Department of
Information Engineering
and Computer Science
DISI

UNIVERSITY
OF TRENTO - Italy

DISI - Via Sommarive 14 - 38123 Povo - Trento (Italy)
http://www.disi.unitn.it

AUTOMATA MODULO THEORY (AMT)

Fabio Massacci and Ida Siahaan

May 2009

Technical Report # DISI-09-027

Contents

Abstract

With the advent of the next generation java servlet on the smartcard, the Future Internet will be composed by web servers and clients silently yet busily running on high end smart cards in our phones and our wallets. In this new world model we can no longer accept the current security model where programs can be downloaded on our machines just because they are vaguely "trusted". We want to know what they do in more precise details.

We claim that the Future Internet needs the notion of *security-by-contract*: a contract describes the security relevant interactions that an application could have with the smart devices hosting them. Compliance with contracts should verified at development time, checked at deployment time and contracts should be accepted by the platform before deployment and possibly their enforcement guaranteed, for instance by in-line monitoring.

In this technical report we provide a formal model and an algorithm for matching the claims on the security behavior of a midlet (for short *contract*) with the desired security behavior of a platform (for short *policy*) on a security-by-contract framework for realistic security scenarios.

Keywords Access control · Language-based security · Malicious code · Security and privacy policies

1 Introduction

In this technical report we provide a formal model and an algorithm for matching the claims on the security behavior of a midlet (for short *contract*) with the desired security behavior of a platform (for short *policy*) for realistic security scenarios (such as the "only https connections" mentioned afore).

The formal model used for capturing contracts and policies is based on the novel concept of *Automata Modulo Theory* ($\mathcal{AMT}$). $\mathcal{AMT}$ generalizes the finite state automata of model-carrying code [43] and extends Büchi Automata (BA). It is suitable for formalizing systems with finitely many states but infinitely many transitions, by leveraging the power of satisfiability-modulo-theory (SMT for short) decision procedures. $\mathcal{AMT}$ enables us to define very expressive and customizable policies as a model for *security-by-contract*, by capturing the infinite transition into finite transitions labeled as expressions in suitable theories.

The second contribution is a decision procedure (and its complexity characterization) for matching the mobile's policy and the midlet's security claims that concretize the meta-level algorithm of security-by-contract [5]. We map the problem into classical automata theoretic construction such as product and emptiness test.

Since our goal is to provide this midlet-contract vs platform-policy matching on-the-fly (during the actual download of the midlet) issues like small memory footprint and effective computations play a key role. We show that the tractability limit is the complexity of the satisfiability procedure for the underlying theories used to describe labels: we use NLOGSPACE and linear time algorithms for the automata theoretic part [26] with oracle queries to a decision procedure solver[1]. Out of a number of requirements studies, most of the policies of interests can be captured by theories which only requires PTIME decision procedures.

We have further customized the decision algorithm the security policy has a particular form. For instance, if one uses security automata á la Schneider those can be mapped to a particular form of $\mathcal{AMT}$ (with all accepting states and an error absorbing state) for which particular optimizations are possible. In the original paper by Schneider security automata specify transitions as a function of the input symbols which can be the entire system state. Our $\mathcal{AMT}$ differs from security automata in this respects: transitions are environmental parameters rather than system states. Writing policies in this way is closer to one's intuition.

This matching on-the-fly however requires to complement the policy of the mobile platform and if we assume a general non-deterministic automaton this complementation might lead to an exponential blow-up. A second problem is that in this way we need two representations of the policy: a direct representation of the policy as an automata that we can use for run-time monitor [47] and the complemented representation that we use for matching.

Thus, we further propose to use the notion of simulation for matching the security policy of the platform against the security claims of the midlet. Simulation is stronger

[1] In a nutshell $\mathcal{AMT}$ makes reasoning about infinite state systems possible without symbolic manipulation procedures of zones and regions or finite representation by equivalence classes [24] that would not be suitable for our intended application i.e. checking security claims before a pervasive download on a mobile phone.

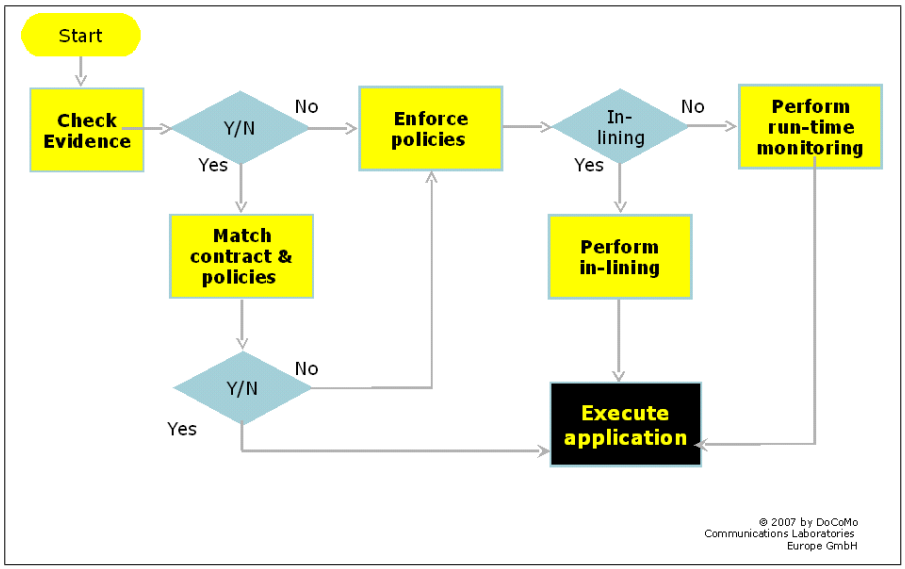


Figure 1: Workflow in Security-by-Contract

than language inclusion (i.e. less midlets will obtain a green light) but they coincide for deterministic policies.

In the next section, we briefly recap the notion of Security-by-Contract (we refer the reader to [5] for more details). Next we introduce $\mathcal{AMT}$ and the corresponding automata operations in (§3). We also expose some specific issues to be considered in $\mathcal{AMT}$. In §4 we describe an approach for lifting finite state tools to $\mathcal{AMT}$. Next, we describe simulation, symbolic simulation and fair simulation for $\mathcal{AMT}$ (§5) and we continue with algorithm for lifting finite state tools to $\mathcal{AMT}$ simulation (§6). Finally, we present related works and a concluding discussion.

2 Security by Contract in a nutshell

Security-by-contract (S×C)[11, 5] proposed to augment mobile code with a claim on its security behavior that can be matched against a mobile platform policy on-the-fly, which provides semantics for digital signatures on mobile code. In an S×Cframework [11, 5] a mobile code is augmented with a claim on its security behavior (an *application's contract*) that could be matched against a mobile *platform's policy* before downloading.

At *development time* the mobile code developers are responsible for providing a description of the security behavior that their code finally provides. Such a code may undergo a formal certification process by the developer's own company, the smart card provider, a mobile phone operator, or any other third party for which the application has been developed. By using suitable techniques such as static analysis, monitor in-lining, or general theorem proving, the code is certified to comply with the developer's contract. Next, the code and the security claims are sealed together with the evidence for compliance (either a digital signature or a proof) and shipped as shown on Figure 2.

