Model Checking Exact Cost for Attack Scenarios

Zaruhi Aslanyan(✉) and Flemming Nielson

DTU Compute, Technical University of Denmark, Kongens Lyngby, Denmark
{zaas,fnie}@dtu.dk

Abstract. Attack trees constitute a powerful tool for modelling security threats. Many security analyses of attack trees can be seamlessly expressed as model checking of Markov Decision Processes obtained from the attack trees, thus reaping the benefits of a coherent framework and a mature tool support. However, current model checking does not encompass the exact cost analysis of an attack, which is standard for attack trees.

Our first contribution is the logic *erPCTL* with cost-related operators. The extended logic allows to analyse the probability of an event satisfying given cost bounds and to compute the exact cost of an event. Our second contribution is the model checking algorithm for *erPCTL*. Finally, we apply our framework to the analysis of attack trees.

Keywords: Attack trees · Markov Decision Processes · Probabilistic model checking · Probabilistic temporal logic

1 Introduction

Securing systems and organisations against possible threats is a crucial problem, which becomes increasingly difficult with their growing complexity and their involvement in our everyday life. Tackling this problem demands a thorough investigation of the attack scenarios threatening the system of interest.

Attack trees are a powerful graphical formalism for representing attack scenarios in a structured, hierarchical way by splitting a complex goal into sub-goals and eventually basic attacks [19]. Attack trees are used to analyse attack scenarios. Analyses are performed by considering specific properties of the scenario and augmenting the tree with attributes. Typical attributes include probability and cost of an attack [16], that are computed by propagating the values of the leaves to the root of the tree. For instance, evaluation of the cost, i.e., the sum of the costs of basic actions leading to an attack, is used to identify the cheapest attack or to compare the cost of executing an attack with the attacker's budget.

Attack scenarios with both probability and cost attributes express a combination of nondeterministic and probabilistic behaviour, i.e., an attacker has the nondeterministic choice of performing a basic action and paying the corresponding cost, while the performed basic action succeeds with a certain probability. Hence, it is natural to construe the corresponding attack trees as Markov Decision Processes (MDPs). On this line, many security analyses of attack trees developed in the literature can be seamlessly expressed as *probabilistic model checking* problems. This approach allows to reap the benefits of a coherent framework – the many developments in the area of probabilistic model checking – and a mature tool support.

In this context, Probabilistic Computation Tree Logic with rewards [13] (*rPCTL*), an extension of CTL [9], can express many security properties of interest. In particular, probabilistic model checking *rPCTL* [11] allows to establish the probability of certain events occurring and a reward associated with them, and therefore can encode analyses developed ad hoc for attack trees. However, *rPCTL* only reasons about *expected cost*, i.e., the sum of the rewards along a path multiplied with probabilities. *Exact cost* properties, which reason about the sum of the costs along the path, are instead useful when studying attackers with fixed resources as is typical for attack trees, but cannot be captured in *rPCTL*.

In order to address exact cost analysis, we extend *rPCTL* with cost-related operators. We present a new exact cost operator C which allows to reason about the cost of an event and to express properties such as "what is the minimum cost of a successful attack?" or "is there a way to attack the system by spending no more than the available budget?" Moreover, we consider a general notation for a reward-bounded until operator and define a new operator which evaluates the probability of an event satisfying the given cost bounds. Finally, we develop a model checking algorithm for the extended logic *erPCTL*. The algorithm works on standard MDPs and we show how to transform an attack tree into an MDP.

As a result, *erPCTL* model checking encompasses standard analyses on attack trees, including exact cost analyses, thus offering a unifying framework for different approaches to the analysis of attack trees.

We demonstrate our developments on an example of a cloud environment studied in the project TREsPASS [20].

Related work. Different operators have been investigated to compute different kinds of rewards. For instance, Forejt et al. [11] extend *PCTL* with new operators that are used to evaluate *instantaneous* and *cumulative* expected reward, while operators for expressing *long-run* and *accumulated* expected reward are presented in [1].

Nevertheless, these extensions do not reason about the cumulative reward along the path, i.e., they cannot express properties such as "the probability of reaching the success state is at least 0.7, while the cumulative reward is at most 50". To overcome this limitation, *rPCTL* has been extended with the path operator *reward-bounded until* [6, Chap. 10], [5,8]. The operator verifies if the cumulative reward along a path satisfying the property meets the given bound. Further development of the logic have been proposed to cope with multi-objective model checking [12]. In particular, [21] introduced the concept of quantile for computing expected rewards within given probability bounds.

Elsewhere, various studies have explored a state-based probabilistic model for evaluation of attack and defence scenarios. In particular, Arnold et al. [2] analysed the timing of attack scenarios using continuous-time Markov chains; [17] used priced time automata and the Uppaal model checker to analyse attack trees, but without probabilities. More recently, [14] explored how stochastic timed automata can be used to study attack-defence scenarios where timing plays a central role. Along a similar line, Aslanyan et al. [4] proposed a game-theoretic approach for the formal analyses of complex attack-defence scenarios, allowing to both verify security properties of interest and to synthesise strategies for attacker and defender with respect to some goal.

Organisation of the paper. In Sect. 2 we provide background material on attack trees, Markov Decision Processes and *rPCTL*. The new logic *erPCTL* and the model checking algorithm are presented in Sects. 3 and 4, respectively. In Sect. 5 we describe our proposed translation from attack trees to MDPs and their evaluation. We conclude and discuss future research directions in Sect. 6.

2 Preliminaries

2.1 Attack Trees

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

The abstract syntax of an attack tree t is as follows [3]:

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

A tree is either a leaf or the application of a tree operator to one or two sub-trees. A leaf a is a basic action of the attacker. We denote the set of basic actions by *Act*. The special leaves `true` and `false` represent a trivially-successful and a trivially-failed action, respectively.

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

We associate each basic action $a \in Act$ with a success probability $p(a)$ in case of performing a, $p : Act \to [0, 1]$. Moreover, we associate with each basic action $a \in Act$ a cost c of performing a, $c : Act \to \mathbb{Q}_{\geq 0}$.

2.2 Markov Decision Processes

In the following we recall the basic definitions on MDPs following [6,11].

Definition 1 (MDP). *A Markov Decision Process is a tuple* $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ *where we can find sets* S_A *(of attacker nondeterministic states),* S_P *(of probabilistic states), and* $S_{\circledcirc}$ *(of final states), such that*

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

The probabilistic transition function P describes the probability $P(s, s')$ of a transition from the state s to the state s' in one step. The transition function T is used to solve nondeterminism. For a state s and an action $l \in \alpha$ selected nondeterministically, function T specifies the successor state s', $T(s, l) = s'$. We denote by $\alpha(s)$ the set of enabled actions in the state $s \in S_A$, $\alpha(s) = \{l \in \alpha \mid s \in S_A \text{ and } T(s, l) \text{ is defined}\}$.

