Towards a Formal Verification of Attack Graphs

Davide Catta[1,†], Jean Leneutre[1] and Vadim Malvone[1]

[1]LTCI, Télécom Paris, Institut Polytechnique de Paris, Paris, France

Abstract

In this perspective paper, we propose different formalizations of games that are played over Attack Graphs between an Attacker and a Defender. In all such games we propose a formal approach (such as logics and automata theory) to check whether the Attacker has a strategy to win the game.

Keywords

Attack Graphs, Sabotage Games, Logics in Games

1. Introduction

Modern systems are inherently complex and security plays a crucial role. The main challenge in developing a secure system is to come up with tools able to detect vulnerability and unexpected behaviors at a very early stage of its life-cycles. These methodologies should be able to measure the grade of resilience to external attacks. Crucially, the cost of repairing a system flaw during maintenance is at least two order of magnitude higher, compared to a fixing at an early design stage [1]. In the past fifty years several solutions have been proposed and a story of success is the use of *formal methods* techniques [1]. They allow checking whether a system is correct by formally checking whether a mathematical model of it meets a formal representation of its desired behavior. Recently, classic approaches such as *model checking* and automata-theoretic techniques, originally developed for monolithic systems [2, 3], have been meaningfully extended to handle *multi-agent systems* [4, 5, 6, 7, 8]. These are systems that encapsulate the behavior of two or more rational agents interacting among them in a cooperative or adversarial way, aiming at a designed goal [9].

In system security checking, a malicious attack can be seen as an attempt of an Attacker to gain an unauthorized resource access or compromise the system integrity. In this setting, *Attack Graph* [10] is one of the most prominent attack model developed and receiving much attention in recent years. This encompasses a graph where each state represents an Attacker at a specified network location and edges represent state transitions, i.e., attack actions by the Attacker. In this paper, we sketch three different methodologies to reason about Attack Graphs by introducing game models in which a player (called Attacker) travels through adjacent edges of an Attack Graph and tries to reach a certain target state, and another player (called Defender)

IPS-RiCeRcA-SPIRIT 2022: 10th Italian Workshop on Planning and Scheduling, RiCeRcA Italian Workshop, and SPIRIT Workshop on Strategies, Prediction, Interaction, and Reasoning in Italy.

[†]These authors contributed equally.

davide.catta@telecom-paris.fr (D. Catta); jean.leneutre@telecom-paris.fr (J. Leneutre); vadim.malvone@telecom-paris.fr (V. Malvone)

tries to prevent to Attacker to reach his goal by dynamically deploying countermeasures. All the game models that we will introduce can be handled by means of formal methods, i.e., to reason about such games we will use logical methods or automata-theoretic approaches.

2. Attack Graphs

The term Attack Graph has been first introduced by Phillips and Swiler [11]. Attack Graphs represent the possible sequence of attacks in a system as a graph. An Attack Graph may be generated by using the following information: a description of the system architecture (topology, configurations of components, etc.), the list of the known vulnerabilities of the system and the Attacker's profile (his capabilities, password knowledge, privileges etc.) and attack templates (Attacker's atomic action, including preconditions and postconditions). An attack path in the graph corresponds to a sequence of atomic attacks. Several works have developed this approach and each work introduces its own Attack Graph model with its specificity, and thus there is no standard definition of an Attack Graph, see [12] for a survey. However, all introduced models can be mapped into a *canonical Attack Graph* as introduced in [13]. It is a labelled oriented graph, where:

1. each node represents both the state of the system (including existing vulnerabilities) and the state of the Attacker including constants (Attacker skills, financial resources, etc.) and variables (knowledge on the network topology, privilege level, obtained credentials, etc.);
2. each edge represents an action of the Attacker (a scan of the network, the execution of an exploit based on a given vulnerability, access to a device, etc.) that changes the state of the network or the states of the Attacker; an edge is labelled with the name of the action (several edges of the Attack Graph may have the same label).

We recall that a *rooted* Kripke structure M is given by a non-empty set S of states, a designated initial state s_0, a finite-non empty set Σ of labels, a ternary relation $R \subseteq S \times \Sigma \times S$ and a labeling function V that maps any element of S to a set of atomic proposition. By abstracting all the data in the above discussion, one can define an Attack Graph as follows.