At *deployment time*, the target platform follows a workflow as depicted in Figure 1

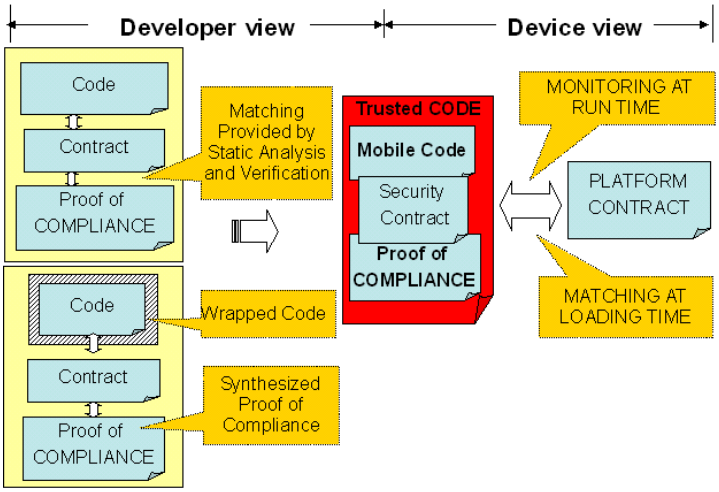


Figure 2: Mobile Code Components with Security-by-Contract

[5]. This workflow is a modification of S×Cworkflow [5]) by adding optimization step. First, the correctness of the evidence of a code is checked. Such evidence can be a trusted signature [51] or a proof that the code satisfies the contract (one can use Proof-Carrying-Code (PCC) techniques to check it [34]). When there is evidence that a contract is trustworthy, a platform checks, that the claimed contract is compliant with the policy to enforce. If it is, then the application can be run without further ado. It is a significant saving from in-lining a security monitor. In case that at *run-time* we decide to still monitor the application, then we add a number of checks into the application so that any undesired behavior can be immediately stopped or corrected.

Matching succeeds, if and only if, by executing an application on the platform, every behavior of the application that satisfies its contract also satisfies the platform's policy. If matching fails, but we still want to run the application, then we use either a security monitor in-lining, or run-time enforcement of the policy (by running the application in parallel with a reference monitor that intercepts all security relevant actions). However with a constrained device, where CPU cycles means also battery consumption, we need to minimize the run-time overheads as much as possible.

A *contract* is a formal specification of the behavior of an application for relevant security actions for example Virtual Machine API Calls, Web Messages. By signing the code the developer certifies that the code complies with the stated claims on its security-relevant behavior. A *policy* is a formal specification of the acceptable behavior of applications to be executed on a platform for what concerns relevant security actions. Thus, a digital signature does not just certify the origin of the code but also bind together the code with a contract with the main goal to provide a semantics for digital signatures on mobile code. Therefore, this framework is a step in the transition from trusted code to trustworthy code.

Technically, a contract is a security automaton in the sense of Schneider [20], and it specifies an upper bound on the security-relevant behavior of the application: the sequences of security-relevant events that an application can generate are all in the

language accepted by the security automaton.

A *policy*(also *contract*) covers a number of issues such as file access, network connectivity, access to critical resources, or secure storage. A single contract can be seen as a list of disjoint claims (for instance rules for connections). An example of a rule for sessions regarding A Personal Information Management (PIM) and connections is shown in Example 2.1, which can be one of the rules of a contract. Another example is a rule for method invocation of a Java object as shown in Example 2.2. This example can be one of the rules of a policy. Both examples describe safety properties, which are common properties to be verified.

Example 2.1 *PIM system on a phone has the ability to manage appointment books, contact directories, etc., in electronic form. A privacy conscious user may restrict network connectivity by stating a policy rule: "After PIM is opened no connections are allowed". This contract permits executing the* `javax.microedition.io.Connector.open()` *method only if the* `javax.microedition.pim.PIM.openPIMList()` *method was never called before.*

Example 2.2 *The policy of an operator may only require that "After PIM was accessed only secure connections can be opened". This policy permits executing the* `javax.microedition.io.Connector.open(string url)` *method only if the started connection is a secure one i.e.* `url` *starts with "https://".*

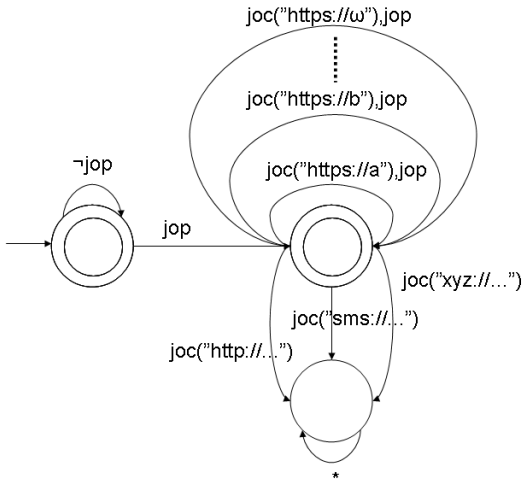
We can have a slightly more sophisticated approach using Büchi automata [44] if we also want to cover liveness properties as shown in the following Example 2.3.

Example 2.3 *If the application should use all the permissions it requests then for each permission p at least one reachable invocation of a method permitted by p must exist in the code. For example if p is* `io.Connector.http` *then a call to method* `Connector.open()` *must exist in the code and the url argument must start with "http". If p is* `io.Connector.https` *then a call to method* `Connector.open()` *must exist in the code and the url argument must start with "https" and so on for other constraints e.g. permission for sending SMS.*

3 Automata Modulo Theory

The security behaviors, provided by the contract and desired by the policy, can be represented as automata, where transitions corresponds to invocation of APIs as suggested by Erlingsson [12, p.59] and Sekar et al. [43]. Thus, the operation of matching the midlet's claim with platform policy can be mapped into classical problems in automata theory.

One possible mechanism to represent matching is *language inclusion*: given two automata Aut^C and Aut^P representing respectively the formal specification of a contract and of a policy, we have a match when the execution traces of the midlet described by Aut^C are a subset of the acceptable traces for Aut^P. To check this property we can complement the automaton of the policy, thus obtaining the set of traces disallowed by the policy and check its intersection with the traces of the contract. If the intersection is not empty, any behavior in it corresponds to a security violation.



(a) An Infinite Automaton of Ex. 2.2

$$
\begin{aligned}
joc(url) &\doteq \texttt{javax.microedition.io.}\\
&\quad\ \texttt{Connector.open}(url)\\
jop &\doteq \texttt{javax.microedition.pim.}\\
&\quad\ \texttt{PIM.openPIMList}(\ldots)\\
p(url) = type &\doteq url.\texttt{startsWith}(type)
\end{aligned}
$$

(b) Abbreviations for Java APIs

Figure 3: Infinite Transitions Security Policies

The other alternative is the notion of *simulation*: we have a match when every APIs invoked by Aut^C can also be invoked by Aut^P. In other words, every behavior of Aut^C is also a behavior of Aut^P. Simulation is a stronger notion than language inclusion as it requires that the policy allows the actions of the midlet's contract in a "step-by-step" fashion, whereas language inclusion looks at an execution trace as a whole. We pursue the language inclusion approach in [32] and in this technical report and refer to [33] for the simulation approach.

While this idea of representing the security-digest as an automaton is almost a decade old [43, 12], the practical realization has been hindered by a significant technical hurdle: we cannot use the naive encoding into automata for practical policies. Even the basic policies in Ex. 2.1 and Ex. 2.2 lead to automata with infinitely many transitions.

Fig.3a represents an automaton for Ex. 2.2. We start from state p_0 and stay in this state while PIM is not accessed (jop). As PIM is accessed, we move to state p_1 and stay in state p_1 only if the started connection javax.microedition.io.Connector.open(string url) method is a secure one (url starts with "https://") or we keep accessing PIM (jop). If we start an insecure connection javax.microedition.io.Connector.open(string url), for example url starts with "http://" or "sms://", then we enter state e_p.

The examples presented are from a Java VM; since we do not consider it useful to invent our own names for API calls, we use the `javax.microedition` APIs (even though verbose) for the notation shown in Fig.3b.

