)}{=} \max_{\mathbf{x}\in\{0,1\}^{\mathrm{var}_t}} \psi_t(\mathbf{x}) \\ &\overset{(17)}{=} \max_{\mathbf{z}\in\{0,1\}^{\mathrm{var}_t\cup d_t}} \big(\phi_{t\,|Y_t=1}(\mathbf{z}) \times p(\mathbf{z}^{\downarrow \mathrm{var}_t})\big). \end{aligned} \tag{19}$$

We illustrate the factorized form for the probability related to ADTerm on our running example.

Example 10. Let $u = \{Y_1, Y_2, Y_3, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}\}$. The probability related to ADTerm t given by Eq. (1) is computed as

$$\begin{aligned} P(t) \overset{(18),(14),(6)}{=} \sum_{\mathbf{z}\in\{0,1\}^{u}} \Big(& F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \phi_2(\mathbf{z}^{\downarrow\{Y_2,X_{\mathrm{RA}},X_{\mathrm{SA}}\}}) \\ & \times \phi_3(\mathbf{z}^{\downarrow\{Y_3,X_{\mathrm{IA}},Y_2\}}) \times \phi_4(\mathbf{z}^{\downarrow\{Y_4,Y_1,Y_3\}}) \times \phi_5(\mathbf{z}^{\downarrow\{Y_t,Y_4,X_{\mathrm{EV}}\}}) \\ & \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{EV}},X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{SE}},X_{\mathrm{SA}}\}}) \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{DU}},X_{\mathrm{SA}}\}}) \\ & \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{SA}}\}}) \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{RA}},X_{\mathrm{IA}}\}}) \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\Big). \end{aligned} \tag{20}$$

6. Efficiency considerations

As we already pointed out, in most cases it is impossible to construct the global indicator function and the joint probability distribution efficiently. Hence, we need a way of computing that limits the size of intermediate results and therefore makes head against the exponential growth. In this section, we introduce an algorithm, called *fusion*, that allows us to improve the efficiency of evaluating formulas (18) and (19).

6.1. Semiring valuations

Let us start by recalling the definition of a commutative semiring.

Definition 6. An algebraic structure $\langle A, \oplus, \odot\rangle$ with binary operations $\oplus$ and $\odot$ is called *commutative semiring* if both operations are associative, commutative, and if $\odot$ distributes over $\oplus$, i.e., if for $a, b, c \in A$, we have $a \odot (b \oplus c) = (a \odot b) \oplus (a \odot c)$ and $(a \oplus b) \odot c = (a \odot c) \oplus (b \odot c)$.

Typical examples of commutative semirings include

- the Boolean semiring $\langle\{0, 1\}, \max, \times\rangle$,
- the tropical semiring $\langle\mathbb{N}, \min, +\rangle$,
- the product t-norm semiring $\langle[0, 1], \max, \times\rangle$, and
- the arithmetic semiring $\langle\mathbb{R}, +, \times\rangle$.

Further examples can be found in [31].

Let $u \subseteq r$ be a finite set of propositional variables and $\langle A, \oplus, \odot\rangle$ be a commutative semiring. A *semiring valuation* over $\langle A, \oplus, \odot\rangle$ is a function ϕ: $\{0, 1\}^u \to A$ associating a value from A with each configuration from $\{0, 1\}^u$. We denote by $\mathrm{dom}(\phi) = u$ the domain of valuation ϕ.

Consider two valuations ϕ and ψ over a semiring $\langle A, \oplus, \odot \rangle$, with domains $\mathrm{dom}(\phi) = u$ and $\mathrm{dom}(\psi) = w$. The *combination of ϕ and ψ*, denoted by $(\phi \otimes \psi)$, is defined, for all $\mathbf{x} \in \{0, 1\}^{u \cup w}$, as

$$(\phi \otimes \psi)(\mathbf{x}) = \phi(\mathbf{x}^{\downarrow u}) \odot \psi(\mathbf{x}^{\downarrow w}).$$