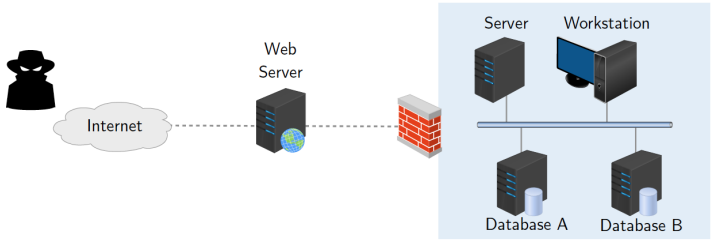
Definition 1. *Let Σ be a finite set of labels, and $\mathcal{P}$ a finite set of atomic propositions, such that* $\mathsf{win} \in \mathcal{P}$. *An **Attack Graph** is a tuple $AG = \langle S, s_0, \Sigma, R, V, W\rangle$ where:*

- *$\langle S, s_0, \Sigma, R, V\rangle$ is a rooted Kripke structure where the set S of states is finite and $V(s) \subseteq \mathcal{P}$ for any $s \in S$;*
- *W is a non-empty subset of S such that* $\mathsf{win} \in V(s)$ *for all $s \in W$.*

The set W represents the set of target states of the Attacker.

Let us make things more concrete. Consider the following scenario: an enterprise has a local area network (LAN) that features a *Server*, a *Workstation* and two databases A and B. The LAN also provides a *Web Server*. Internet's access to the LAN are controlled by a firewall. Such scenario is depicted in Figure 1. Suppose that we know some vulnerabilities and that we have

established that a malevolent user can make the attacks listed in the Table of Figure 1, e.g., by making att_2, an Attacker can exploit a vulnerability related to the *Server*: as a precondition the Attacker needs to have root access to the *Web Server* and, as a postcondition, he will obtain root access to the *Server*. Then we can construct an Attack Graph built from this set of atomic attacks and collecting possible attack paths as depicted in Figure 1[1]. The Attacker's initial state is a node in the Attack Graph. Let us suppose that the Attacker is in state s_1 and wants to reach state s_4. To get to this target, he can perform the sequences of atomic attacks att_2, att_4 or att_3, att_2, att_4.



Attack	Location	Precondition	Postcondition	Counter measure
att_1	Web Server		$web\_server : root$	_
att_2	Server	$web\_server : root$	$server : root$	c_2
att_3	Workstation	$web\_server : root$	$password : 1234$	_
att_4	Database A	$server : root$	$databaseA : root$	c_4
att_5	Database B	$server : root \wedge$ $password : 1234$	$databaseB : root$	c_5

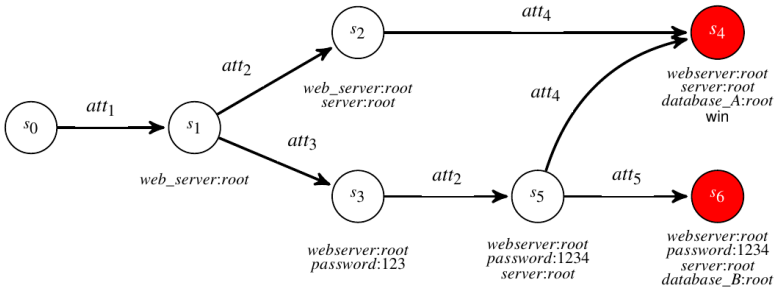


Figure 1: An illustrating LAN architecture example, a list of Atomic attacks and countermeasures over the LAN, and an Attack Graphs generated by such Atomic attacks.

[1]This Attack Graph is not complete w.r.t. our previous description, since some possible sequences of atomic attacks are not listed: for instance $att_1, att_2, att_3, att_5$ are not taken into account.

3. Attack Graph Games

In the previous section, we saw that an Attack Graph represents a sequence of possible actions made by an Attacker to reach a specific goal. Let us add another character to this story, the Defender, whose objective is to counter the attack. Suppose that she has the power to dynamically deploy a predefined set of countermeasures: for instance by reconfiguring the firewall filtering rules, or patching some vulnerabilities, that is by removing one or several preconditions of an atomic attack. A given countermeasure c will prevent the Attacker from longing a given attack att. One can imagine a game played over the Attack Graph in which the Attacker tries to reach one of its goal states, and the Defender tries to prevent the Attacker from reaching a goal state by deploying countermeasures. Given such a game, one natural question to ask is the following: is there a winning Attacker Strategy? We will now define three possible settings in which such games can be formalized.