3.1 Theory in Automaton Modulo Theory

The notion of *theory* in $\mathcal{AMT}$ is derived from the notion of *theory* in the Satisfiability Modulo Theories (SMT) problem. The SMT problem focuses on the satisfiability of quantifier-free first-order formulas modulo background theories [7]. Some theories of

Table 1: Theories of Interest

Theory	(Non)Convex	Decidability	Complexity
$\mathcal{EUF}$	convex	decidable	polynomial [1]
$\mathcal{LA}(\mathbb{Q})$	convex	decidable	polynomial [19]
$\mathcal{LA}(\mathbb{Z})$	non-convex	decidable	NP-Complete [38]
$\mathcal{DL}(\mathbb{Q})$	convex	decidable	polynomial [9]
$\mathcal{DL}(\mathbb{Z})$	non-convex	decidable	NP-Complete [31]

interest for example are the theory of equality and uninterpreted functions ($\mathcal{EUF}$), the quantifier-free fragment of linear arithmetic over the rationals ($\mathcal{LA}(\mathbb{Q})$), and over the integers ($\mathcal{LA}(\mathbb{Z})$), and the corresponding subtheories of difference logic both over the rationals ($\mathcal{DL}(\mathbb{Q})$), and over the integers ($\mathcal{DL}(\mathbb{Z})$).

Example 3.1 *A security policy may set limits on resources that can be captured with constraints expressed in different theories*

1. *no communication allowed if the battery level falls below 30% ($\mathcal{LA}(\mathbb{Q})$ can be used);*
2. *no jpeg file can be downloaded with size more than 500KB while avi files can arrive up to 1MB ($\mathcal{LA}(\mathbb{Z})$ can be used here)*
3. *$\mathcal{EUF}$ can be used when comparing a policy requiring* `protocol(url)=''https''` *and* `port(url)=''8080''` *with a contract claiming to use only connections where* `protocol(url)=''https''` *or* `protocol(url)=''http''`*. We do not need to extract a protocol from the url. It is sufficient to treat protocol and port as uninterpreted functions and apply the theory of equality and uninterpreted functions $\mathcal{EUF}$.*

The previous examples show simple security policies each uses only one kind of theory. However, we are particularly interested in the combination of two or more theories to accommodate complex security policies.

Example 3.2 *A policy may allow only secure connections with limited size of downloads. To express this policy we combine $\mathcal{EUF}$ for handling* `protocol(url)=''https''` *and $\mathcal{LA}(\mathbb{Z})$ for handling downloading a file of at most 500KB.*

We use traditional first-order logic terminology [15] for defining a SMT theory. A signature Σ consists a set of function symbols $\mathcal{F}$ and a set of predicate symbols $\mathcal{P}$ with their arities, and a set of variables V. A 0-ary function symbol c is called a *constant* and 0-ary predicate symbol B is called a *Boolean atom*. A Σ-term is a variable in V or constructed from application of function symbols $\mathcal{F}$ to Σ-terms. If $t_1, \ldots, t_n$ are Σ-terms and p is a predicate symbol then $p(t_1, \ldots, t_n)$ is a Σ-atom. A Σ-literal is a Σ-atom or negation of Σ-atom. Σ-formula is defined over Σ-literals, the universal and the existential quantifiers $\forall, \exists$, and the boolean connectives $\neg, \wedge$. A Σ-formula is named quantifier-free when it contains no quantifier and sentence when it contains no free variables. A Σ-theory $\mathcal{T}$ is a set of first-order sentences with signature Σ.

A Σ-structure $\mathcal{M}$ is a model of Σ-theory $\mathcal{T}$ if $\mathcal{M}$ satisfies every sentences in $\mathcal{T}$. A Σ-structure $\mathcal{M}$ consists of a set D of elements as *domain* and an interpretation $\mathcal{I}$ as in first order logic. The interpretation of an n-ary function symbol is a mapping of each n-ary function symbol $f \in \Sigma$ to a total function $f^{\mathcal{M}} : D^n \to D$. The interpretation of a constant symbol is a mapping of each constant $c \in \Sigma$ to itself. The interpretation of an n-ary predicate symbol is a mapping of each n-ary predicate symbol $p \in \Sigma$ to a relation $p^{\mathcal{M}} \subseteq D^n$ and the interpretation of a Boolean atom is a mapping of each Boolean atom $B \in \Sigma$ to $(\top, \bot)$. Let $\mathcal{M}$ denote a Σ-structure, ϕ a formula, and $\mathcal{T}$ a theory, all of signature Σ. We say that ϕ is satisfiable in $\mathcal{M}$ (or ϕ is $\mathcal{T}$-satisfiable) if there exists some assignment α which assigns the set of variables to values in the domain such that $(\mathcal{M}, \alpha) \models \phi$. A theory $\mathcal{T}$ is convex [42] if all the conjunctions of literals are convex in theory $\mathcal{T}$. A conjunction of $\mathcal{T}$-literals in a theory $\mathcal{T}$ is convex if for each disjunction $(\mathcal{M}, \alpha) \models \bigvee_{i=1}^{n} e_i$ if and only if $(\mathcal{M}, \alpha) \models e_i$ for some i, where e_i are equalities between variables occurring in $(\mathcal{M}, \alpha)$. For practical purposes we make some additional restrictions.

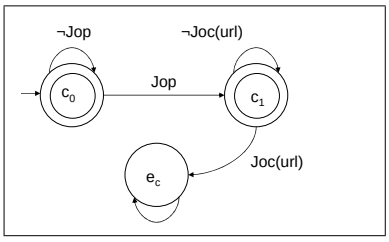
First-order as base logic We use classical first-order logic based SMT. Extension to a higher-order logic is possible as proposed in [30], where they introduced *parametric theories*. In the sequel, we consider only quantifier-free Σ-formulas on theories $\mathcal{T}$ where the $\mathcal{T}$-satisfiability of conjunctions of literals is decidable by a $\mathcal{T}$-solver [37].

Combination of theories is consistent Given a consistent theory $\mathcal{T}_1$ and a consistent theory $\mathcal{T}_2$, we assume that the combination theory $\mathcal{T} := \mathcal{T}_1 \cup \mathcal{T}_2$ is also consistent and there exists a $\mathcal{T}$-solver for the combined theory. We are interested in $\mathcal{T}_1 \cup \mathcal{T}_2$-satisfiability of $\Sigma_1 \cup \Sigma_2$-formulas that can be generalized to combine many possibly signature-disjoint theories $\mathcal{T}_1 \cup \ldots \cup \mathcal{T}_n$. The Nelson-Oppen (NO) combination procedure [35] is a seminal work in this area. NO combines decision procedures for first-order theories restricted to theories that are *stably-infinite* (informally the theory that has infinite models (see [35])) and that *have disjoint signatures* ($\Sigma_1 \cap \Sigma_2 = \emptyset$). Tinelli-Zarba's combination procedure [46] extends NO for combining an arbitrary theory which maybe stably infinite with a stably infinite theory that is also *shiny*. They also proposed a variant of the combination method for combining theories having only finite models with theories that are stably finite. Ghilardi's combination procedure [17] extends NO for combining theories that share signature with restriction that the theories are compatible with respect to a common sub theory in the shared signature.

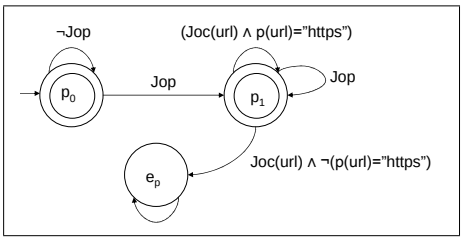
Conjunctions of formulas Given theories $\mathcal{T}_1$ and $\mathcal{T}_2$ that can be combined as $\mathcal{T}$ where $\mathcal{T} := \mathcal{T}_1 \cup \mathcal{T}_2$ and conjunctive normal form formula ϕ_1(resp. ϕ_2) that is satisfiable in $\mathcal{T}_1$ (resp. $\mathcal{T}_2$) then $\phi_1 \wedge \phi_2$ is decidable in $\mathcal{T}$ (not necessarily satisfiable). We do not impose restrictions as in Proposition 3.8. in [45] that if ϕ_1(resp. ϕ_2) is satisfiable, then $\phi_1 \wedge \phi_2$ is satisfiable in $\mathcal{T}$.