Likewise, the *elimination* of variable $X \in u$ is defined for all $\mathbf{x} \in \{0, 1\}^{u \setminus \{X\}}$ as

$$\phi^{-X}(\mathbf{x}) = \phi(\mathbf{x}, 0) \oplus \phi(\mathbf{x}, 1).$$

Due to associativity of semiring addition $\oplus$, we can eliminate variables in any order. For $\{X_1, \ldots, X_m\} \subseteq \mathrm{dom}(\phi)$, we may therefore write

$$\phi^{-\{X_1,\ldots,X_m\}} = (\cdots((\phi^{-X_1})^{-X_2})\cdots)^{-X_m}.$$

Indicator functions, as used in Section 5.1 for modeling ADTerms in the propositional semantics, are Boolean semiring valuations over $\langle \{0, 1\}, \max, \times \rangle$. Arithmetic semiring valuations over $\langle \mathbb{R}, +, \times \rangle$ capture conditional probability tables from Bayesian networks in Section 4.2, and product t-norm semiring valuations over $\langle [0, 1], \max, \times \rangle$ compute maximum attack probabilities, as in formula (19).

It has been shown in [13] that semiring valuations over arbitrary commutative semirings always satisfy the axioms of a valuation algebra [12,31]. The computational interest in valuation algebras is stated by the *inference problem*. Given a set of (semiring) valuations $\{\phi_1, \ldots, \phi_n\}$, called *knowledgebase*, with domains $u_i = \mathrm{dom}(\phi_i)$, for $i = 1, \ldots, n$, and a set of variables $\{X_1, \ldots, X_m\} \subseteq u_1 \cup \cdots \cup u_n$, the inference problem consists of computing

$$\phi^{-\{X_1,\ldots,X_m\}} = (\phi_1 \otimes \cdots \otimes \phi_n)^{-\{X_1,\ldots,X_m\}}. \tag{21}$$

Example 11. Let $u = \{Y_1, Y_2, Y_3, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}\}$. Computing the probability in Example 10 amounts to solving the inference problem

$$\left(F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \cdots \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\right)^{-u} = \sum_{\mathbf{z}\in\{0,1\}^u} \left(F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \cdots \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\right).$$

Here, the knowledgebase consists of all local indicator functions, filter F_t and all conditional probability tables, which instantiate arithmetic semiring valuations. Likewise, computing maximum attack probability, expressed by formula (19), amounts to solving a similar inference problem over the product t-norm semiring, i.e.,

$$\left(F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \cdots \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\right)^{-u} = \max_{\mathbf{z}\in\{0,1\}^u} \left(F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \cdots \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\right).$$

6.2. *Fusion*

We argued that a direct evaluation of formulas (18), (19), and more generally of (21), is in most cases not possible, due to the exponential complexity of combination of semiring valuations. However, because the computational tasks are stated with respect to a factorization of the global indicator function and the joint probability distribution, we may exploit the additional structure inside the factorization and perform calculations locally on the domain of the factors. This general technique is called *local computation*. Different local computation algorithms have been proven correct for arbitrary valuation algebras [31]. One such algorithm is the *fusion* [37] (or *bucket-elimination* [6]) algorithm presented below. It corresponds to the standard algorithm for evaluating Bayesian networks. Because it can more generally be applied to factorizations of arbitrary (semiring) valuation algebras, we may use fusion for processing inference problems obtained from ADTerms.

Let us first consider the elimination of a single variable $X \in \mathrm{dom}(\phi) = \mathrm{dom}(\phi_1) \cup \cdots \cup \mathrm{dom}(\phi_n)$ from a set $\{\phi_1, \ldots, \phi_n\}$ of valuations. This operation can be performed as follows:

$$\mathrm{Fus}_X(\{\phi_1, \ldots, \phi_n\}) = \{\psi^{-X}\} \cup \{\phi_i : X \notin \mathrm{dom}(\phi_i)\}, \tag{22}$$

where $\psi = \bigotimes_{i:X\in\mathrm{dom}(\phi_i)} \phi_i$. This means that we only need to eliminate X from the factors that have X in the domain. The fusion algorithm then follows by repeated application of this operation

$$(\phi_1 \otimes \cdots \otimes \phi_n)^{-\{X_1,\ldots,X_m\}} = \bigotimes \mathrm{Fus}_{X_m}(\cdots(\mathrm{Fus}_{X_1}(\{\phi_1, \ldots, \phi_n\}))).$$