3.1. Attack Graph Games & (Subset) Sabotage Modal Logic

Above, we introduced a Defender, whose power is to deploy certain countermeasures, and we introduced the concept of game over an Attack Graph. To make things less abstract, consider a two player turn-based game over Attack Graph in which: the Defender starts the game by selecting a certain countermeasure c. By choosing such a countermeasure, she deletes a subset of edges of the Attack Graph: all the edges that are labeled by c. The Attacker takes his turn and moves from s_0 to one of its successor along the edges that have not being erased (if any). The game evolves following this pattern. The Attacker wins if he can reach one of the states in W in a finite number of moves, the Defender wins otherwise. A run in such a game is nothing more than a run in a insidious sabotage game. Sabotage games were introduced by van Benthem in 2005 [14], with the aim of studying the computational complexity of a special class of graph-reachability problems. Namely, graph reachability problems in which the set of edges of the graph became thinner and thinner as long as a path of the graph is constructed. To reason about sabotage games, van Benthem introduced Sabotage Modal Logic. Such Logic is obtained by adding to the $\Diamond$-modality of classical modal logic another modality $\blacklozenge$.

Let G be a directed graph and s one of its vertex (or states); the intended meaning of a formula $\blacklozenge\varphi$ is " $\blacklozenge\varphi$ is true at a state s of G iff φ is true at s in the graph obtained by G by erasing an edge e". Our sabotage games differs from the one introduced by van Benthem because one of the players can erase *an entire subset of edges of a given graph.* To reason about such games, in [15] we introduce a variant of Sabotage Modal Logic that we call, for lack of wit, Subset Sabotage Modal Logic (SSML, for short). The logic SSML is obtained by adding a modality $\blacklozenge_a^{\subseteq}$ to the language of classical modal logic. Being more precise, let $\mathcal{P}$ be a non-empty set of atomic propositions and Σ a finite, non empty set of labels. Formulas of SSML are defined by the following grammar

$$\varphi ::= p \mid \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Diamond_a\varphi \mid \blacklozenge_a^{\subseteq}\varphi$$

where p is any atomic proposition and a any label. One can define the boolean connectives $\vee$ and $\rightarrow$ in the usual way, $\Box_a\varphi$ as $\neg\Diamond_a\neg\varphi$, and $\blacksquare_a^{\subseteq}\varphi$ as $\neg\blacklozenge_a^{\subseteq}\neg\varphi$.

The intended meaning of a formula $\blacklozenge^{\subset}\varphi$ is " $\blacklozenge_a^{\subset}\varphi$ is true at a state s of a directed graph G iff φ is true at s in the graph G' that is obtained from G by erasing a non-empty set of edges that are labeled by a". We now precisely define the semantics of SSML. If $M = \langle S, s_0, \Sigma, R, V\rangle$ is a Kripke structure and $E \subseteq R$, we write $M \setminus E$ to denote the Kripke structure obtained by erasing the subset E from the relation R. If $e = (s, a, s') \in R$ we say that e is a *labeled edge* or simply an edge, and that a is the label of e. We denote by R_a the subset of labeled edges of R whose label is a, that is $R_a = \{e \in R \mid e \in S \times \{a\} \times S\}$.

The notion of satisfaction of a formula φ at a given state s of a Kripke structure M (written $M, s \models \varphi$) is inductively defined as follows:

$M, s \models \top$ for all $s \in S$;

$M, s \models p$ iff $p \in V(s)$;

$M, s \models \neg\varphi$ iff not $M, s \models \varphi$ (notation $M, s \not\models \varphi$);

$M, s \models \varphi_1 \wedge \varphi_1$ iff $M, s \models \varphi_1$ and $M, s \models \varphi_2$;

$M, s \models \Diamond_a \varphi$ iff there is a $s' \in S$ such that $(s, a, s') \in R$ and $M, s' \models \varphi$;

$M, s \models \blacklozenge_a^{\subset} \varphi$ iff there is a non-empty $E \subseteq R_a$ such that $M \setminus E, s \models \varphi$.