3.2 Automaton Modulo Theory Preliminary

Having defined the *theory* in $\mathcal{AMT}$, in this section we continue by defining tuple, run, and word in $\mathcal{AMT}$. An automaton in $\mathcal{AMT}$ is defined as a tuple of a finite set of



(a) $\mathcal{AMT}$ rule from Example 2.1



(b) $\mathcal{AMT}$ rule from Example 2.2

$$
\begin{aligned}
Joc(url) &\doteq \texttt{Joc(joc,url)} \\
Jop &\doteq \texttt{Jop(jop,}x_1,\ldots,x_n\texttt{)} \\
p(url) = type &\doteq url.\texttt{startsWith(}type\texttt{)} \\
joc &\doteq \texttt{javax.microedition.io.Connector.open} \\
jop &\doteq \texttt{javax.microedition.pim.PIM.openPIMList}
\end{aligned}
$$

`Joc,Jop` are predicate symbols representing respectively `joc(url),jop(`$x_1,\ldots,x_n$`)` APIs.

(c) Abbreviations for expressions

Figure 4: $\mathcal{AMT}$ Examples

Σ-formulas in Σ-theory $\mathcal{T}$, a finite set of states, an initial state, a labeled transition relation, and a set of accepting states. Formally, it is given in Definition 3.1.

Definition 3.1 (Automaton Modulo Theory ($\mathcal{AMT}$)) *An $\mathcal{AMT}$ is a tuple $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$, where E is a finite set of Σ-formulas in Σ-theory $\mathcal{T}$, S is a finite set of states, $\mathbf{s_0} \in S$ is the initial state, $\Delta \subseteq S \times E \times S$ is a labeled transition relation, and $F \subseteq S$ is a set of accepting states.*

Figure 4 shows two examples of $\mathcal{AMT}$ using the signature for $\mathcal{EUF}$ with a function symbol $p()$ representing the protocol type used for the opening of a *url*. As described in the cited examples the first automaton forbids the opening of plain http-connections as soon as the PIM is invoked while the second just restricts connections to be only https.

The transitions in these automata describe with an expression a potentially infinite set of transitions: the opening of all possible *urls* starting with https. The automaton modulo theory is therefore an abstraction for a concrete (but infinite) automaton. The *concrete automaton* corresponds to the behavior of the actual system in terms of API calls, value of resources and the likes.

From a formal perspective, the concrete model of an automaton modulo theory intuitively corresponds to the automaton where each symbolic transition labeled with an expression is replaced by the set of transitions corresponding to all satisfiable instantiations of the expression. To characterize how an automaton captures the behavior of programs we need to define the notion of a trace. So, we start with the notion of a symbolic run which corresponds to the traditional notion of run in automata.

Definition 3.2 ($\mathcal{AMT}$ symbolic run) *Let $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ be an $\mathcal{AMT}$. A* symbolic run *of A is a sequence of states alternating with expressions $\sigma = \langle s_0 e_1 s_1 e_2 s_2 \ldots\rangle$, such that:*

1. $s_0 = \mathbf{s_0}$
2. $(s_i, e_{i+1}, s_{i+1}) \in \Delta$ *and* e_{i+1} *is $\mathcal{T}$-satisfiable, that is there is some Σ-structure $\mathcal{M}$ a model of Σ-theory $\mathcal{T}$ and there exists some assignment α such that $(\mathcal{M}, \alpha) \models e_{i+1}$.*

A finite symbolic run *is denoted by $\langle s_0 e_1 s_1 e_2 s_2 \ldots s_{n-1} e_n s_n\rangle$. An* infinite symbolic run *is denoted by $\langle s_0 e_1 s_1 e_2 s_2 \ldots\rangle$. A finite run is* accepting *if the last state goes through some accepting state, that is $s_n \in F$. An infinite run is* accepting *if the automaton goes through some accepting states infinitely often.*

In order to capture the actual system invocations we introduce another type of run called *concrete run* which is defined over valuations that represent actual system traces. A valuation ν consists of interpretations and assignments which are actual system traces.

Definition 3.3 ($\mathcal{AMT}$ concrete run) *Let $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ be an $\mathcal{AMT}$. A* concrete run *of A is a sequence of states alternating with a valuation $\sigma_C = \langle s_0 \nu_1 s_1 \nu_2 s_2 \ldots\rangle$, such that:*

1. $s_0 = \mathbf{s_0}$
2. *there exists expressions $e_{i+1} \in E$ such that $(s_i, e_{i+1}, s_{i+1}) \in \Delta$ and there is some Σ-structure $\mathcal{M}$ a model of Σ-theory $\mathcal{T}$ such that $(\mathcal{M}, \alpha_{i+1}) \models e_{i+1}$, where ν_{i+1} represents α_{i+1} and $\mathcal{I}(e_{i+1})$.*

A finite concrete run *is denoted by $\langle s_0 \nu_1 s_1 \nu_2 s_2 \ldots s_{n-1} \nu_n s_n\rangle$. An* infinite concrete run *is denoted by $\langle s_0 \nu_1 s_1 \nu_2 s_2 \ldots\rangle$. A finite run is* accepting *if the last state goes through some accepting state, that is $s_n \in F$. An infinite run is* accepting *if the automaton goes through some accepting states infinitely often. The trace associated with $\sigma_C = \langle s_0 \nu_1 s_1 \nu_2 s_2 \ldots\rangle$ is the sequence of valuations in the run. Thus a trace is* accepting *when the corresponding run is accepting.*

We use definition of run as in [14] which is slightly different from the one we use in [32], where we use only states.

Example 3.3 *An example of an* accepting symbolic run *of $\mathcal{AMT}$ rule from Example 2.2 shown in Figure 4b is*

c_0 $Jop(jop,file,permission)$ c_1 $Joc(joc,url) \wedge p(url)$="https" c_1 $Jop(jop,file,permission)$ c_1 $Joc(joc,url) \wedge p(url)$="https" ...

that corresponds with a non empty set of accepting concrete runs *for example*

$c_0(jop,PIM.CONTACT\_LIST,PIM.READ\_WRITE)$ c_1 $(joc,$"https://www.esse3.unitn.it/")

$c_1(jop,PIM.CONTACT\_LIST,PIM.READ\_ONLY)$ c_1 $(joc,$"https://online.unicreditbanca.it/login.htm") ...

Remark 3.1 *A symbolic run defined in Definition 3.2 is interpreted by a non empty set of concrete runs in Definition 3.3. This is a nature of our application domain where security policies define $\mathcal{AMT}$ in symbolic level and the system to be enforced has concrete runs. In other domains where we need the converse, namely to define symbolic runs from concrete runs, then a symbolic run defined in Definition 3.2 can be considered as an abstraction of concrete runs by Definition 3.3.*

In $\mathcal{AMT}$ a system uses variables that represent parameters over invoked methods. Hence, variables are only environment variables, and we can represent them as edge variables without memory. This observation leads to a subtle difference between traditional state variables in infinite systems and edge variables. For example, a guard $x < 3$ in classical hybrid automata for state variable x means that, after taking the transition, x must be smaller than 3. In our case, since x is some external parameter of a Java method, this means that this edge is taken each time the Java method is invoked with a value of x smaller than 3.

The *alphabet* of $\mathcal{AMT}$ is defined as a set of valuations $\mathcal{V}$ that satisfy E. A finite sequence of alphabet of A is called a *finite word* or *word* or *trace* denoted by $w = \langle \nu_1 \nu_2 \dots \nu_n \rangle$ and the length of w is denoted by $|w|$. An infinite sequence of alphabet of A is called an *infinite word* or *infinite trace* is denoted by $w = \langle \nu_1 \nu_2 \dots \rangle$. The set of infinite words recognized by an automaton A, denoted by $L_\omega(A)$, is the set of all accepting infinite traces in A. $L_\omega(A)$ is called the language accepted by A.