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

A *scheduler* is a function $\sigma : S^* S_A \rightarrow \alpha$ that maps a finite path to an action. A scheduler corresponds to one possible resolution of nondeterminism. A scheduler σ is *memoryless* if for any $\pi, \pi' \in S^*$ and $s \in S_A$, $\sigma(\pi s) = \sigma(\pi' s) = \sigma(s)$. We denote by Σ the set of all possible schedulers of an MDP. A probability measure Pr_s^{σ} under a scheduler σ is defined in the standard fashion [15].

We also define a *reward structure* of the form $r : S \rightarrow \mathbb{Q}_{\geq 0}$, which we use to model costs associated with an MDP model. For a finite path $\pi = s_0 s_1 \cdots s_n$ we define its *total cost* as $cost(\pi) = \sum_{s_i \in \pi} r(s_i)$.

2.3 Probabilistic Model Checking

For expressing the probability and cost-related properties of MDPs we shall use the Probabilistic Computation Tree Logic with rewards (*rPCTL*) [6,11].

Definition 2 (rPCTL Syntax). *The syntax of rPCTL is as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie p}(\psi) \mid E^{r}_{\bowtie x}(F\phi)$$
$$\psi ::= X\phi \mid \phi_1 U \phi_2$$

where $a \in AP$ is an atomic proposition, $\bowtie \in \{\geq, >, \leq, <\}$, $p \in \mathbb{Q} \cap [0, 1]$, $x \in \mathbb{Q}_{\geq 0}$, and $r : S \rightarrow \mathbb{Q}_{\geq 0}$ is a reward structure.

A formula defined in *rPCTL* can be either a state formula ϕ evaluated over states, or a path formula ψ evaluated over paths. State formulae are used to express the properties of the model, while path formulae are used only as the parameter of the *probabilistic operator P*. The operator P reasons about the probability of paths satisfying a formula ψ, while the expected rewards operator E is used to evaluate the expected cost of reaching a state that satisfies ϕ.

Path formulae are constructed with the operators *next* and *until*, denoted by X and U, respectively. The path operator U allows to derive the new path operator *eventually*, denoted by F, as follows: $F\phi \equiv true\ U\ \phi$.

Definition 3 (rPCTL Semantics). *Let $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ be an MDP, σ a scheduler of $\mathcal{M}$ and $s \in S$. The satisfaction relation $\models$ of rPCTL for state formulae is defined inductively by:*

$$\begin{array}{lll}
s \models true & \forall s \in S & \\
s \models a & iff & a \in L(s) \\
s \models \neg\phi & iff & s \not\models \phi \\
s \models \phi_1 \wedge \phi_2 & iff & s \models \phi_1 ands \models \phi_2 \\
s \models P_{\bowtie p}(\psi) & iff & Pr_s^\sigma(\psi) \bowtie p \ for\ all\ schedulers\ \sigma \in \Sigma \\
s \models E^r_{\bowtie x}(F\phi) & iff & Exp_s^\sigma(Z^r_{F\phi}) \bowtie x \ for\ all\ schedulers\ \sigma \in \Sigma
\end{array}$$

where $Pr_s^\sigma(\psi) = Pr_s^\sigma(\{\pi \in Paths_s \mid \pi \models \psi\})$, and $Exp_s^\sigma(Z^r_{F\phi})$ denotes the expectation of the random variable $Z^r_{F\phi} : Paths_s \to \mathbb{Q}_{\geq 0}$ under scheduler σ with respect to the probability measure Pr_s^σ,

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

For a path π in $\mathcal{M}$, the satisfaction relation is defined by:

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

Operators P and E. We expand on the semantics of P and E defined above, showing how queries over all schedulers reduce to reasoning over infimum and supremum over all schedulers.

We are interested in computing the *minimum* and the *maximum* probabilities and expected cost for certain formulae to hold. By the result in [6, Ch. 10], we know that there exist memoryless schedulers $\sigma_{\min}$ and $\sigma_{\max}$ that minimise and maximise, respectively, the probabilities of eventually reaching a state that satisfies ϕ:

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

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

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

A similar reasoning holds for the operator E, where we are interested in computing the minimum and the maximum expected cost values over all schedulers. From [11] we know that there exist memoryless schedulers $\sigma_{\min}$ and $\sigma_{\max}$

that minimise and maximise, respectively, the expected cumulative reward of reaching a state that satisfies ϕ:

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

In particular, we can write:

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

We refer the reader to [11] for full details on *rPCTL*. We shall follow the same ideas when defining the model checking algorithms for *erPCTL* in Sect. 4.

3 The Logic erPCTL

In this section we introduce *erPCTL* (Probabilistic Computation Tree Logic with Exact Rewards) for expressing probability as well as cost-related properties of MDPs. The logic *erPCTL* is an extension of the temporal logic *rPCTL*. It allows to reason about the properties over cost measures such as probability within a cost bound or minimum exact cost of an execution.

Definition 4 (erPCTL Syntax). *The syntax of the extended logic erPCTL is defined as follows:*

$$\phi ::= true \mid a \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid P_{\bowtie p}(\psi) \mid E^r_{\bowtie x}(F\phi) \mid P_J(\psi \mid I) \mid C_I(\psi)$$
$$\psi ::= X\phi \mid \phi_1 U \phi_2$$

where $a \in AP$, $\bowtie \in \{\geq, >, \leq, <\}$, $p \in \mathbb{Q} \cap [0,1]$, $J \subseteq [0,1]$ is a closed non-empty interval with rational bounds, $x \in \mathbb{Q}_{\geq 0}$, $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty interval with rational bounds (allowing infinity as upper bound), and $r : S \to \mathbb{Q}_{\geq 0}$ is a reward structure.

Similarly to *rPCTL*, we differentiate between state formulae (ϕ) and path formulae (ψ). The operators inherited from *rPCTL* have the same semantics. The intuitive interpretation of the new operators is as follows. The *probabilistic operator with cost bound* $P_J(\psi \mid I)$ is used to evaluate the probability over the paths satisfying the formula ψ and the cost bound I. The *cost operator* $C_I(\psi)$ is used to evaluate the *exact cost* of the paths satisfying the formula ψ. These operators allow us to check queries like “is the probability of an attack in the cost interval [300,540] smaller than or equal to 0.85?” or “is the cost of all successful attacks greater than 300?”. Such queries cannot be expressed in *rPCTL* if by “cost” we mean “exact cost”.

For simplifying the technical developments, without loss of generality we move from rational numbers to a sparse subset of the rationals for costs.