For proofs see [12]. In every step $i = 1, \ldots, m$ of the fusion algorithm, the combination in (22) creates an intermediate factor ψ_i with domain $\mathrm{dom}(\psi_i)$. Then, variable X_i is eliminated only from ψ_i in (22). We define $\lambda(i) = \mathrm{dom}(\psi_i) \setminus \{X_i\}$ called *label* and observe that $\lambda(m) = (\mathrm{dom}(\phi_1) \cup \cdots \cup \mathrm{dom}(\phi_n)) \setminus \{X_1, \ldots, X_m\}$. The domains of all intermediate results of the fusion algorithm are therefore bounded by the size of the largest label. The smaller the labels are, the more efficient fusion is. We further remark that the labels depend on the chosen elimination sequence for variables $X_1, \ldots, X_m$. Finding a sequence that leads to the smallest largest label is NP-complete [1], however, there are good heuristics that achieve reasonable execution time [7]. In summary, the complexity of computing (21) is not necessarily exponential in the number of variables involved in the problem, but only in the size of the largest label that occurs during fusion. The size of the largest label is also called *tree width* [34].

Intuitively speaking, the goal of the fusion algorithm is to apply the distributivity law in a 'smart' way. Example 12 illustrates the application of fusion to the computation of the probability related to our running ADTerm t.

Example 12. We have applied fusion to the first inference problem from Example 11. For the elimination sequence

$$Y_t, X_{\mathrm{RA}}, X_{\mathrm{IA}}, Y_2, X_{\mathrm{EV}}, Y_3, X_{\mathrm{SA}}, X_{\mathrm{DU}}, X_{\mathrm{SE}}, Y_4, Y_1,$$

the algorithm results in the following distributed form for $P(t)$:

$$\begin{aligned}
&\sum_{\mathbf{z}\in\{0,1\}^u} \left(F_t(\mathbf{z}^{\downarrow\{Y_t\}}) \times \phi_1(\mathbf{z}^{\downarrow\{Y_1,X_{\mathrm{SE}},X_{\mathrm{DU}}\}}) \times \cdots \times p(\mathbf{z}^{\downarrow\{X_{\mathrm{IA}}\}})\right)\\
&= \sum_{Y_1,Y_4,X_{\mathrm{SE}},X_{\mathrm{DU}}} \phi_1(Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}) \left[\sum_{X_{\mathrm{EV}}} p(X_{\mathrm{EV}}|X_{\mathrm{SE}}, X_{\mathrm{DU}}) \times \left(\sum_{Y_t} F_t(Y_t) \times \phi_5(Y_t, Y_4, X_{\mathrm{EV}})\right)\right]\\
&\quad \times \left\{\sum_{X_{\mathrm{SA}}} p(X_{\mathrm{SE}}|X_{\mathrm{SA}}) \times p(X_{\mathrm{DU}}|X_{\mathrm{SA}}) \times p(X_{\mathrm{SA}})\right.\\
&\quad \left.\times \left(\sum_{Y_3} \phi_4(Y_4, Y_1, Y_3) \left[\sum_{Y_2,X_{\mathrm{IA}}} \phi_3(Y_3, X_{\mathrm{IA}}, Y_2) \times p(X_{\mathrm{IA}}) \left(\sum_{X_{\mathrm{RA}}} \phi_2(Y_2, X_{\mathrm{RA}}, X_{\mathrm{SA}}) \times p(X_{\mathrm{RA}}|X_{\mathrm{IA}})\right)\right]\right)\right\}. \qquad (23)
\end{aligned}$$

Taking a closer look at the resulting distributed form, we notice that the largest intermediate factor has domain $\{Y_4, X_{\mathrm{SA}}, Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}\}$. This factor is produced right before variable X_{SA} is eliminated (marked with curly braces) in the 7th step of the elimination sequence. We therefore have

$$\#\lambda(7) = \#(\{Y_4, X_{\mathrm{SA}}, Y_1, X_{\mathrm{SE}}, X_{\mathrm{DU}}\} \setminus \{X_{\mathrm{SA}}\}) = 4.$$

This means that the tree width of our problem is at most 4, which is only one variable more than the largest input factor. This implies that time and space complexity of the computation of $P(t)$ using fusion are bounded by 2^4. To compare, a naive, direct computation, making use of expression (20), is bounded by 2^{11}.