We say that a formula φ is true in a rooted Kripke structure M (written $M \models \varphi$) iff φ is true at the initial state of M. Remark that $M, s \models \blacksquare_a^{\subset} \varphi$ if and only if for any non-empty subset E of R_a we have that $M \setminus E, s \models \varphi$, R_a included. This implies that if $M, s \models \blacksquare_a^{\subset} \varphi$ then $M \setminus R_a, s \models \varphi$.

A strategy is a plan of action. As it is logical, the plan is winning when it leads me to victory, whatever my opponent's plan of action. Thus, a winning strategy can be expressed as an alternance of universally quantified sentences and existentially quantified sentences "for all actions of my Opponent, there is an action that I can make that leads me to victory". Let us put ourselves in the villain's shoes: suppose that we are the Attacker, and that, by playing, we have reached a certain state s of an Attack Graph AG. It is now Defender's turn. If I have a winning strategy, I must be able to reach a successor state s' of s and this for any subset that is removed by the Defender. Said differently, we must have that $AG, s \models \blacksquare^{\subset} \Diamond \varphi = (\blacksquare_a^{\subset} \Diamond\varphi) \wedge \cdots \wedge (\blacksquare_b^{\subset} \Diamond \varphi)$ for some formula φ that expresses the winning condition. First, we define:

$$\blacksquare^{\subset} \varphi \doteq \bigwedge_{a\in\Sigma} \blacksquare_a^{\subset} \varphi \qquad \Diamond \varphi \doteq \bigvee_{a\in\Sigma} \Diamond_a \varphi$$

We can now define a family of formulas expressing the fact the the Attacker as a winning strategy to win a Subset Sabotage Game played over an Attack Graph.

Definition 2 (Winning Formulas)**.** *The family $\{\psi_n\}_{n\in\mathbb{N}}$ of Winning formulas is defined by induction on n as follows:*

$$\psi_n = \begin{cases} \mathsf{win} & \text{if } n = 0 \\ \Diamond \top \wedge \blacksquare^{\subset} \Diamond(\psi_{n-1} \vee \mathsf{win}) & \text{otherwise} \end{cases} \tag{1}$$

The following thereom can be proved by induction on the size of the Attack Graph.

Theorem 1. *For any attack Graph $AG = \langle S, s_0, \Sigma, R, V, W\rangle$ such that $|S| = n$ we have that $M \models \mathsf{win} \vee \psi_{n-1}$ if and only if there is a Winning Attacker Strategy for the Subset Sabotage Game played over AG.*

The model-checking problem for SSML, consist in deciding if $M \models \varphi$ given a *finite* rooted Kripke Structure M and a SSML formula φ. The proof of the following theorem is obtained by defining a translation (parametrized by a given Kripke structure) from SSML formulas to SML formulas.

Theorem 2. *The model checking problem for SSML is decidable: if M is a finite rooted Kripke Structure and φ an SSML formula, we can decide whether $M \models \varphi$ or not.*

As a consequence of the two above theorems, we can decide if there is a winning Attacker Strategy for the Subset Sabotage Game played over AG, by checking if AG is a formal model of the formula ψ_{n-1} where n is the size of AG.

3.2. Attack Graph Games & Automata

In the SSML setting that we sketched in the previous section, we can decide if there is a winning Attacker Strategy over the subset sabotage game played over AG, by checking if AG is a formal model of a certain SSML formula ψ_n. Such a solution is quite ad-hoc: we have designed the logic SSML exactly to solve our problem. To reason about the kind of games that we described in the previous section, we can use another technique. We have presented such a technique in [16] by working together with other colleagues.

We define a game $\mathcal{G}$ as a structure $\langle V, v_I, T, G\rangle$ where $V = V_1 \cup V_2$ is a set of vertex such that $V_1 \cap V_2 = \emptyset$, $v_0 \in V_1$ is the initial vertex, $T = T_1 \cup T_2$ is a transition relation such that $T_1 \subseteq V_1 \times V_2$ and $T_2 \subseteq V_2 \times V_1$ and $G \subseteq S_2$ is a non-empty subset representing the set of Attacker's goal states. Given an Attack Graph $AG = \langle S, s_0, \Sigma, R, V, W\rangle$ together with a set of countermeasures $\mathcal{C}$ (as showed in Figure 1), we can easily convert the Attack Graph in a game $\mathcal{G}$ by using an algorithm. We briefly describe the idea of this algorithm with the following points:

1. Each vertex v of the game produced will be a pair $\langle s, L\rangle$, where s is an AG-state and L is a list of edge labels.