Proposition 1. *For any finite set $Y \subseteq \mathbb{Q}$ there exists $N \in \mathbb{N}_{>0}$ such that $Y \subseteq \frac{\mathbb{Z}}{N} \subsetneq \mathbb{Q}$, where $\frac{\mathbb{Z}}{N}$ is a set of rational numbers expressed as fractions of the same non-zero denominator N.*

Proposition 2. *The set $\frac{\mathbb{Z}}{N}$ is closed under addition.*

Proposition 3. *All intervals in $\frac{\mathbb{Z}_{\geq 0}}{N}$ are downwards closed (contain their own infimum); all upward bounded intervals in $\frac{\mathbb{Z}_{\geq 0}}{N}$ are upwards closed (contain their own supremum).*

Corollary 1. *For all natural numbers $N \in \mathbb{N}_{>0}$ and sets $Y \subseteq \frac{\mathbb{Z}_{\geq 0}}{N}$ it holds that*

$$\sup(Y) \in \begin{cases} \{-\infty\} & if\, Y = \emptyset \\ \{+\infty\} & if\, Y\, not\, bounded \\ & \quad from\, above \\ Y & otherwise \end{cases} \qquad \inf(Y) \in \begin{cases} \{+\infty\} & if\ Y = \emptyset \\ Y & otherwise \end{cases}$$

Remark 1. The new operators of *erPCTL* are treated similarly to the operators of *rPCTL*, reducing to the computation of infimum and supremum.

The cost operator $C_I(\psi)$ computes the exact cost of reaching a state that satisfies ψ, where the cost values are summed along the path without multiplying with probability, as opposed to the computation of the standard expected cost operator of *rPCTL*. Hence, with the help of Propositions 2 and 3, we only need to consider intervals of the form $[c_1, c_2]$ and $[c, \infty)$.

A similar reasoning holds for the cost interval I in the probabilistic operator with cost bound $P_J(\psi \mid I)$. However, this is not the case for the probability interval J. In the evaluation of the formula $P_J(\psi \mid I)$ probabilities are multiplied along the path, hence we cannot use Corollary 1 as $\frac{\mathbb{Z}}{N}$ is not closed under multiplication. Thus, we limit ourselves to consider only closed intervals J. □

Quantitative extension of *erPCTL*. The operators $P_J(\psi \mid I)$ and $C_I(\psi)$ are validating whether or not the given bound is satisfied. They are not determining the actual probability and cost values. However, as the model checking algorithm is computing such values, we can extend the logic with quantitative operators such as $P_{min=?}(\psi \mid I)$, $P_{max=?}(\psi \mid I)$, $C_{min=?}(\psi)$ and $C_{max=?}(\psi)$. Formally, such formulae can be expressed as numeric state formulae [18].

The semantics of the propositional logic fragment and of probabilistic and reward formulae is defined as for *rPCTL*. Below we will discuss the semantics of the new operators $P_J(\psi \mid I)$ and $C_I(\psi)$.

3.1 Probabilistic Operator with Cost Bound $P_J(\psi \mid I)$

We propose the operator $P_J(\psi \mid I)$ for probability computation with cost bound, where ψ is a path formula, $J \subseteq [0, 1]$ is a closed non-empty probability interval and $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty cost interval of the form $[c_1, c_2]$ or $[c, \infty)$.

Before defining the formal semantics of $P_J(\psi \mid I)$, let us introduce some useful notation. We define the semantics of each path formula with cost interval I as follows:

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

The semantics of the probabilistic operator with cost bound is as follows:

$$s \models P_J(\psi \mid I) \text{ iff } Pr_s^\sigma(\psi \mid I) \in J \text{ for all schedulers } \sigma \in \Sigma$$

where $Pr_s^\sigma(\psi \mid I) = Pr_s^\sigma\{\pi \in Path_s^\sigma \mid \pi \models {}_I\psi\}$.

Intuitively, $P_J(\psi \mid I)$ states that the probability of the paths starting from state s and satisfying the formula ψ and cost bound I is in the interval J.

The formula $P_J(\psi)$ is treated as a special case of the formula $P_J(\psi \mid I)$:

$$P_J(\psi) \equiv P_J(\psi \mid [0, \infty))$$

As mentioned above, the semantics considers all possible schedulers, but we can rephrase it in terms of infimum and supremum. It is immediate that the following equation holds:

$$P_{[p_1,p_2]}(\psi) \equiv P_{\geq p_1}(\psi) \wedge P_{\leq p_2}(\psi)$$

The result holds also in case of cost intervals on both sides of the equation:

$$P_{[p_1,p_2]}(\psi \mid I) \equiv P_{\geq p_1}(\psi \mid I) \wedge P_{\leq p_2}(\psi \mid I)$$

We are interested in computing the *minimum* and the *maximum* probability values within given cost bounds:

$$\begin{aligned} s &\models P_{\geq p}(\psi \mid I) \Leftrightarrow \inf_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \geq p \\ s &\models P_{\leq p}(\psi \mid I) \Leftrightarrow \sup_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \leq p \end{aligned}$$

where the clauses above hold thanks to the reduction of costs from $\mathbb{Q}_{\geq 0}$ to $\frac{\mathbb{Z}_{\geq 0}}{N}$ explained in Propositions 1, 3 and Corollary 1.

3.2 Cost Operator $C_I(\psi)$

We propose the operator $C_I(\psi)$ for exact cost computation, where ψ is a path formula and $I \subseteq \mathbb{Q}_{\geq 0}$ is a non-empty cost interval of the form $[c_1, c_2]$ or $[c, \infty)$.

Before defining the formal semantics of $C_I(\psi)$, let us introduce some useful notation. We define the cost set of an infinite path $\pi = s_0 s_1 \cdots$ for each path formula, denoted by $cost(\pi, \psi)$, as follows:

$$\begin{aligned} cost(\pi, X\phi) &= \{cost(\pi[0\ 1]) \mid \pi[1] \models \phi\} \\ cost(\pi, \phi_1 U \phi_2) &= \{cost(\pi[0 \cdots k]) \mid \pi[k] \models \phi_2 \wedge (0 \leq i < k : \pi[i] \models \phi_1)\} \end{aligned}$$

When the path formula ϕ is not satisfied, then the set is empty. Otherwise, it contains the set of possible costs.

Fact 1. For a path π, a cost interval I and a path formula ψ it holds that

$$\pi \models_I \psi \Leftrightarrow \exists c \in cost(\pi, \psi) : c \in I$$

Fact 2. For a path π and a path formula ψ it holds that

$$\pi \models \psi \Leftrightarrow \pi \models_{[0,\infty)} \psi$$

The semantics of the cost operator is as follows:

$$s \models C_I(\psi) \text{ iff } \forall \sigma \in \Sigma : \forall \pi \in Path_s^\sigma : \forall c \in cost(\pi, \psi) : c \in I$$

Intuitively, $C_I(\psi)$ states that the exact (cumulative) cost of paths starting in state s and satisfying formula ψ under scheduler σ is in the interval I.

In order to verify the cost formula with a general cost interval, we reduce the problem to intervals with only lower and upper bounds according to the following equivalence result:

$$C_{[c_1,c_2]}(\psi) \equiv C_{\geq c_1}(\psi) \wedge C_{\leq c_2}(\psi)$$

Thus, to verify that the exact cost of each path satisfying the formula ψ is in the interval it is sufficient to verify that the exact cost of each path satisfying ψ meets the lower and upper bounds. Again, this problem can be reduced to verify that the infimum (respectively the supremum) cost meets the bound:

$$s \models C_{\geq c}(\psi) \Leftrightarrow (\inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, \psi)) \geq c$$

$$s \models C_{\leq c}(\psi) \Leftrightarrow (\sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} \sup cost(\pi, \psi)) \leq c$$

where the clauses above hold thanks to the reduction of costs from $\mathbb{Q}_{\geq 0}$ to $\frac{\mathbb{Z}_{\geq 0}}{N}$ explained in Propositions 1, 3 and Corollary 1.