Fusion applies to computational tasks that take the form of formula (21), i.e., that amount to eliminating variables from a factorized global semiring valuation. Due to the encoding with indicator functions, both, the probability related to an ADTerm, expressed by formula (18), and the probability of the most probable attack, expressed by formula (19), can be written in this form. However, sometimes we are less interested in a concrete value, such as the maximum attack probability, but rather in finding one or more configurations that map to this value, as they describe the *most probable attacks*. Fusion can be modified to additionally output such *solution configurations* [31] under the condition that the semiring is totally ordered and idempotent. This is the case for the product t-norm semiring so that we may use fusion to compute maximum (or minimum) attack probabilities as well as the most (or least) probable attacks. A detailed description of how this can be achieved can be found in [30]. It basically amounts to storing intermediate results during the fusion process. Whenever we eliminate one variable during fusion and therefore perform a maximization, we memorize the variable value (0 or 1) that maps to the maximum probability value. Then, after fusion has been completed, we revisit all eliminated variables in the inverse direction of the elimination sequence and build solution configurations from these intermediate results.

Finally, the computation of $P(t)$, as presented in Examples 10 and 12 assumes that we do not know which actions will and which ones will not be executed. This is why we consider all $\mathbf{z} \in \{0, 1\}^u$ for $u = \{Y_1, Y_2, Y_3, Y_4, Y_t, X_{\mathrm{SE}}, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}\}$. In practice however, we might have some additional knowledge about certain actions, e.g., we might know that an e-mail with a malicious attachment has successfully been sent ($X_{\mathrm{SE}} = 1$), and ask what the success probability of the related scenario is in this case. The factorized form and the fusion algorithm can be used to answer this question, as illustrated in Example 13.

Example 13. Let us denote by $P(Y_t = 1|X_{\mathrm{SE}} = 1)$ the success probability of the security scenario represented by ADTerm t given by Eq. (1), assuming that an e-mail with a malicious attachment has successfully been sent ($X_{\mathrm{SE}} = 1$). Since

$$P(Y_t = 1|X_{\mathrm{SE}} = 1) = P(Y_t = 1, X_{\mathrm{SE}} = 1)/P(X_{\mathrm{SE}} = 1),$$

it suffices to compute the values of $P(Y_t = 1, X_{\mathrm{SE}} = 1)$ and $P(X_{\mathrm{SE}} = 1)$ to evaluate the probability $P(Y_t = 1|X_{\mathrm{SE}} = 1)$.

- By applying the fusion algorithm to the factorization from Example 10 and the set of variables $u = \{Y_1, Y_2, Y_3, Y_4, X_{\mathrm{DU}}, X_{\mathrm{RA}}, X_{\mathrm{SA}}, X_{\mathrm{IA}}, X_{\mathrm{EV}}\}$, we compute the distribution $P(Y_t, X_{\mathrm{SE}})$ and extract the values $P(Y_t = 1, X_{\mathrm{SE}} = 1)$ and $P(Y_t = 0, X_{\mathrm{SE}} = 1)$.
- According to the law of marginalization, we then obtain

$$P(X_{\mathrm{SE}} = 1) = P(Y_t = 1, X_{\mathrm{SE}} = 1) + P(Y_t = 0, X_{\mathrm{SE}} = 1).$$

Hence, one run of fusion, one addition, and one division result in the computation of $P(Y_t = 1|X_{\mathrm{SE}} = 1)$.

7. Practical considerations

Before concluding our paper, we discuss practical aspects of the proposed framework.

7.1. Automated computation

The approach presented in Section 6.2 consists of transforming the problem of computing probabilities on ADTrees to an inference problem such that it can be efficiently solved by the fusion algorithm. As we have seen in Example 12, employing fusion implies that time and space complexity are bounded by a structural parameter of the problem rather than by the total number of variables involved. This makes computation of even very large problems possible.

Fusion itself is a well-established technique implemented in many tools for modeling and solving Bayesian networks and constraint systems. We have automated the computation of the probability related to our running ADTerm with the help of the open-source tool Nenok [32]. Nenok provides an extensive library of generically implemented (i.e., for general valuation algebras) local computation algorithms, including fusion. The default heuristic of Nenok called *One-Step-Look-Ahead Smallest Clique* [23] proposes the elimination sequence used in Example 12. When applying fusion, Nenok outputs the value of $P(t)$ after 0.031 sec, in contrast to 3.422 sec that the application requires to compute the same value in a naive way, i.e., by using expression (20) directly.