2. We start by setting as initial state of the game the pair $\langle s_0, \epsilon\rangle$ and for each s', such that $(s_0, a, s') \in R$ for some $a \in \Sigma$ we create an edge $(\langle s_0, \epsilon\rangle, \langle s', \epsilon\rangle) \in T_1$.

3. If one of the s' is in W we create a sink and we add $\langle s', \epsilon\rangle \in G$. Otherwise, for each countermeasure $c \in \mathcal{C}$ we create a vertex $\langle s', c\rangle \in V_2$ and we add a transition $(\langle s', \epsilon\rangle, \langle s', c\rangle) \in T_2$.

4. If in the Attack Graph AG there is an edge labeled with $d \neq c$ such that $(s', d, s'') \in R$ we add a vertex $\langle s'', d\rangle \in V_1$ and an edge $(\langle s', c\rangle, \langle s'', d\rangle) \in T_1$, otherwise we create a sink $(\langle s', c\rangle, \langle s', c\rangle) \in T_1$.

We repeat this procedure until there are states in AG. It is clear that if the Attack graph AG does not contain cycles, then the game $\mathcal{G}$ given by the procedure is a finite, labeled tree whose root is labeled by $v_I = \langle s_0, \epsilon \rangle$. We can thus define a winning strategy for the Attacker as a pruning of $\mathcal{G}$ in which each state in V_1 has at most one child and whose leaves are all in G, i.e., we can define winning strategies as follows.

Definition 3. *An attacker strategy for a game $\langle V, v_I, T, G \rangle$ is a finite tree whose root is v_I, whose branches are plays over $\mathcal{G}$ and that satisfy the following properties:*

1. *For each node s of the strategy: if $s \in V_1$ then s has at most one child.*
2. *For each node s of the strategy: if $s \in V_2$ and $s \notin G$ then s has as many child as there are nodes s' such that $(s, s') \in T_2$.*

An attacker strategy is winning whenever each leaf of the strategy belongs to G.

Finite trees can be recognized by *finite tree automaton.*

Definition 4. *A nondeterministic tree automaton (NTA, for short) is a tuple $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, F \rangle$, where: Q is a set of states, $q_0 \in Q$ is an initial state, $\delta : Q \times \Sigma \to 2^Q$ is a transition function mapping pairs of states and symbols to a set of states, and $F \subseteq Q$ is a set of the accepting states.*

A NTA $\mathcal{A}$ recognizes trees and works as follows. For a node tree labelled by a and $\mathcal{A}$ being in a state q, it sends different copies of itself to successors in accordance with δ. Given a non-deterministic tree automaton $\mathcal{A}$ (NTA, for short), we can check in linear time if the set of trees accepted by the automaton is empty. If such set is empty, there will be no winning strategies for the Attacker. Since the size (i.e. number of vertices) of a tree constructed by the NTA does not exceed the size of the game, we obtain the following result.

Theorem 3. *Given a game $\mathcal{G}$ it is possible to decide in linear time whether the Attacker has a strategy to win the game.*

3.3. Attack Graph Games & Interpreted System

The games we described in the two previous subsections are two-player turn-based games with perfect information. The two players move in turns and they have perfect information about each other's actions. These two assumptions are quite unrealistic. In particular, it is unrealistic to think that a Defender always have perfect information about Attacker's actions. Moreover, while in both models we have a clear view of what an Attacker is — we can identify him with the Attack Graph itself— the Defender is relegated to a subordinate role. Such a character does not have a clear ontological status has it is identified with a set of countermeasures that are given independently of the graph structure. If we want to think in terms of multi-agent system, we should try to give a clear agent's status to the Defender as well. To tackle all these problems we propose to think about Attacker and Defender(s) in a game played over an Attack Graph as *Agents* of an Interpreted System [17]. An Interpreted Systems (IS) is constructed starting from a finite set of Agents $Ag = \{1, \ldots, n\}$, each agent i is specified by giving:

- a set L_i of local states;

- a set act_i of agent's action;
- a function P_i associating to each local state a non-empty set of agent's i actions;
- a transition function t_i that takes an agent i state l_i and an action a_k for each agent in Ag as argument and outputs an agent i state. The function $t_i(l_i, a_1, \ldots, a_k)$ is defined if and only if we have that $a_i \in P_i(l_i)$.