As we have noted already, the intuitive idea behind concrete runs is that they are sequences of models of the expressions of the abstract specification of the automaton modulo theory. In the practical setting, for example security policies over midlets, we want to capture sequences of API calls then this general theory can be actually narrowed.

Example 3.4 *A possible alternative is to use a predicate name corresponding to each api call (such as $joc(url, port)$, $jop()$, etc.) and then introduce a theory that specify that predicates are mutually exclusive.*

This formalization would correspond essentially to the guard-and-condition representation of Schneider's security automata.

Example 3.5 *Another alternative is to use predicate $API(APIsymbol, parameters)$ with the first argument the API name itself as a constant symbol to identify different methods. For example $joc(url, port)$ is denoted as $Joc(joc, url, port)$ and $jop(x_1, \dots, x_n)$ is denoted as $Jop(jop, x_1, \dots, x_n)$ imposing each constant as unique, i.e. $joc \neq jop$.*

Both formalizations capture the same concrete behavior in terms of API calls. Our current implementation uses the second option as the unique name assumption was built-in the SMT solver implementation and therefore it could be used more efficiently.

The transition relation of A may have many possible transitions for each state and expression, i.e. A is potentially non-deterministic.

Definition 3.4 (Deterministic $\mathcal{AMT}$) *$A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F \rangle$ is a* deterministic automaton *modulo theory $\mathcal{T}$, if and only if, for every $s \in S$ and every $s_1, s_2 \in S$ and every $e_1, e_2 \in E$, if $(s, e_1, s_1) \in \Delta$ and $(s, e_2, s_2) \in \Delta$, where $s_1 \neq s_2$ then the expression $(e_1 \wedge e_2)$ is unsatisfiable in the Σ-theory $\mathcal{T}$.*

3.3 Operations in Automaton Modulo Theory

In order to define the test for language inclusion we introduce the operation of complement and intersection of $\mathcal{AMT}$ operations at the concrete level, for example API calls, and then we give the notion of symbolic operations as in [22].

In this paper we consider only the *complementation of deterministic* $\mathcal{AMT}$, for all security policies in our application domain are naturally deterministic because a platform owner should have a clear idea on what to allow or disallow.

Complementation of $\mathcal{AMT}$ $\mathcal{AMT}$ automaton can be considered as a Büchi automaton where infinite transitions are represented as finite transitions. Therefore, for each deterministic $\mathcal{AMT}$ automaton A there exists a (possibly nondeterministic) $\mathcal{AMT}$ that accepts all the words which are not accepted by automaton A. The A^c can be constructed in a simple approach as in [48] as follows:

Definition 3.5 ($\mathcal{AMT}$ Complementation) *Given a deterministic* $\mathcal{AMT}$ $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ *the* complement $\mathcal{AMT}$ automaton $A^c = \langle E, \mathcal{T}, \Sigma, S^c, \mathbf{s_0}^c, \Delta^c, F^c\rangle$ *is:*

1. $S^c = S \times \{0\} \cup (S - F) \times \{1\}, \quad \mathbf{s_0}^c = (\mathbf{s_0}, 0), \quad F^c = (S - F) \times \{1\},$
2. *and for every* $s \in S$ *and* $e \in E$

$$
\begin{aligned}
((s,0),e,s') \in \Delta^c, s' &= \begin{cases} \{(t,0)\} & (s,e,t) \in \Delta \text{ and } t \in F \\ \{(t,0),(t,1)\} & (s,e,t) \in \Delta \text{ and } t \notin F \end{cases} \\
((s,1),e,s') \in \Delta^c, s' &= \{(t,1)\} \text{ if } (s,e,t) \in \Delta \text{ and } t \notin F
\end{aligned}
$$

To apply complementation in Definition 3.5, the deterministic automata has to be completed, meaning the sum of the transitions labels covers all the set of formulas in E. Return to our Example 2.1 shown in Figure 4a, the automaton is complete.

Proposition 3.1 *Let A be an $\mathcal{AMT}$ over a set of valuations $\mathcal{V}$. Then a (possibly nondeterministic) $\mathcal{AMT}$ A^c constructed by Definition 3.5 accepts all the concrete runs which are not accepted by A, that is A^c is a complement automaton such that $L_\omega(A^c) = \mathcal{V}^\omega - L_\omega(A)$.*

Proof.

Correctness.

"$\supseteq$" we take an arbitrary concrete run not accepted by A that corresponds to a word $w = \langle \nu_1\nu_2\nu_3 \ldots\rangle$, meaning $w \in \mathcal{V}^\omega - L_\omega(A)$, so there is a unique concrete run $\sigma_C = \langle s_0\nu_1 s_1\nu_2 s_2 \ldots\rangle$ of A. Hence, there is some k such that $\forall i > k, s_i \notin F$, meaning that $\sigma_C^c = \langle (s_0,0)\nu_1 \ldots (s_k,0)\nu_{k+1}(s_{k+1},1)\ldots\rangle$ is an accepting concrete run of A^c.

"$\subseteq$" we take an arbitrary concrete run accepted by A^c that corresponds to a word $w = \langle \nu_1\nu_2\nu_3 \ldots\rangle$, meaning that $w \in L_\omega(A^c)$, so there is a unique concrete run $\sigma_C^c = \langle (s_0,0)\nu_1 \ldots (s_k,0)\nu_{k+1}(s_{k+1},1)\ldots\rangle$ of A^c, corresponds to a concrete run $\sigma_C = \langle s_0\nu_1 \ldots s_k\nu_{k+1}s_{k+1}\ldots\rangle$of A on w but this concrete run is rejecting.

Termination This construction terminates because our states in S and formulas in E are finite.

Complexity The time and space complexity of the construction is linear. □

The construction in Definition 3.5 can be optimized if our security policy is a pure security automaton à la Schneider. The policy automaton for safety properties has all (but one) accepting states. The complementation will result in only one accepting state which is $(err, 1)$. However, the state can be collapsed with a non accepting state $(err, 0)$. Hence, no need to mark states with 0 and 1; and the only accepting state is (err). Furthermore, the complementation transitions remain as the original transitions.

Intersection of $\mathcal{AMT}$ $\mathcal{AMT}$ automaton can be considered as a Büchi automaton where infinite transitions are represented as finite transitions. Therefore, for $\mathcal{AMT}$ automata A^a, A^b, there is an $\mathcal{AMT}$ A^{ab} that accepts all the words which are accepted by both A^a, A^b synchronously. The A^{ab} can be constructed in a simple approach as in [48] as follows:

Definition 3.6 ($\mathcal{AMT}$ Intersection) *Let $\langle E^a, \mathcal{T}^a, \Sigma^a, S^a, {\mathbf{s_0}}^a, \Delta^a_{\mathcal{T}}, F^a\rangle$ and $\langle E^b, \mathcal{T}^b, \Sigma^b, S^b, {\mathbf{s_0}}^b, \Delta^b_{\mathcal{T}}, F^b\rangle$ be (non) deterministic $\mathcal{AMT}$, the $\mathcal{AMT}$* intersection automaton *$A^{ab} = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ is defined as follows:*

1. $E = E^a \cup E^b, \quad \mathcal{T} = \mathcal{T}^a \cup \mathcal{T}^b, \quad \Sigma = \Sigma^a \cup \Sigma^b,$
2. $S = S^a \times S^b \times \{1, 2\}, \quad \mathbf{s_0} = \langle {\mathbf{s_0}}^a, {\mathbf{s_0}}^b, 1\rangle, \quad F = F^a \times S^b \times \{1\},$
3. $$\Delta = \left\{ \left\langle (s^a, s^b, x), e^a \wedge e^b, (t^a, t^b, y)\right\rangle \;\middle|\; \begin{array}{l} (s^a, e^a, t^a) \in \Delta^a \text{ and} \\ (s^b, e^b, t^b) \in \Delta^b \text{ and} \\ DecisionProcedure(e^a \wedge e^b) = SAT \end{array} \right\}$$

$$y = \begin{cases} 2 & \text{if } x = 1 \text{ and } s^a \in F^a \text{ or} \\ & \text{if } x = 2 \text{ and } s^b \notin F^b \\ 1 & \text{if } x = 1 \text{ and } s^a \notin F^a \text{ or} \\ & \text{if } x = 2 \text{ and } s^b \in F^b \end{cases}$$

Proposition 3.2 *Let A^a, A^b be $\mathcal{AMT}$ over a set of valuations $\mathcal{V}$. Then an $\mathcal{AMT}$ A^{ab} constructed by Definition 3.6 accepts all the concrete runs which are accepted by A^a, A^b, that is A^{ab} is an intersection automaton such that $L_\omega(A^{ab}) = L_\omega(A^a) \cap L_\omega(A^b)$.*

Proof.
Correctness.