7.2. Creating Bayesian network

As any method for quantitative analysis, our framework may be exposed to the problem of how to identify the dependency relations and obtain the appropriate conditional probability values, in the case of real-life scenarios. Since the approach presented in this paper keeps the ADTree and the Bayesian network separate, we may directly use existing knowledge engineering techniques from the Bayesian network community in order to collect and process data and related structural information. Such approaches combine domain expert knowledge with machine learning techniques. They have already been successfully applied in the security context. In [35], for instance, the authors make use of probabilistic learning methods and introduce Bayesian Spam filters which learn conditional probability distributions with user interaction. Furthermore, there exist techniques, e.g., noisy OR, that simplify the construction of conditional distributions with many parent nodes [11]. We refer to [43] for an overview of different approaches for constructing Bayesian networks.

Setting up conditional probability tables requires, in the worst case, initialization of exponentially many values. However, it is often possible to reason with parametrized families of distributions, where it is sufficient to only provide the respective parameters of the distribution. This, in turn, is not more complicated than estimating regular, non-probabilistic parameters, such as, cost or time and it is linear with respect to the number of variables of the model. Finally, our algorithmic framework is also suitable to work with imprecise probabilities [10], which also reduces the size of the necessary input.

7.3. Scalability

The complexity of fusion is $O(m \cdot 2^{\omega})$, where m denotes the number of variables (in our case basic actions) to eliminate and ω is the tree width of the problem. This holds for any semiring and is in contrast to a direct computation of expressions such as (20) with complexity $O(2^m)$. As the complexity of fusion mainly depends on a structural problem parameter, it cannot be predicted in general how well it can cope with large problems involving many variables. It all depends on whether a small tree width (or good elimination sequence) can be found by some heuristic. Prediction of the tree width is possible for specific families of graphs [4]. It is one of our future research directions to investigate whether combination of an ADTree with a Bayesian network, both produced by human security experts, would satisfy the definition of one such family.

8. Conclusion and future work

8.1. Conclusion

This paper proposes to combine the ADTree methodology with Bayesian networks in order to *evaluate probabilistic measures on attack–defense scenarios involving dependent actions*. The approach has been illustrated on a running example and the probabilistic computations have been automated using the Nenok tool. A diagram visualizing the proposed framework is given in Fig. 3.

The introduced approach improves upon the standard, bottom-up, computational routine for attack tree-based formalisms, which assumes that all actions involved in the model are independent. By lifting the independency assumption, we provide a pragmatic methodology for accurate probabilistic assessment of security scenarios modeled using attack trees or ADTrees.

As depicted in Fig. 3, in our framework, the Bayesian network *does not replace* the information represented by the structure of an ADTree, but complements it with *additional probabilistic dependencies* between attack steps, which cannot be depicted using AND-OR relations. Keeping the two models separated allows us to take advantage of the strengths of both formalisms. The propositional encoding of ADTrees is used to identify configurations which represent attacks and makes