4 Model Checking erPCTL

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

For atomic propositions, logical connectives, the probabilistic operator and the reward operator the model checking algorithm is the same as for *rPCTL* [11]. In the following we will discuss the algorithm for the new operators.

4.1 Model Checking the Operator $P_J(\psi \mid I)$

The algorithm for the probabilistic operator with a cost bound is reduced to the computation of the minimum and the maximum values:

$$Sat(P_{\geq p}(\psi \mid I)) = \{s \in S \mid \inf_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \geq p\}$$
$$Sat(P_{\leq p}(\psi \mid I)) = \{s \in S \mid \sup_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I) \leq p\}$$

Here we explain how to determine the minimum probability satisfying the formula in the cost interval, separately for each path formula ψ. The computation for the maximum probability is performed analogously.

The Operator Next ($\psi = X\phi$). First, we consider the operator Next. For computing the minimum probability of satisfying $X\phi$ in the cost interval I,

$$x_s^{min} = \inf_{\sigma \in \Sigma} Pr_s^\sigma(X\phi \mid I)$$

we are solving the following equations:

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

As the sets $S_{\circledcirc}, S_P, S_A$ are disjoint and we identified the set of states for which x_s^{min} equals 0, we can compute x_s^{min} as the unique solution of the system above.

The Operator Until ($\psi = \phi_1 U \phi_2$). Let us now discuss the computation of $P_J(\psi \mid I)$ for the operator Until. Again we are interested in computing $\inf_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I)$ and $\sup_{\sigma \in \Sigma} Pr_s^\sigma(\psi \mid I)$. Before presenting the computation, it is worthwhile noticing that in many real-life scenarios the cost interval I has only an upper bound or a lower bound. Thus, we develop the computation in three different cases with respect to the cost bounds; only an upper bound $[0, c_2]$, only a lower bound $[c_1, \infty)$, *or* both bounds (cost interval) $[c_1, c_2]$.

Case $I = [0, c_2]$. Having only an upper bound c_2 for cost, the values of interest are $\inf_{\sigma \in \Sigma} Pr_s^\sigma(\phi_1 U \phi_2 \mid [0, c_2])$ and $\sup_{\sigma \in \Sigma} Pr_s^\sigma(\phi_1 U \phi_2 \mid [0, c_2])$. First, we define

$$x_s^{min}(\mathfrak{c}) = \inf_{\sigma \in \Sigma} Pr_s^\sigma(\phi_1 U \phi_2 \mid [0, \mathfrak{c}])$$

where $\mathfrak{c} \geq 0$ is the maximum amount that may be spent, where initially $\mathfrak{c} = c_2$. The algorithm follows the corresponding one for the probabilistic operator in *rPCTL*. The difference is that in each considered case (set of states) we examine the cost bound as well. For instance, for the set of states for which $Pr_s^\sigma(\phi_1 U \phi_2)$ is 1 we need to ensure that their costs are within the threshold $\mathfrak{c}$ ($r(s) \leq \mathfrak{c}$): instead, when their costs exceed the threshold ($r(s) > \mathfrak{c}$), these states are in the set for which $Pr_s^\sigma(\phi_1 U \phi_2)$ is 0. Thus, $x_s^{min}(\mathfrak{c})$ can be computed by solving the following equation system:

$$x_s^{min}(\mathfrak{c}) = \begin{cases} 1 & \text{if } s \in Sat(\phi_2) \wedge r(s) \leq \mathfrak{c} \quad (1) \\ 0 & \text{if } s \in S^0_{min} \vee r(s) > \mathfrak{c} \quad (2) \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{min}(\mathfrak{c} - r(s)) & \text{if } s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (3) \\ \min_{l \in \alpha} x_{T(s,l)}^{min}(\mathfrak{c} - r(s)) & \text{if } s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \\ & \quad \wedge\, r(s) \leq \mathfrak{c} \quad (4) \end{cases}$$

where $S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr^\sigma_s(\phi_1 U \phi_2) = 0\}$.

To better understand the system, let us look into the following table:

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\leq \mathfrak{c}$	(1)	(2)	(3)	(4)
$> \mathfrak{c}$	(2)	(2)	(2)	(2)

The first row of the table illustrates the four disjoint sets of states (1–4). The first column shows the cost threshold. As we have an upper cost bound, all costs can be divided into two groups; those that are within the bound ($\leq \mathfrak{c}$) and those that are outside the bound ($> \mathfrak{c}$). The table maps each possible combination of a set of states and cost bound for a state with the corresponding equation.

We consider the minimum value of the equation system in case of multiple solutions. However, the problem is similar to the stochastic shortest path problem, discussed in [7,10], and thus, the equation system has a unique solution.

Case $I = [c_1, \infty)$. Let us present now the case when I has only a lower bound. We are interested in computing the probability of the paths that have cost greater than or equal to c_1. In this case the values of interest are $\inf_{\sigma \in \Sigma} Pr^\sigma_s(\phi_1 U \phi_2 \mid [c_1, \infty))$ and $\sup_{\sigma \in \Sigma} Pr^\sigma_s(\phi_1 U \phi_2 \mid [c_1, \infty))$.

We define

$$x_s^{min}(\mathfrak{c}) = \inf_{\sigma \in \Sigma} Pr^\sigma_s(\phi_1 U \phi_2 \mid [\mathfrak{c}, \infty))$$

where $\mathfrak{c} \in \mathbb{Q}$ is the required minimum amount to be spent. First, we identify the set of states for which $Pr^\sigma_s(\phi_1 U \phi_2)$ is 0:

$$S^0_{min} = \{s \in S \mid \exists \sigma \in \Sigma : Pr^\sigma_s(\phi_1 U \phi_2) = 0\}$$

Observe that having a lower cost bound it might happen that a prefix of a path satisfies the formula but the required cost budget is not reached. We handle this situation by continuing the computation until we find a point where both the formula and the required cost budget are satisfied. Thus, for the set with probability 1 we check the satisfiability of the cost budget. If a state satisfies ϕ_2 ($s \in Sat(\phi_2)$) and the required cost amount $\mathfrak{c}$ ($r(s) \geq \mathfrak{c}$), then we stop the computation. Otherwise, we continue the iteration based on a type of the state.

$$x_s^{min}(\mathfrak{c}) = \begin{cases} 1 & \text{if } s \in Sat(\phi_2) \wedge r(s) \geq \mathfrak{c} \quad (1) \\ 0 & \text{if } s \in S^0_{min} \quad (2) \\ \sum_{s' \in S} P(s,s') \cdot x_{s'}^{min}(\mathfrak{c} - r(s)) & \text{if } s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \vee \quad (3a) \\ & (s \in S_P \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}) \quad (3b) \\ \min_{l \in \alpha} x_{T(s,l)}^{min}(\mathfrak{c} - r(s)) & \text{if } s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \vee \quad (4a) \\ & (s \in S_A \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}) \quad (4b) \end{cases}$$

We present the following table to associate each set of states and cost amount for a state with the corresponding equation.