"$\supseteq$" we take an arbitrary concrete run accepted by A^{ab} that corresponds to a word $w = \langle \nu_1\nu_2\nu_3 \ldots \rangle$, where for all $i \geq 1$, ν_i satisfies $(e^a \wedge e^b)$, thus ν_i satisfies both e^a and e^b. Let the concrete run be $\langle (\mathbf{s_0}^a, \mathbf{s_0}^b, 1)\nu_1(s_1^a, s_1^b, x)\nu_2(s_2^a, s_2^b, x)\nu_3 \ldots \rangle$ of A^{ab}. This concrete run corresponds to $\langle \mathbf{s_0}^a\nu_1 s_1^a\nu_2 s_2^a\nu_3 \ldots \rangle$ of A^a, which is accepted by A^a because it goes infinitely often through $F^a \times S^b \times \{1\}$ thus it goes infinitely often through F^a. And $\langle \mathbf{s_0}^b\nu_1 s_1^b\nu_2 s_2^b\nu_3 \ldots \rangle$ of A^b is also accepting because whenever the automaton goes through an accepting state of A^b, the marker changes to 1 again. Thus, the acceptance condition guarantees that the run of the automaton visits accepting states of A^b infinitely often.

"$\subseteq$" we take an arbitrary concrete run $\langle \mathbf{s_0}^a\nu_1 s_1^a\nu_2 s_2^a\nu_3 \ldots \rangle$ accepted by A^a , where for all $i \geq 1$, ν_i satisfies e^a. And an arbitrary concrete run $\langle \mathbf{s_0}^b\nu_1 s_1^b\nu_2 s_2^b\nu_3 \ldots \rangle$ accepted by A^b, where for all $i \geq 1$, ν_i satisfies e^b. Both runs correspond to a word $w = \langle \nu_1\nu_2\nu_3 \ldots \rangle$. So, there is a concrete run $\langle (\mathbf{s_0}^a, \mathbf{s_0}^b, 1)\nu_1(s_1^a, s_1^b, x)\nu_2(s_2^a, s_2^b, x)\nu_3 \ldots \rangle$ of A^{ab} on w, where for all $i \geq 1$, ν_i satisfies $(e^a \wedge e^b)$ and whenever the automaton goes through an accepting state, the marker changes. Thus, the acceptance condition guarantees that the run of the automaton visits accepting states infinitely often, since a run accepts if and only if it goes infinitely often through $F^a \times S^b \times \{1\}$.

Termination This construction terminates because our states in S and formulas in E are finite.

Complexity The construction uses an oracle to an SMT solver to solve $DecisionProcedure(e^a \wedge e^b) = SAT$, where the theory $\mathcal{T}$ is decidable in the complexity class $\mathcal{C}$. Hence, the time and space complexity of the construction is $O(|S^a| \,.\, |S^b| \,.\, |\Delta_{\mathcal{T}}^a| \,.\, |\Delta_{\mathcal{T}}^b|)^{\mathcal{C}}$. □

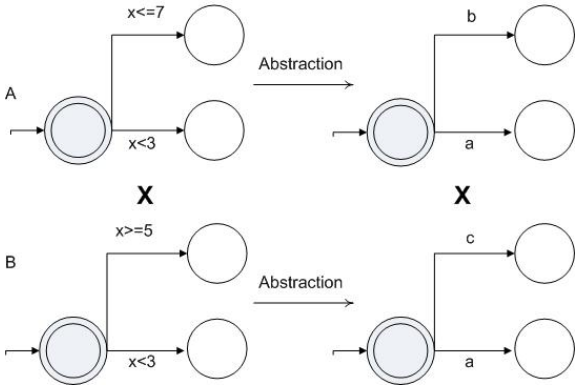
Intersection of automata illustrates another subtle difference with lazy satisfiability approach (based on boolean abstraction in SMT). For example, in Figure 5a, classically we have the result of automata intersection as in Figure 5c, however in $\mathcal{AMT}$ we have more transitions, as shown in Figure 5b.

Definition 3.6 is a general construction, as depicted on Figure 6a (see abbreviations on Fig. 6c). However, when we consider our domain of application, namely matching a mobile's policy and a midlet's contract, then the fact that we intersect a contract automaton with a special property (i.e. it has only one non accepting state (namely the error state)) and a complement of policy automaton which has also a special property (i.e. it has only one accepting state that is the error state), enable us to optimize the intersection such that we only consider correct contract transitions (shown in Figure 6b).

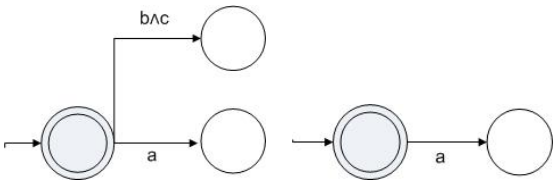
Emptiness problem of $\mathcal{AMT}$ An $\mathcal{AMT}$ automaton A is not empty when there exists some words accepted by A, meaning $L_\omega(A) \neq \emptyset$ if and only if there exists some accepting concrete run as defined in Definition 3.3.

Proposition 3.3 *Let the theory $\mathcal{T}$ be decidable with an oracle for the SMT problem in the complexity class $\mathcal{C}$ then:*

1. *The non-emptiness problem for $\mathcal{AMT}$ is decidable in $LIN - TIME^{\mathcal{C}}$.*
2. *The non-emptiness problem for $\mathcal{AMT}$ is $NLOG - SPACE^{\mathcal{C}}$-complete.*



(a) Example of Boolean Abstraction



(b) $\mathcal{AMT}$ Product (c) Normal Product

Figure 5: Boolean Abstraction

Proof. We proof Proposition 3.3 by showing that $L_\omega(A) \neq \emptyset$ if and only if there exists some accepting state which is connected to the initial state and also connected to itself as in [48]. Let $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F \rangle$.

Correctness.

"$\supseteq$" we assume that $L_\omega(A) \neq \emptyset$, meaning there exists an arbitrary concrete run $\sigma_C = \langle s_0 \nu_1 s_1 \nu_2 s_2 \ldots \rangle$ accepted by A that corresponds to a word $w = \langle \nu_1 \nu_2 \nu_3 \ldots \rangle$. By Definition 3.3 $\forall i \geq 0$ state s_i is directly connected to state s_{i+1}. Thus, when $i < k$ then s_i is connected to s_k. Furthermore, there exists some accepting state which is visited infinitely often, meaning that there is some $s_t \in F$ and there are i, k where $0 < i < k$ such that $s_t = s_i = s_k$. Hence, s_t is connected to the initial state s_0 and also connected to itself.