If Ag is a set of k agents, a **global state** s is a tuple $s = \langle l_1, \ldots, l_k \rangle$ where each l_i is an agent i state. The set of global states is denoted by G_I. Finally an **Interpreted System** is a tuple $I = \langle Ag, s_0, T, \Pi \rangle$ where Ag is a set of agents, $s_0 \in G_I$ is the global initial state, $T : G_I \times act^{Ag} \to G_I$ is the global transition function such that $T(s, a_1, \ldots, a_k) = s'$ iff for every $i \in Ag$, $t_i(s, a_i) = s'$, and Π is a labeling function, associating to any global states a subset of atomic propositions. Interpreted systems can be used together with logics for the strategic reasoning such as Alternating-time temporal logic (ATL)[5] and Strategy Logic (SL)[7]. Moreover, interpreted systems came with a natural concept of imperfect information: if s is a global state, we denote by $s[i]$ its i-component. Two global states s and s', are **equivalent** for the agent i whenever $s[i] = s'[i]$. We denote such notion by $s \sim_i s'$. Two histories, i.e. finite sequences of global states, h and h', are equivalent for the agent i whenever they have the same length m and $h_j \sim_i h'_j$ for any $0 \leq j \leq m$.

Given an Attack Graph AG we can define an **Attacker Agent** whose states are the states of the attack graph, whose actions are the labels of the Attack Graph's edges (and eventually the idle action), and in which given two states s and s' of the AG if $(s, b, s') \in R$ then $t(s, (b, \vec{a})) = s'$ for some tuple of other's agent actions $\vec{a}$. The natural question is: how can we define the Defender? As we can see from the table in Figure 1, each attack in the Attack Graph, is identified with a set of preconditions and postconditions. The latter is nothing more than atomic propositions labeling states. If the Attacker is in a state s that is labeled with some preconditions, he can make an attack and move to a state s' labeled by the postconditions of the attack. We can suppose that the Defender's set $\mathcal{C}$ of countermeasures is defined symmetrically: each Defender countermeasure is identified with a pair of sets $\langle C_{pre}, C_{post} \rangle$ of atomic propositions that labels some states in the Attack Graph. Now, let $AG = \langle S, s_0, \Sigma, R, V, W \rangle$ be an attack graph. By definition, $V(s)$ is a set of atomic propositions, for any $s \in S$. In Section 2, we saw that the atomic propositions labeling a state represent two heterogeneous types of information: information about the *system* (vulnerabilities, state of the system, etc.), and information about the Attacker (his skills, knowledge of the network, access to device, etc.). We can imagine that a Defender has partial knowledge, e.g., she does not know exactly what are the skills of the Attacker, or she does not know to which devices the Attacker can access, or she has only information about the system or, even, of some subset of states of the system. We can thus imagine that the pairs $\langle C_{pre}, C_{post} \rangle$ in the set of countermeasures vary over the knowledge of the defender, i.e. $\mathcal{P}_D \subseteq \mathcal{P}$. Consider the directed graph $AG_D = \langle S_D, R_D \rangle$ where the set of states S_D is the set of equivalence classes of states of the Attack Graph modulo $\mathcal{P}_D$ (the knowledge of the defender). That is, $s \in S_D$ iff s is an equivalence class of states modulo the relation $s_i \equiv s_j \iff V(s_i) \cap \mathcal{P}_D = V(s_j) \cap \mathcal{P}_D$, where $s_i, s_j \in S$. The set of edges R_D is given by:

1. $(s, s') \in R_D$ if $(s_i, b, s_j) \in R$ for some states of the AG, s_i belonging to the equivalence class s and s_j belonging to the equivalence class s' or,

2. $(s, s') \in R_D$ if $V(s_i) = C_{pre}$ and $V(s_j) = C_{post}$ for some state of the AG, s_i belonging to the equivalence class s and s_j belonging to the equivalence class s'.

Over such a structure, we can construct a **Defender Agent**, whose set of states is S_D, having an action for each edge in R_D and in which the local transition functions mimics the relation R_D. We can impose that whenever the defender is in a state that is a precondition of a countermeasure, if she plays a certain action then the local transition functions will output the associated postconditions *no matter what are the actions of the other agents*.