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\geq \mathfrak{c}$	(1)	(2)	(3a)	(4a)
$< \mathfrak{c}$	(3b), (4b)	(2)	(3a)	(4a)

Observe that for the states in $Sat(\phi_2)$ and cost $< \mathfrak{c}$ there are two equations. The Eqs. (3b) and (4b) correspond to the continuation of the computation in case the formula is satisfied but the required cost amount is not reached.

In case of multiple solutions we consider the minimum value of the equation system above.

Remark 2. Consider an MDP with one state $s \in S_A$ and $r(s) = 0$, like the one presented in Fig. 1. We are interested in computing $\inf_{\sigma \in \Sigma} Pr_s^{\sigma}(true\ U \phi \mid [10, \infty))$. Checking the conditions of the equations above, we can see that the state s satisfies the condition $(4b)$, as $s \in Sat(\phi)$ and $r(s) < \mathfrak{c}$. From the equation system we have that $x_s^{min}(\mathfrak{c}) = x_s^{min}(\mathfrak{c})$, and thus the system above has infinitely many solutions.

For ensuring a unique solution of the equation system we can use the techniques described in [10,11], where the reader is referred for details. The main idea is to modify the MDP by removing states with self-loop and zero cost.

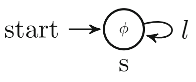

Fig. 1. The MDP example discussed in Remark 2.

General case $I = [c_1, c_2]$. Let us now present the general case, where we have both lower and upper bounds. We are interested in computing $\inf_{\sigma \in \Sigma} Pr_s^{\sigma}(\phi_1 U \phi_2 \mid [c_1, c_2])$ and $\sup_{\sigma \in \Sigma} Pr_s^{\sigma}(\phi_1 U \phi_2 \mid [c_1, c_2])$, where $c_1 \leq c_2$.

We define

$$x_s^{min}(\mathfrak{c}', \mathfrak{c}'') = \inf_{\sigma \in \Sigma} Pr_s^{\sigma}(\phi_1 U \phi_2 \mid [\mathfrak{c}', \mathfrak{c}''])$$

where $\mathfrak{c}' \in \mathbb{Q}$ is the required minimum amount to be spent and $\mathfrak{c}'' \geq 0$ is the maximum amount that may be spent. Similarly to previous cases, we examine the cost amount for each set of states. For instance, the states in the set with probability 1 should satisfy not only the formula ϕ_2 but also be in the cost interval $[\mathfrak{c}', \mathfrak{c}'']$, while the states above the cost interval $(r(s) > \mathfrak{c}'')$ should be in the set with probability 0.

$$x_s^{min}(\mathfrak{c}', \mathfrak{c}'') = \begin{cases} 1 \quad \text{if} \;\; s \in Sat(\phi_2) \wedge r(s) \in [\mathfrak{c}', \mathfrak{c}''] & (1) \\ 0 \quad \text{if} \;\; s \in S^0_{min} \vee r(s) > \mathfrak{c}'' & (2) \\ \sum_{s' \in S} P(s, s') \cdot x_{s'}^{min}(\mathfrak{c}' - r(s), \mathfrak{c}'' - r(s)) & \\ \qquad \text{if } (s \in S_P \backslash (S^0_{min} \cup Sat(\phi_2)) \wedge r(s) \leq \mathfrak{c}'') & (3a) \\ \qquad \vee \; (s \in S_P \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}') & (3b) \\ \min_{l \in A} x_{T(s,l)}^{min}(\mathfrak{c}' - r(s), \mathfrak{c}'' - r(s)) & \\ \qquad \text{if } (s \in S_A \backslash (S^0_{min} \cup Sat(\phi_2)) \wedge r(s) \leq \mathfrak{c}'') & (4a) \\ \qquad \vee \; (s \in S_A \cap Sat(\phi_2) \wedge r(s) < \mathfrak{c}') & (4b) \end{cases}$$

We present the following table to associate each set of states and cost amount for a state with the corresponding equation.

	$Sat(\phi_2)$	S^0	$s \in S_P \backslash (S^0 \cup Sat(\phi_2))$	$s \in S_A \backslash (S^0 \cup Sat(\phi_2))$
$\geq \mathfrak{c}'$ and $\leq c''$	(1)	(2)	(3a)	(4a)
$> \mathfrak{c}''$	(2)	(2)	(2)	(2)
$< \mathfrak{c}'$	(3b), (4b)	(2)	(3a)	(4a)

Differently from the previous cases, here the cost values are divided into three groups; those that are inside the cost interval, those that are below the cost interval and those that are above the cost interval.

We consider the minimum solution of the equation system above. Like in the case with only a lower bound, here as well the system above does not have a unique solution. Again, we can use the techniques presented in [10,11] and modify the MDP in order to ensure a unique solution.

4.2 Model Checking the Operator $C_I(\psi)$

Let us now present the model checking algorithm for the operator $C_I(\psi)$. We need to compute the exact cost of the paths satisfying the formula ψ and check whether they are in I. The procedure reduces to the computation of the minimum and the maximum values depending on the bound:

$$Sat(C_{\geq c}(\psi) = \{s \in S \mid (\inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, \psi)) \geq c\}$$

$$Sat(C_{\leq c}(\psi) = \{s \in S \mid (\sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^\sigma} \sup cost(\pi, \psi)) \leq c\}$$

In the following we explain how to determine the minimum and maximum cost, separately for each path formula ψ.

The Operator Next ($\psi = X\phi$)**.** We start with the computation of the minimum cost for the operator next. The minimum cost

$$y_s^{min} = \inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^\sigma} \inf cost(\pi, X\phi)$$

for each state s can be computed by means of the following equations:

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

The equation system above has a unique solution.

The computation of the maximum cost is performed analogously.

The Operator Until ($\psi = \phi_1 U \phi_2$)**.** Similarly to the computations of the operator U for $P_J(\psi \mid I)$, the minimum cost of a path satisfying the formula $\phi_1 U \phi_2$ can be computed recursively. For computing the minimum cost of a path, we stop the first time ϕ_2 is satisfied, i.e., we compute the cost of the path $\pi = s_0 \cdots s_j$ where $j = \min\{j \mid \pi[j] \models \phi_2 \wedge (\forall k < j : \pi[k] \models \phi_1)\}$.