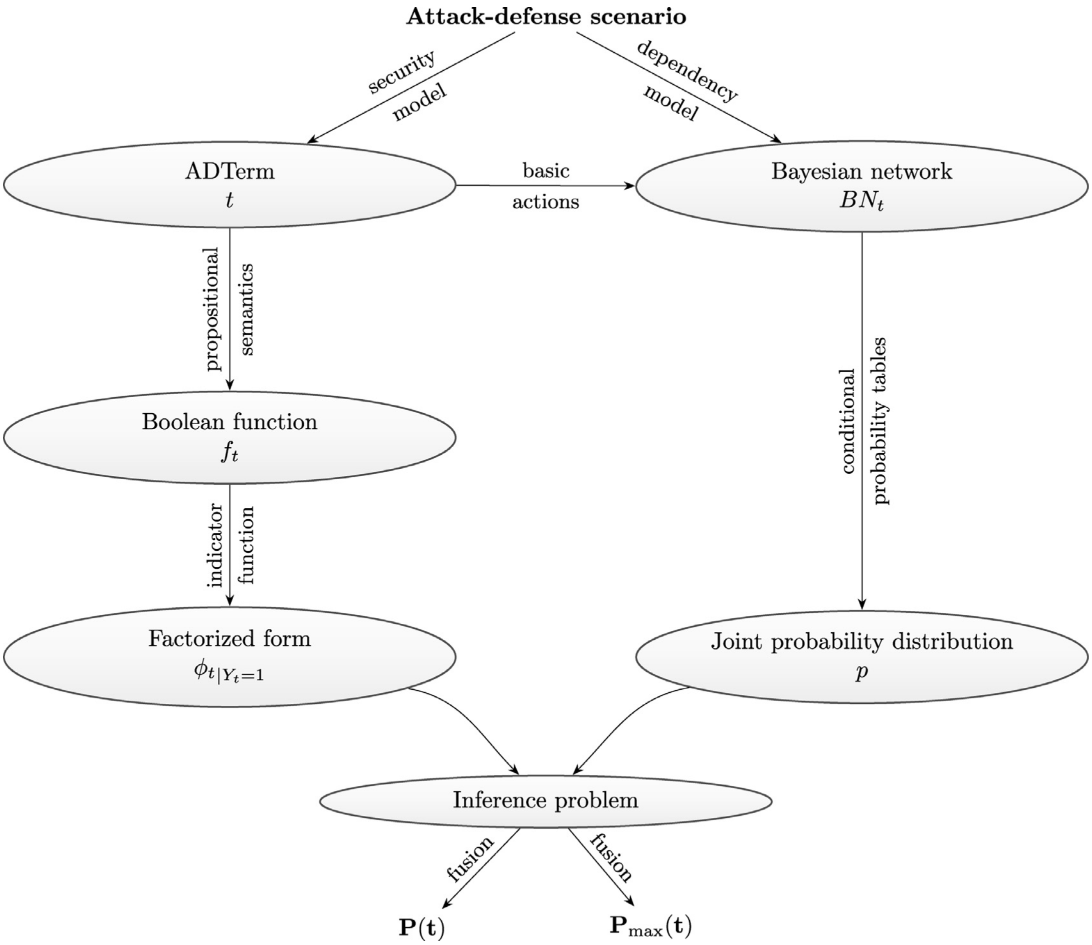


Fig. 3. Our probabilistic framework for attack–defense scenarios with dependent actions.

reasoning about equivalent scenarios possible. Bayesian networks together with the fusion algorithm and techniques based on semiring valuation algebras provide ways to improve the efficiency of probabilistic computations.

We prove that our computational framework is well defined with respect to the propositional semantics. Furthermore, we show that, if no dependent actions are present, the method proposed in this paper matches the standard technique for quantitative evaluation of ADTrees. This means that our approach is a conservative extension of the ADTree methodology from [19].

8.2. Future work

We are currently extending the functionality of ADTool [17] by interfacing it with Nenok [32], so that it can handle the framework introduced in this paper. Since Nenok implements generic algorithms for efficient processing of semiring-based computations, the extended tool will support efficient, automated probabilistic analysis of real-world, possibly large-scale, attack–defense scenarios.

We are also setting-up a realistic case study in order to test and validate the computational framework developed in this paper. The goals of the case study are twofold. First, we would like to evaluate the efficacy of the encoding technique based on indicator functions for real-world security assessment. Moreover, we will investigate whether fusion can cope with realistic, large-scale models, as explained in Section 7.3, or whether we have to fall back to methods such as approximations or anytime algorithms. Second, we will turn our attention towards the problem of identification of dependencies between actions and their quantification, i.e., estimation of the input values for the related conditional probability tables. In particular, we plan to experimentally analyze the influence of possible errors in the inputs to the sensitivity of the final output.

The algorithmic technique based on semiring valuations that we have used in this paper also works in a broader context. From an algebraic perspective, the combination of indicator functions and probabilities is possible because the Boolean semiring for indicator functions is a sub-algebra of both semirings used for expressing probabilities, i.e., the arithmetic and product t-norm semiring. Consequently, we may directly apply the same construction to other semiring valuations under the additional condition that the corresponding semiring takes the Boolean semiring as sub-algebra. The large family of t-norm semirings [31] are important candidates used in possibility and fuzzy set theory [45].

Acknowledgments

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement number 318003 (TREsPASS) and from the Fonds National de la Recherche Luxembourg under grants PHD-09-167 and C13/IS/5809105.