"$\subseteq$" we assume that there exists some accepting state $s_t \in F$ which is connected to the initial state and also connected to itself. So, there is a sequence of states $\langle s_{s0} s_{s1} s_{s2} \ldots s_{s2} \rangle$ from the initial state to $s_{sk} = s_t$ that corresponds to a word $\langle \nu_{s1} \nu_{s2} \nu_{s3} \ldots \nu_{sk} \rangle$ and $\forall i \geq 0$ state s_{si} is directly connected to state s_{si+1}. Furthermore, there are also sequences of states $\langle s_{t0} s_{t1} s_{t2} \ldots s_{tl} \rangle$ from $s_{t0} = s_t$ to $s_{tl} = s_t$ that corresponds to a word $\langle \nu_{t1} \nu_{t2} \nu_{t3} \ldots \nu_{tl} \rangle$ and $\forall i \geq 0$ state s_{ti} is directly connected to state s_{ti+1}. Thus $\langle \nu_{s1} \nu_{s2} \nu_{s3} \ldots \nu_{sk} \rangle \langle \nu_{t1} \nu_{t2} \nu_{t3} \ldots \nu_{tl} \rangle^\omega$ is accepted by A and $L_\omega(A) \neq \emptyset$.

Complexity The emptiness problem of $\mathcal{AMT}$ can be reduced to graph reachability. A combination of an algorithm based on Nested DFS [41] with a decision procedure for SMT can solve this problem. The algorithm takes as input the an $\mathcal{AMT}$ automaton

A and starts a depth first search procedure **check_safety** ($\mathbf{s_0}$) (Algorithm 1) over the initial state $\mathbf{s_0}$. When an accepting state in $\mathcal{AMT}$ is reached, we start a new depth first search (Algorithm 2)from the candidate state to determine whether it is in a cycle, in other words if it is reachable from itself. If it is, then report the automaton is non-empty.

When a state is first generated, it is marked as `safe`. During an unfinished search in Algorithm 1, a state is marked as `in_current_path`. When a state has finished its Algorithm 1 and not yet processed in Algorithm 2, it is marked as `safety_checked`. Finally, a state is marked `availability_checked` after processed by both algorithms.

This algorithm needs linear time on the size of A's states and an oracle to an SMT solver is used to solve $DecisionProcedure(e) = SAT$. Hence, it is in $LIN - TIME^{\mathcal{C}}$. It needs only a logarithmic memory, since at each step it needs to remember fewer states than the number of its total states and there are only two bits added to each state for the marker. Also, an SMT solver is used to solve $DecisionProcedure(e) = SAT$ and Jones [27] showed that graph reachability problem is $NLOG - SPACE$-hard. Hence, the emptiness problem of $\mathcal{AMT}$ is $NLOG - SPACE^{\mathcal{C}}$-complete. □

Language inclusion problem of $\mathcal{AMT}$ Language of an $\mathcal{AMT}$ automaton A^a is subsumed by the language of an $\mathcal{AMT}$ automaton A^b when for all the words $w = \langle \nu_1 \nu_2 \ldots \rangle$ (as defined in Definition 3.3) accepted by A^a, w is also accepted by A^b.

Proposition 3.4 *Let A^a, A^b be $\mathcal{AMT}$ over a set of valuations $\mathcal{V}$. Then $\mathcal{L}_{A^a} \subseteq \mathcal{L}_{A^b}$ such that A^b accepts all the concrete runs which are accepted by A^a is decidable.*

Proof. We proof Proposition 3.4 by showing that $\mathcal{L}_{A^a} \subseteq \mathcal{L}_{A^b}$ if and only if the language of $A^a \times \overline{A^b}$ is empty that is:
$\mathcal{L}_{A^a} \subseteq \mathcal{L}_{A^b} \Leftrightarrow \mathcal{L}_{A^a} \cap \overline{\mathcal{L}_{A^b}} = \emptyset \Leftrightarrow \mathcal{L}_{A^a} \cap \mathcal{L}_{\overline{A^b}} = \emptyset \Leftrightarrow \mathcal{L}_{A^a \times \overline{A^b}} = \emptyset$.

Correctness.

"$\supseteq$" we assume that there exists some concrete run which is accepted by A^a but not by A^b. Thus, $\mathcal{L}_{A^a \times \overline{A^b}}$ is not empty, which is a contradiction.

"$\subseteq$" we assume that $\mathcal{L}_{A^a \times \overline{A^b}}$ is not empty, meaning there exists some concrete runs accepted by $A^a \times \overline{A^b}$. Thus, this run is accepted by both A^a and $\overline{A^b}$. Because $L_\omega(\overline{A^b}) = \mathcal{V}^\omega - L_\omega(A^b)$, thus there exists some concrete run which is accepted by A^a but not by A^b, which is a contradiction.

Complexity Language inclusion problem of $\mathcal{AMT}$ is decidable follows from Proposition 3.1, Proposition 3.2, and Proposition 3.3 and derived the complexity from the afore mentioned propositions. □

The language inclusion problem of $\mathcal{AMT}$ (Proposition 3.4) is defined over concrete runs, thus in $\mathcal{AMT}$ symbolic language inclusion coincides with concrete one.

4 On-the-fly Language Inclusion Matching

In order to do matching between a contract with a security policy, our algorithm takes as input two automata Aut^C and Aut^P representing respectively the formal specification

of a contract and of a policy. A match is obtained when the language accepted by Aut^C (the execution traces of the midlet) is a subset of the language accepted by Aut^P (the acceptable traces for the policy). The matching problem can be reduced to an emptiness test: $\mathcal{L}_{\mathrm{Aut}^C} \subseteq \mathcal{L}_{\mathrm{Aut}^P} \Leftrightarrow \mathcal{L}_{\mathrm{Aut}^C} \cap \overline{\mathcal{L}_{\mathrm{Aut}^P}} = \emptyset \Leftrightarrow \mathcal{L}_{\mathrm{Aut}^C} \cap \mathcal{L}_{\overline{\mathrm{Aut}^P}} = \emptyset \Leftrightarrow \mathcal{L}_{\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P}} = \emptyset$. In other words, there is no behavior of Aut^C which is disallowed by Aut^P. If the intersection is not empty, then any behavior in it corresponds to a counterexample.

Constructing the product automaton explicitly is not practical for mobile devices. First, this can lead into an automaton too large for the mobile limited memory footprint. Second, to construct a product automata we need software libraries for the explicit manipulation and optimizations of symbolic states, which are computationally heavy and not available on mobile phones. Furthermore, we can exploit the explicit structure of the contract-policy as a number of separate requirements. Hence, we use on-the-fly emptiness test (constructing product automaton while searching the automata). The on-the-fly emptiness test can be lifted from the traditional algorithm by a technique from Coucubertis et al. [10] while modification of this algorithm from Holzmann et al's [25] is considered as state-of-the-art (used in Spin [26]). Gastin et al [16] proposed two modifications to [10] for finding faster and minimal counterexample.

Remark 4.1 *Our algorithm is tailored particularly for contract-policy matching, as such, it exploits a special property of $\mathcal{AMT}$ representing security policies, namely each automaton has only one non accepting state (the error state). The algorithm can be generalized by removing all specialized tests, for example on line 8 from Algorithm 3 $\cdots \wedge s^{\overline{\mathbf{P}}} = err^{\overline{\mathbf{P}}} \wedge \ldots$ can be replaced by accepting states from $\overline{Aut^P}$, and reporting only availability violation (corresponding to a non-empty automaton). This generic algorithm corresponds to on-the-fly algorithm for model checking of BA.*

We are now in the position to state our contract-policy matching's result using language inclusion:

Proposition 4.1 *Let the theory $\mathcal{T}$ be decidable with an oracle for the SMT problem in the complexity class $\mathcal{C}$ then:*

1. *The contract-policy matching problem for $\mathcal{AMT}$ using language inclusion is decidable in $LIN - TIME^{\mathcal{C}}$.*

2. *The contract-policy matching problem for $\mathcal{AMT}$ using language inclusion is decidable in $NLOG - SPACE^{\mathcal{C}}$-complete.*

Proof. We proof Proposition 4.1 by showing that $\mathcal{L}_{\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P}} = \emptyset$ if and only if there exists no accepting state of $\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P}$ which is connected to the initial state of $\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P}$ and also connected to itself where $\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P} = A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F \rangle$. Let A accepts all the concrete runs which are accepted by Aut^C and $\overline{\mathrm{Aut}^P}$, that is A is an intersection automaton such that $L_\omega(A) = L_\omega(\mathrm{Aut}^C) \cap L_\omega(\overline{\mathrm{Aut}^P})$.