Thus, the computation of

$$y_s^{min} = \inf_{\sigma \in \Sigma} \inf_{\pi \in Path_s^{\sigma}} cost^{inf}(\pi, \phi_1 U \phi_2)$$

corresponds to solving the following equations:

$$y_s^{min} = \begin{cases} r(s) + \min(\{y_{s'}^{min} \mid s \models \phi_1 \wedge P(s,s') > 0\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if} \quad s \in S_P \vee s \in S_{\circledcirc} \\ r(s) + \min(\{y_{s'}^{min} \mid s \models \phi_1 \wedge T(s,l) = s'\} \cup \{0 \mid s \in Sat(\phi_2)\}) \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if} \quad s \in S_A \end{cases}$$

Note that the system is solved in the set $\mathbb{Q} \cup \{-\infty, +\infty\}$, where $\inf \emptyset = \min \emptyset = +\infty$. Thus, when there is no state satisfying the formula (the set of solutions is empty), the system returns $+\infty$. The equation system above might give more than one solution. In this case we consider the maximum one.

The equations for computing $\sup_{\sigma \in \Sigma} \sup_{\pi \in Path_s^{\sigma}} cost^{sup}(\pi, \phi_1 U \phi_2))$ can be obtained by replacing "min" with "max" in the system above. Observe that in the computation of the maximum cost we do not stop the first time ϕ_2 is satisfied but we continue till the last time it is satisfied.

For a finite MDP $\mathcal{M}$ and an *erPCTL* formula ϕ, we expect the complexity of the model checking algorithm to be polynomial in the size of $\mathcal{M}$ and linear in the size of the formula ϕ.

5 Analysis of Attack Trees

So far we have defined the extended logic *erPCTL* and presented a model checking algorithm for it. We now formalise the evaluation of attack scenarios using probabilistic model checking of *erPCTL*. The basic idea is to transform attack trees into MDPs, as an attack tree with probability and cost attributes encodes behaviour encompassing both probabilistic and nondeterministic features.

Before presenting the translation, it is worthwhile noticing that in an attack tree the order in which the basic actions are performed is not fixed. However, in the MDP this needs to be made explicit. Since we assume that the basic actions of a tree are independent, we will also assume any linear order of the set of basic actions Act.

5.1 From Attack Trees to MDPs

We construct an MDP $\mathcal{M} = (S, \alpha, P, T, s_0, AP, L)$ from an attack tree t according to Table 1, where $s_0 = \mathsf{construct}[t](Act, \emptyset, \emptyset)$. The set of states S is the disjoint union of sets S_A, S_P, $S_\circledcirc$, $S = S_A \uplus S_P \uplus S_\circledcirc$, while the set of actions is $\alpha = \{Y, N\}$. The transition functions P, T and the labelling function L are constructed according to Table 1, and the set of atomic propositions is $AP = Act \uplus \{success, failure\}$. The target state space S is exponential in the size of t, as often the size of a model is exponential in the size of the description that gives rise to the model.

The call $\mathsf{construct}[t](Act, \emptyset, \emptyset)$ of the recursive function $\mathsf{construct}$, defined in Table 1, constructs an MDP from t. The procedure first constructs all nondeterministic transitions of the target MDP, and then the probabilistic transitions.

Throughout the evaluation of the function we assume to have an attack tree t as a global parameter. At each step of the evaluation of $\mathsf{construct}[t](A, Done, Succ)$ the first parameter A corresponds to the remaining set of attacker's basic actions that has still to be evaluated, the second parameter $Done$ is the set of attempted actions, and the last parameter $Succ$ is the set of attempted and succeeded actions. The construction function is structurally defined over the set of basic actions as explained below.

Table 1. The construction of an MDP from an attack tree

$\mathsf{construct}[t](A \cup \{a\}, Done, Succ) =$ new state $s \in S_A$ with:

$L(s) = \{a\}, T(s, Y) = s', T(s, N) = s''$ where:

$s' = \mathsf{construct}[t](A, Done \cup \{a\}, Succ)$

$s'' = \mathsf{construct}[t](A, Done, Succ)$

and: $r(s') = c(a), r(s'') = 0$

$\mathsf{construct}[t](\emptyset, Done \cup \{a\}, Succ) =$ new state $s \in S_P$ with:

$L(s) = \{a\}, P(s, s') = p(a), P(s, s'') = 1 - p(a)$ where:

$s' = \mathsf{construct}[t](\emptyset, Done, Succ \cup \{a\})$

$s'' = \mathsf{construct}[t](\emptyset, Done, Succ)$

and: $r(s') = r(s'') = 0$

$\mathsf{construct}[t](\emptyset, \emptyset, Succ) =$ new state $s \in S_\circledcirc$ with:

$L(s) = \{success\}$ if $[\![t]\!](Succ)$ and $\{failure\}$ otherwise

If the set of remaining actions contains the action a, $A = A' \cup \{a\}$, we create a nondeterministic state in the MDP labelled with a and with two outgoing transitions. One transition corresponds to attempting a and is labelled with the action Y and cost $c(a)$, while the other transition corresponds to not attempting a and is labelled with the action N and cost 0. The successors of the state are constructed recursively, by calling $\mathsf{construct}[t](A', \mathit{Done} \cup \{a\}, \mathit{Succ})$ and $\mathsf{construct}[t](A', \mathit{Done}, \mathit{Succ})$, respectively.

If the set of remaining actions is empty, $A = \emptyset$, while the set of attempted actions contains the action a, $\mathit{Done} = \mathit{Done}' \cup \{a\}$, we create a probabilistic state labelled with a and with the outgoing transitions corresponding to the success and the failure of the attempted action a. We label these transitions with probabilities $p(a)$ and $1-p(a)$, and construct the successor of the state by calling $\mathsf{construct}[t](\emptyset, \mathit{Done}', \mathit{Succ} \cup \{a\})$ and $\mathsf{construct}[t](\emptyset, \mathit{Done}', \mathit{Succ})$, respectively.

If both the set of remaining actions and the set of attempted actions are empty, $A = \emptyset, \mathit{Done} = \emptyset$, then we are at the end of the procedure. We create a final state and label it with the result of the evaluation of t over the success of the basic actions, $[\![t]\!](\mathit{Succ})$, where $[\![t]\!]$ is the Boolean formula of which the tree t is a parse tree and the atoms in the formula are tt if the corresponding actions are in the set Succ and ff otherwise.

Observe that MDPs constructed from attack trees are finite and acyclic.

Example. Let us introduce an example that we will develop in the following. We consider a small fragment of the real-life scenario of cloud environment studied in the project TREsPASS [20], where an attacker wants to steal money from a cardholder by forcing him/her to pay for fake services. In order to do so, the attacker needs to threaten or blackmail. For a successful threatening the attacker should threaten the cardholder and access the household. In order to succeed in blackmailing the attacker should collect necessary information and blackmail the cardholder. The corresponding attack tree is shown in Fig. 2, where we label the leaves to refer to them easily.