References

[1] S. Arnborg, D. Corneil, A. Proskurowski, Complexity of finding embeddings in a *k*-tree, SIAM J. Algebraic Discret. Methods 8 (1987) 277–284.
[2] A. Bagnato, B. Kordy, P.H. Meland, P. Schweitzer, Attribute decoration of attack–defense trees, Int. J. Secur. Softw. Eng. 3 (2) (2012) 1–35, doi:10.4018/jsse.2012040101.
[3] S. Bistarelli, F. Fioravanti, P. Peretti, Defense trees for economic evaluation of security investments, in: Proceedings of the ARES, IEEE Computer Society, 2006, pp. 416–423.
[4] H.L. Bodlaender, A partial *k*-arboretum of graphs with bounded treewidth, Theor. Comput. Sci. 209 (1–2) (1998) 1–45.
[5] D. Byers, N. Shahmehri, Unified modeling of attacks, vulnerabilities and security activities, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems (SESS ’10), ACM, New York, NY, USA, 2010, pp. 36–42.
[6] R. Dechter, Bucket elimination: a unifying framework for reasoning, Artif. Intell. 113 (1999) 41–85.
[7] R. Dechter, Constraint Processing, Morgan Kaufmann, 2003.
[8] M. Frigault, L. Wang, Measuring network security using Bayesian network-based attack graphs, in: Proceedings of the COMPSAC, 2008, pp. 698–703, doi:10.1109/COMPSAC.2008.88.
[9] M. Frigault, L. Wang, A. Singhal, S. Jajodia, Measuring network security using dynamic Bayesian network, in: Proceedings of the QoP, 2008, pp. 23–30.
[10] J.Y. Halpern, Reasoning About Uncertainty, MIT Press, Cambridge, MA, USA, 2003.
[11] Y. Halpern, D. Sontag, Unsupervised learning of noisy-or Bayesian networks, in: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence (UAI’13), AUAI Press, 2013, pp. 272–281.
[12] J. Kohlas, Information Algebras: Generic Structures for Inference, Springer, 2003.
[13] J. Kohlas, N. Wilson, Semiring induced valuation algebras: exact and approximate local computation algorithms, Artif. Intell. 172 (11) (2008) 1360–1399.
[14] B. Kordy, S. Mauw, M. Melissen, P. Schweitzer, Attack–defense trees and two-player binary zero-sum extensive form games are equivalent, in: T. Alpcan, L. Buttyán, J.S. Baras (Eds.), GameSec, LNCS, vol. 6442, Springer, 2010, pp. 245–256.
[15] B. Kordy, S. Mauw, S. Radomirović, P. Schweitzer, Foundations of attack–defense trees, in: P. Degano, S. Etalle, J.D. Guttman (Eds.), FAST, LNCS, vol. 6561, Springer, 2010, pp. 80–95.
[16] B. Kordy, M. Pouly, P. Schweitzer, Computational aspects of attack–defense trees, in: P. Bouvry, M.A. Klopotek, F. Leprévost, M. Marciniak, A. Mykowiecka, H. Rybinski (Eds.), SIIS, LNCS, vol. 7053, Springer, 2011, pp. 103–116, doi:10.1007/978-3-642-25261-7_8.
[17] B. Kordy, P. Kordy, S. Mauw, P. Schweitzer, ADTool: security analysis with attack–defense trees, in: K.R. Joshi, M. Siegle, M. Stoelinga, P.R. D’Argenio (Eds.), QEST, LNCS, vol. 8054, Springer, 2013, pp. 173–176.
[18] B. Kordy, S. Mauw, P. Schweitzer, Quantitative questions on attack–defense trees, in: T. Kwon, M. Lee, D. Kwon (Eds.), ICISC, LNCS, vol. 7839, Springer, 2013, pp. 49–64, doi:10.1007/978-3-642-37682-5_5.
[19] B. Kordy, S. Mauw, S. Radomirović, P. Schweitzer, Attack–defense trees, J. Log. Comput. 24 (1) (2014) 55–87, doi:10.1093/logcom/exs029.
[20] B. Kordy, L. Piètre-Cambacédès, P. Schweitzer, DAG-based attack and defense modeling: don’t miss the forest for the attack trees, Comput. Sci. Rev. 13–14 (2014) 1–38, doi:10.1016/j.cosrev.2014.07.001.
[21] B. Kordy, M. Pouly, P. Schweitzer, A probabilistic framework for security scenarios with dependent actions, in: E. Albert, E. Sekerinski (Eds.), IFM’14, LNCS, vol. 8739, Springer, 2014, pp. 256–271.
[22] R. Lagerström, P. Johnson, P. Närman, Extended influence diagram generation, in: R. Jardim-Gonçalves, J.P. Müller, K. Mertins, M. Zelm (Eds.), IESA, Springer, 2007, pp. 599–602.