Remark that by using such a construction, we can easily consider n-different defenders: we can imagine that each defender knows a different subset of $\mathcal{P}$ and as a consequence generates different defender's agent. This approach paves the way to the introduction of the strategic reasoning typical of the ATL logics into the Attack Graphs frame. In the two previous subsections, we only asked whether, given an AG one can find a winning Attacker strategy, i.e., we were obliged to consider only reachability goals for the Attacker. Thanks to the fact that Interpreted Systems are a model of the logic ATL, we can specify the strategic properties that we are interested to check via such a logic, and e.g., verify properties about the strategy ability of a coalition of Defenders.

4. Conclusion

We have sketched three possible formalization of Attacker-Defender(s) games that are played over an Attack Graph. All the approaches make use of formal methods techniques to estimate whether one of the players can win the proposed games. Two of the formalization consider two-players turn-based reachability games with perfect information, and are obtained by the means of, respectively, modal logic and automata theory. The third conceptualization, which in our opinion is the more promising, paves the way to a formalization of such games in terms of concurrent n-agents games with imperfect information and the use of formal verification techniques, e.g., model checking for the various strategic logics, to reason about such games.

The amount of work that still needs to be done to make such formalization attractive is enormous. However, we take the liberty of pointing out a research direction that we feel is a priority. It is known that under the hypothesis of uniformity, the model checking problem for ATL* with memoryfull strategies and imperfect information is undecidable [18]. Thus, if we want to use effectively the Interpreted System framework we sketched, we should restrict ourselves to use partial solutions to the problem, such as abstractions either on the information [19, 20, 21, 22] or on the strategies [23, 24, 25, 26, 27]. In the last decade, ATL has been extended to deal with the concept of strategies to obtain an objective under a limited bound of resources [28]. Such a concept is attractive for the security scenario that we are trying to take into account: an Attacker could have a limited amount of resources to make an attack and, equally, a Defender could have a limited amount of resources to make a countermeasure. To the best of our knowledge, quantitative extensions of ATL with imperfect information have not been studied in the literature. Furthermore, in this context, we can also study if an Attacker has some backup strategies to

achieve his objectives by following the line on graded modalities as done in [29, 30, 31]. We believe that it would be interesting and useful for our goals to study such variants.

References

[1] E. M. Clarke, O. Grumberg, D. A. Peled, Model Checking, The MIT Press, Cambridge, Massachusetts, 1999.

[2] E. Clarke, E. Emerson, Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic., in: LP 1981 ,1981, pp. 52–71.

[3] O. Kupferman, M. Vardi, P. Wolper, An Automata Theoretic Approach to Branching-Time ModelChecking., Journal of the ACM 47 (2000) 312–360.

[4] O. Kupferman, M. Vardi, P. Wolper, Module Checking., Information and Computation 164 (2001) 322–344.

[5] R. Alur, T. Henzinger, O. Kupferman, Alternating-Time Temporal Logic., Journal of the ACM 49 (2002) 672–713.

[6] A. Lomuscio, H. Qu, F. Raimondi, MCMAS: A model checker for the verification of multi-agent systems, in: (CAV09), 2009, pp. 682–688.

[7] F. Mogavero, A. Murano, G. Perelli, M. Y. Vardi, Reasoning about strategies: On the model-checking problem, ACM Trans. Comput. Log. 15 (2014) 34:1–34:47. URL: https://doi.org/10.1145/2631917. doi:10.1145/2631917.

[8] W. Jamroga, A. Murano, Module checking of strategic ability, in: AAMAS 2015, 2015, pp. 227–235.

[9] N. R. Jennings, M. Wooldridge, Application of intelligent agents, in: Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

[10] R. P. Lippmann, K. W. Ingols, An annotated review of past papers on attack graphs (2005).

[11] C. Phillips, L. P. Swiler, A graph-based system for network-vulnerability analysis, in: NSPW98, 1998, pp. 71–79.

[12] K. Kaynar, A taxonomy for attack graph generation and usage in network security, J. Inf. Secur. Appl. 29 (2016) 27–56.

[13] T. Heberlein, M. Bishop, E. Ceesay, M. Danforth, C. Senthilkumar, T. Stallard, A taxonomy for comparing attack-graph approaches, [Online] http://netsq.com/Documents/AttackGraphPaper. pdf (2012).