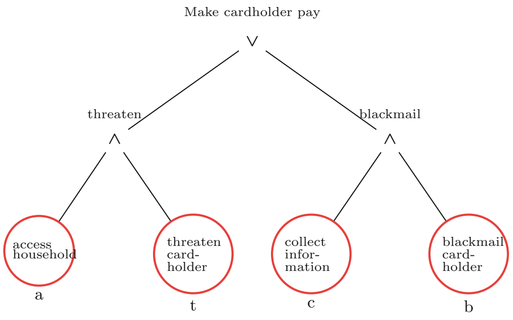

Fig. 2. An attack tree for forcing the cardholder to pay

The probability of success and cost values for the basic actions of the tree are given in Table 2, and we consider the following linearly ordered set for basic

Table 2. Probabilities and costs for the basic actions in the example

Label	Name of the node	Probability	Cost
a	Access household	0.6	70
t	Threaten cardholder	0.3	30
c	Collect information	0.55	50
b	Blackmail cardholder	0.2	30

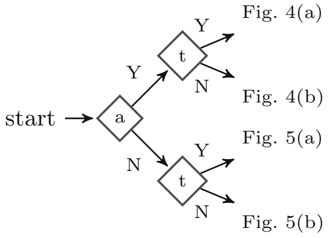

Fig. 3. The MDP $\mathcal{M}_t$ constructed from the tree t. Due to the size of $\mathcal{M}_t$ we have split it into subfigures

actions, $Act = \{\text{a, t, c, b}\}$. Determining realistic estimates for the probabilities and costs for basic actions is a research topic in itself and is outside the scope of this work.

Let us construct an MDP from the tree t displayed in Fig. 2, by following the rules described in Table 1. First, all nondeterministic transitions are constructed, and then the probabilistic ones. The resulting MDP M_t is presented in Figs. 3, 4 and 5.

5.2 Evaluation of Attack Scenarios

In the previous section we have proposed a translation from attack trees to MDPs. The main focus of this section is to show how to evaluate security properties by means of model checking *erPCTL*. We start with a discussion of the security properties of interest and then discuss their representation in *erPCTL*.

Security properties. Attributes to basic actions play an important role in the analysis of an attack scenario. They are used to express various properties of interest. In this paper we characterise the basic actions of an attack scenario with the success probability and the cost of performing the action. The properties we study range from quantitative to qualitative as well as from one-objective to multiple-objective properties. We formalise them in *erPCTL*.

We study probability-related properties such as "is the success probability of an attack greater than or equal to 0.2?" or "what is the maximum probability of an attack?". The first qualitative property is expressed in *erPCTL* as the formula $P_{\geq 0.2}(F\,success)$, while the second quantitative property is express as the formula $P_{max=?}(F\,success)$.

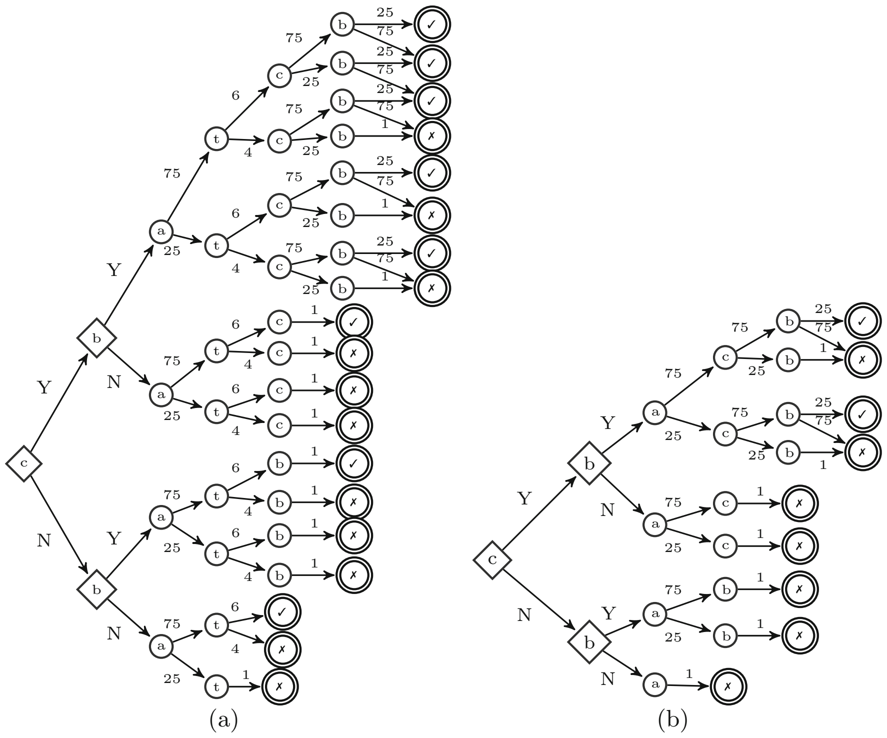

Fig. 4. A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t, where ✓ stands for *success* and ✗ stands for *failure*

The characterisation of basic actions with cost allows to compute the cheapest attack, phrased as "what is the minimum cost of an attack?". This property is expressed by the formula $C_{min=?}(F success)$. Moreover, having a cost budget c for the attacker, we can study more specific properties. For example, the attacker might want to know if whatever he/she does the cost of all successful attacks is in I, i.e., whether the attacker can always succeed by spending no more than the budget. We can express such property with the question "is the cost of all successful attacks within the budget c?" and phrase it in *erPCTL* with the formula $C_{[0,c]}(F success)$. On the other hand, a defender who is looking at the attack scenario might want to verify whether all successful attacks are outside the attacker's budget, i.e., "is the cost of all successful attacks greater than or equal to c?". The corresponding formula is $C_{[c,\infty)}(F success)$.

So far the cost-related properties we considered are evaluated over all attacks. However, the (clever) attacker might want to know if there exists at least one successful attack within the budget c. We can express this property as the formula $\neg C_{[c,\infty)}(F success)$.

Our framework allows also to study multiple-objective properties such as "is there an attack with success probability at least 0.4 and cost at most 1500?" or

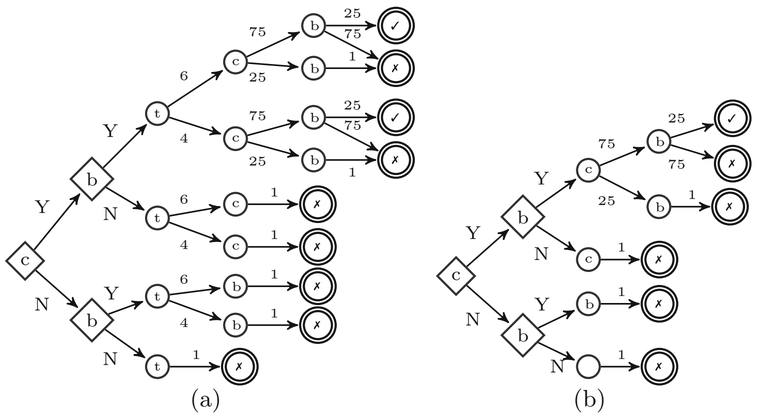

Fig. 5. A fragment of the MDP $\mathcal{M}_t$ constructed from the tree t

"what is the maximum probability of an attack with cost at most 1500?". They expressed by the formulae $P_{\geq 0.4}(F success \mid [0, 1500])$ and $P_{max=?}(F success \mid [0, 1500])$, respectively.