[23] N. Lehmann, Argumentation System and Belief Functions. Ph.D. thesis, Department of Informatics, University of Fribourg, Switzerland, 2001. Available at https://doc.rero.ch/record/3230/files/LehmannN.pdf (Accessed 18.01.16).
[24] S. Mauw, M. Oostdijk, Foundations of attack trees, in: D. Won, S. Kim (Eds.), ICISC, LNCS, vol. 3935, Springer, 2005, pp. 186–198.
[25] C. Meadows, A representation of protocol attacks for risk assessment, in: Proceedings of the DIMACS Workshop on Network Threats, New Brunswick, NJ, USA, 1996, pp. 1–10.
[26] P. Mell, K. Scarfone, S. Romanosky, A complete guide to the common vulnerability scoring system version 2.0, 2007. Available at https://www.first.org/cvss/cvss-v2-guide.pdf (Accessed 18.01.16).
[27] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann, 1988.
[28] L. Piètre-Cambacédès, M. Bouissou, Attack and defense modeling with BDMP, in: I. Kotenko, V. Skormin (Eds.), Computer Network Security, LNCS, vol. 6258, Springer, 2010, pp. 86–101. http://dx.doi.org/10.1007/978-3-642-14706-7_7.
[29] N. Poolsappasit, R. Dewri, I. Ray, Dynamic security risk management using Bayesian attack graphs, IEEE Trans. Dependable Secur. Comput. 9 (1) (2012) 61–74, doi:10.1109/TDSC.2011.34.
[30] M. Pouly, Generic solution construction in valuation-based systems, in: C. Butz, P. Lingras (Eds.), CAI, LNAI, vol. 6657, Springer, 2011, pp. 335–346.
[31] M. Pouly, J. Kohlas, Generic Inference – A Unifying Theory for Automated Reasoning, John Wiley & Sons, Inc., 2011.
[32] M. Pouly, NENOK – a software architecture for generic inference, Int. J. Artif. Intell. Tools 19 (2010) 65–99.
[33] X. Qin, W. Lee, Attack plan recognition and prediction using causal networks, in: Proceedings of the ACSAC, 2004, pp. 370–379, doi:10.1109/CSAC.2004.7.
[34] N. Robertson, P. Seymour, Graph minors I: excluding a forest, J. Comb. Theory, Ser. B 35 (1) (1983) 39–61.
[35] M. Sahami, S. Dumais, D. Heckerman, E. Horvitz, A Bayesian approach to filtering junk e-mail, in: Learning for Text Categorization: Papers from the 1998 Workshop, AAAI Technical Report WS-98-05, 1998.
[36] B. Schneier, Attack trees, Dr. Dobb’s J. Softw. Tools 24 (12) (1999) 21–29.
[37] P.P. Shenoy, Valuation-based systems: a framework for managing uncertainty in expert systems, in: L. Zadeh, J. Kacprzyk (Eds.), Fuzzy Logic for the Management of Uncertainty, John Wiley & Sons, Inc., 1992, pp. 83–104.
[38] T. Sommestad, M. Ekstedt, P. Johnson, Cyber security risks assessment with Bayesian defense graphs and architectural models, in: Proceedings of the HICSS-42, 2009a, pp. 1–10.
[39] T. Sommestad, M. Ekstedt, L. Nordström, Modeling security of power communication systems using defense graphs and influence diagrams, IEEE Trans. Power Deliv. 24 (4) (2009b) 1801–1808.
[40] T. Sommestad, M. Ekstedt, H. Holm, The cyber security modeling language: a tool for assessing the vulnerability of enterprise system architectures, IEEE Syst. J. 7 (3) (2013) 363–373.

[41] S.M. Sulaman, K. Weyns, M. Höst, A review of research on risk analysis methods for IT systems, in: Proceedings of the EASE ’13, ACM, 2013, pp. 86–96, doi:10.1145/2460999.2461013.
[42] The Open Group, Risk Taxonomy, Technical Report C081, The Open Group, 2009.
[43] F. van Harmelen, F. van Harmelen, V. Lifschitz, B. Porter, Handbook of Knowledge Representation, Elsevier Science, San Diego, USA, 2007.
[44] L. Wang, T. Islam, T. Long, A. Singhal, S. Jajodia, An attack graph-based probabilistic security metric, in: DAS, in: LNCS, vol. 5094, 2008, pp. 283–296.
[45] L.A. Zadeh, Fuzzy sets as a basis for a theory of possibility, Fuzzy Sets Syst. 1 (1978) 3–28.