[14] J. van Benthem, An Essay on Sabotage and Obstruction, Springer Berlin Heidelberg, 2005, pp. 268–276. URL: https://doi.org/10.1007/978-3-540-32254-2_16. doi:10.1007/978-3-540-32254-2_16.

[15] D. Catta, J. Leneutre, V. Malvone, Subset sabotage games & attack graphs, in: WOA 2022, 2022, pp. 209–218. URL: http://ceur-ws.org/Vol-3261/paper16.pdf.

[16] D. Catta, A. Di Stasio, A. Murano, J. Leneutre, V. Malvone, A Game Theoretic Approach to Attack Graphs, 2023. To appear in ICAART 2023.

[17] R. Fagin, J. Halpern, Y. Moses, M. Vardi, Reasoning about Knowledge., MIT, 1995.

[18] C. Dima, F. Tiplea, Model-checking ATL under imperfect information and perfect recall semantics is undecidable, CoRR abs/1102.4225 (2011). URL: http://arxiv.org/abs/1102.4225.

[19] F. Belardinelli, A. Lomuscio, V. Malvone, An abstraction-based method for verifying

strategic properties in multi-agent systems with imperfect information, in: IAAI 2019, 2019, pp. 6030–6037. URL: https://doi.org/10.1609/aaai.v33i01.33016030. doi:10.1609/aaai.v33i01.33016030.

[20] F. Belardinelli, V. Malvone, A three-valued approach to strategic abilities under imperfect information, in: KR 2020, 2020, pp. 89–98. URL: https://doi.org/10.24963/kr.2020/10. doi:10.24963/kr.2020/10.

[21] A. Ferrando, V. Malvone, Towards the combination of model checking and runtime verification on multi-agent systems, in: PAAMS 2022, 2022, pp. 140–152. URL: https://doi.org/10.1007/978-3-031-18192-4_12. doi:10.1007/978-3-031-18192-4_12.

[22] A. Ferrando, V. Malvone, Towards the verification of strategic properties in multi-agent systems with imperfect information, AAMAS (2023) (to appear).

[23] W. Jamroga, V. Malvone, A. Murano, Natural strategic ability, Artif. Intell. 277 (2019). URL: https://doi.org/10.1016/j.artint.2019.103170. doi:10.1016/j.artint.2019.103170.

[24] W. Jamroga, V. Malvone, A. Murano, Natural strategic ability under imperfect information, in: AAMAS 2019 , 2019, pp. 962–970. URL: http://dl.acm.org/citation.cfm?id=3331791.

[25] F. Belardinelli, A. Lomuscio, A. Murano, S. Rubin, Verification of multi-agent systems with public actions against strategy logic, Artif. Intell. 285 (2020) 103302. URL: https://doi.org/10.1016/j.artint.2020.103302. doi:10.1016/j.artint.2020.103302.

[26] R. Berthon, B. Maubert, A. Murano, S. Rubin, M. Y. Vardi, Strategy logic with imperfect information, ACM Trans. Comput. Log. 22 (2021) 5:1–5:51. URL: https://doi.org/10.1145/3427955. doi:10.1145/3427955.

[27] F. Belardinelli, A. Lomuscio, V. Malvone, E. Yu, Approximating perfect recall when model checking strategic abilities: Theory and applications, J. Artif. Intell. Res. 73 (2022) 897–932. URL: https://doi.org/10.1613/jair.1.12539. doi:10.1613/jair.1.12539.

[28] N. Alechina, B. Logan, H. N. Nguyen, F. Raimondi, L. Mostarda, Symbolic model-checking for resource-bounded ATL, in: AAMAS 2015, 2015, pp. 1809–1810.

[29] M. Faella, M. Napoli, M. Parente, Graded alternating-time temporal logic, Fundam. Informaticae 105 (2010) 189–210. URL: https://doi.org/10.3233/FI-2010-363. doi:10.3233/FI-2010-363.

[30] B. Aminof, V. Malvone, A. Murano, S. Rubin, Graded modalities in strategy logic, Inf. Comput. 261 (2018) 634–649. URL: https://doi.org/10.1016/j.ic.2018.02.022. doi:10.1016/j.ic.2018.02.022.

[31] V. Malvone, F. Mogavero, A. Murano, L. Sorrentino, Reasoning about graded strategy quantifiers, Information and Computation 259 (2018) 390 – 411.