Example. Consider the MDP given in Fig. 3. We exploit the model checking algorithm of *erPCTL* to verify the security properties mentioned above. For example, the verification of the probabilistic query $P_{\geq 0.2}(F success)$ returns "false", meaning that there exists at least one attack with success probability less than 0.2. We compute the maximum success probability of an attack with the query $P_{max=?}(F success)$, which is 0.549.

Assume the attacker has a cost budget equal to 1500 and let us check whether all successful attacks are within the budget. The query $C_{\leq 1500}(F success)$ returns "false" meaning that there exists a successful attack with cost greater than the budget. We can also compute the minimum cost of a successful attack with the query $C_{min=?}(F success)$, which is 900.

Finally, we verify multi-objective queries, such as $P_{\geq 0.4}(F success \mid [0, 1500])$ and $P_{max=?}(F success \mid [0, 1500])$. The first property evaluates to "false" meaning that there is no attack with probability at least 0.4 and cost at most 1500, while the second property computes the maximum probability of an attack with cost at most 1500, which is 0.18.

6 Conclusion

Attack trees constitute a useful tool to study attack scenarios and to present the behaviour of an attacker in an intuitive way. Security attributes, associated with basic actions of attack trees, provide the basis for various types of analysis. Many analyses focus on the evaluation of an exact cost, i.e., the sum of the costs of basic actions leading to an attack, which allows to investigate the required resources for an attack. Exact cost analyses are used to identify the cheapest attacks or to verify that a successful attack is within an attacker's budget.

Probabilistic model checking is used to verify automatically whether or not a model satisfies properties of interest specified in *rPCTL*. This logic allows to reason about probability and expected rewards, thus encompassing many security properties typically investigated on attack trees. However, *rPCTL* operators cannot evaluate *exact cost*, preventing to rely on probabilistic model checking as a general framework for attack trees analysis.

In this work, we extended *rPCTL* with cost-related operators. In the extended logic *erPCTL* the defined cost-bounded probabilistic operator evaluates the probability of an event satisfying the given cost bounds, while the exact cost operator analyses the cost of the occurrence of an event. Moreover, we developed a model checking algorithm for the novel operators of *erPCTL*. The algorithm works on standard MDPs, that we obtain from attack trees with a transformation detailed in the paper.

Since we have considered a standard attack tree model, most properties of interest concern reachability of a success state, i.e., the path formula has the form $(F\,success)$. However, other path formulae allow to capture more elaborate scenarios. For instance, considering the notion of a detected attack, then we would verify whether the property $(\neg detected\ U\ success)$ holds or not.

The benefit of our contribution is two-fold. On the one hand, we have described a unifying framework where different analyses of attack trees can be seamlessly encoded, studied, and compared. On the other hand, the tool support available for model checking problems can be leveraged to analyse attack trees, and perhaps the features of the tools can inspire new interesting analyses.

As future work, we plan to provide a proof-of-concept implementation of our framework. Moreover, strategy synthesis seems a natural extension to the framework, so as to obtain explicitly what are the attacks that satisfy a given *erPCTL* property, if any. As for improving the model checking algorithm, the formula $C_I(\psi)$ could be verified by resorting to weighted-CTL techniques, as probabilities play no role. Finally, it would be worth moving from attack trees and MDPs to attack-defence trees and games, and propose a logic for evaluating exact cost properties of an attack-defence scenario.

Acknowledgment. Part of the research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 318003 (TREsPASS). Special thanks also go to Roberto Vigo for valuable comments.

References

1. Andova, S., Hermanns, H., Katoen, J.-P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) FORMATS 2003. LNCS, vol. 2791, pp. 88–104. Springer, Heidelberg (2004). doi:10.1007/978-3-540-40903-8_8
2. Arnold, F., Hermanns, H., Pulungan, R., Stoelinga, M.: Time-dependent analysis of attacks. In: Abadi, M., Kremer, S. (eds.) POST 2014. LNCS, vol. 8414, pp. 285–305. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54792-8_16

3. Aslanyan, Z., Nielson, F.: Pareto efficient solutions of attack-defence trees. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 95–114. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46666-7_6
4. Aslanyan, Z., Nielson, F., Parker, D.: Quantitative verification and synthesis of attack-defence scenarios. In: IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, 27 June–1 July 2016, pp. 105–119 (2016). http://dx.doi.org/10.1109/CSF.2016.15
5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: On the logical characterisation of performability properties. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 780–792. Springer, Heidelberg (2000). doi:10.1007/3-540-45022-X_65
6. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008). http://mitpress.mit.edu/9780262026499
7. Bertsekas, D.P., Tsitsiklis, J.N.: An analysis of stochastic shortest path problems. Math. Oper. Res. **16**(3), 580–595 (1991). http://dx.doi.org/10.1287/moor.16.3.580
8. Chen, T., Forejt, V., Kwiatkowska, M.Z., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods Syst. Des. **43**(1), 61–92 (2013)
9. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching-time temporal logic. In: Logics of Programs, Workshop, pp. 52–71. Yorktown Heights, New York, May 1981. http://dx.doi.org/10.1007/BFb0025774
10. De Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford, CA, USA (1998). AAI9837082
11. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21455-4_3
12. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 112–127. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19835-9_11
13. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. **6**(5), 512–535 (1994). http://dx.doi.org/10.1007/BF01211866
14. Hermanns, H., Krämer, J., Krčál, J., Stoelinga, M.: The value of attack-defence diagrams. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 163–185. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49635-0_9
15. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains, 2nd edn. Springer, New York (1976)
16. Kordy, B., Mauw, S., Schweitzer, P.: Quantitative questions on attack–defense trees. In: Kwon, T., Lee, M.-K., Kwon, D. (eds.) ICISC 2012. LNCS, vol. 7839, pp. 49–64. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37682-5_5
17. Kumar, R., Ruijters, E., Stoelinga, M.: Quantitative attack tree analysis via priced timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 156–171. Springer, Heidelberg (2015). doi:10.1007/978-3-319-22975-1_11
18. Nielsen, B.F., Nielson, F., Nielson, H.R.: Model checking multivariate state rewards. In: Seventh International Conference on the Quantitative Evaluation of Systems, QEST 2010, Williamsburg, Virginia, USA, 15–18 September 2010, pp. 7–16 (2010). http://dx.doi.org/10.1109/QEST.2010.10
19. Schneier, B.: Attack Trees: Modeling Security Threats. Dr. Dobb's J. Softw. Tools **24**(12), 21–29 (1999). http://www.ddj.com/security/184414879

20. The TREsPASS Project (2014). https://www.trespass-project.eu
21. Ummels, M., Baier, C.: Computing quantiles in markov reward models. In: Pfenning, F. (ed.) FoSSaCS 2013. LNCS, vol. 7794, pp. 353–368. Springer, Heidelberg (2013). doi:10.1007/978-3-642-37075-5_23