Correctness.
The proof is similar to Proof 3.3, however we consider the product of two automata.

"$\supseteq$" we assume that $L_\omega(A) \neq \emptyset$, meaning there exists an arbitrary concrete run $\sigma_C = \langle s_0\nu_1 s_1\nu_2 s_2 \ldots\rangle$ accepted by A that corresponds to a word $w = \langle \nu_1\nu_2\nu_3 \ldots\rangle$ where for all $i \geq 1$, ν_i satisfies $(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}})$, thus ν_i also satisfies $e^{\mathbf{C}}$ and $e^{\overline{\mathbf{P}}}$. By Definition 3.3 $\forall i \geq 0$ state s_i is directly connected to state s_{i+1}. Thus, when $i < k$ then s_i is connected to s_k. Furthermore, there exists some accepting state which is visited infinitely often, meaning that there is some $s_t \in F$ and there are i, k where $0 < i < k$ such that $s_t = s_i = s_k$. Hence, s_t is connected to the initial state s_0 and also connected to itself.

"$\subseteq$" we assume that there exists some accepting state $s_t \in F$ which is connected to the initial state and also connected to itself. So, there is a sequence of states $\langle s_{s0} s_{s1} s_{s2} \ldots s_{s2}\rangle$ from the initial state to $s_{sk} = s_t$ that corresponds to a word $\langle \nu_{s1}\nu_{s2}\nu_{s3} \ldots \nu_{sk}\rangle$, where for all $i \geq 1$, ν_{si} satisfies $(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}})$, thus ν_{si} also satisfies $e^{\mathbf{C}}$ and $e^{\overline{\mathbf{P}}}$, and $\forall i \geq 0$ state s_{si} is directly connected to state s_{si+1}. Furthermore, there are also sequences of states $\langle s_{t0} s_{t1} s_{t2} \ldots s_{tl}\rangle$ from $s_{t0} = s_t$ to $s_{tl} = s_t$ that corresponds to a word $\langle \nu_{t1}\nu_{t2}\nu_{t3} \ldots \nu_{tl}\rangle$, where for all $i \geq 1$, ν_{ti} satisfies $(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}})$, thus ν_{ti} also satisfies $e^{\mathbf{C}}$ and $e^{\overline{\mathbf{P}}}$, and $\forall i \geq 0$ state s_{ti} is directly connected to state s_{ti+1}. Thus $\langle \nu_{s1}\nu_{s2}\nu_{s3} \ldots \nu_{sk}\rangle\langle \nu_{t1}\nu_{t2}\nu_{t3} \ldots \nu_{tl}\rangle^\omega$ is accepted by A and $L_\omega(A) \neq \emptyset$.

Complexity. The matching between a contract with a security policy problem can be reduced to an emptiness test of the product automaton of between a contract with a complement of security policy. A combination of an algorithm based on Nested DFS [41] with a decision procedure for SMT can solve this problem. The algorithm takes as input the midlet's claim and the mobile platform's policy and starts a depth first search procedure **check_safety** $(\mathbf{s_0}^{\mathbf{C}}, \mathbf{s_0}^{\overline{\mathbf{P}}}, 1)$ (Algorithm 3) over the initial state $(\mathbf{s_0}^{\mathbf{C}}, \mathbf{s_0}^{\overline{\mathbf{P}}}, 1)$. When an accepting state in $\mathcal{AMT}$ is reached, we have two cases. First, when the state contains an error state of complemented policy ($err^{\overline{\mathbf{P}}}$), then we report a security policy violation without further ado.[2] Second, the state does not contain an error state of complemented policy ($S^{\overline{\mathbf{P}}} \backslash \{err^{\overline{\mathbf{P}}}\}$). Then, we start a new depth first search (Algorithm 4)from the candidate state to determine whether it is in a cycle, in other words if it is reachable from itself. If it is, then we report an availability violation.

We use the same marking as in $\mathcal{AMT}$ emptiness check, namely when a state is first generated, it is marked as `safe`. During an unfinished search in Algorithm 3, a state is marked as `in_current_path`. When a state has finished its Algorithm 3 and not yet processed in Algorithm 4, then it is marked as `safety_checked`. Finally, a state is marked `availability_checked` when it has been processed by both Algorithm 3 and Algorithm 4. We also apply function **condition**(s, t, x, F_1, F_2) that implements marker signing of y given x and current states from the Definition 3.6 of $\mathcal{AMT}$ intersection.

This algorithm can be solved in linear time on the size of the number of the states of the product. In addition an oracle to an SMT solver is used to solve $DecisionProcedure(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}}) = SAT$. Hence, its complexity is $LIN - TIME^{\mathcal{C}}$.

The algorithm needs only a logarithmic memory, since at each step it needs to remember fewer states than the number of the total product states and there are

[2]The Error state is a convenient mathematical tool, but the trust assumption of the matching algorithm is that the code obeys the contract and therefore, it should never reach the error state where any action is permitted.

only two bits added to each state for the marker. Also, an SMT solver is used to solve $DecisionProcedure(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}}) = SAT$ and as in non-emptiness of $\mathcal{AMT}$ we have $NLOG - SPACE$-hardness follows from Jones [27] who showed that graph reachability problem is $NLOG - SPACE$-hard. Hence, the contract-policy matching problem of $\mathcal{AMT}$ is $NLOG - SPACE^{\mathcal{C}}$-complete. □

As we have shown, matching between a contract with a security policy problem can be reduced to an emptiness test of the product automaton of between a contract with a complement of security policy: $\mathcal{L}_{\mathrm{Aut}^C} \subseteq \mathcal{L}_{\mathrm{Aut}^P} \Leftrightarrow \mathcal{L}_{\mathrm{Aut}^C \times \overline{\mathrm{Aut}^P}} = \emptyset$. Furthermore, the set of infinite words recognized by an automaton A, denoted by $L_\omega(A)$, is the set of all accepting infinite traces in A ($w = \langle \nu_1 \nu_2 \ldots \rangle$). Because the language of an automaton A is defined in concrete level, thus the symbolic language coincides with the concrete language. Therefore, contract-policy matching using language inclusion in symbolic and concrete notion coincides.

5 Simulation

On the previous section we have seen on-the-fly matching using language inclusion and this approach requires complementation of the policy of the mobile platform. However, matching using language inclusion as in presented in Section 3 has a limitation in the structure of the policy automaton, i.e. only deterministic automaton. The constraint arises from the $\mathcal{AMT}$ complementation, where as BA complementation, the non-deterministic complementation is complex and exponentially blow-up in the state space [8]. Safra in [39], gives a better lower bound ($2^{O(n \log n)}$) for nondeterministic BA complementation, however it is still exponential(see [49]). This limitation does not evolve in matching using simulation as presented in this chapter, because using simulation approach we can also deal with nondeterministic automata.

The notion of *simulation* in $\mathcal{AMT}$ is both *fair* and *symbolic*. The fairness in $\mathcal{AMT}$ is similar to *fair simulation* in Büchi automata as in [23]. A system fairly simulates another system if and only if in the simulation game, there is a strategy that matches each fair computation of the simulated system with a fair computation of the simulating system. Efficient algorithms for computing a variety of simulation relations on the state space of a Büchi automaton were proposed in [14] using parity game framework, that is based on small progress measures [28]. Another algorithm based on the notion of fair simulation was presented in [18]. The *symbolism* in $\mathcal{AMT}$ is similar to the theory of symbolic bi-simulation for the π-calculus [22]. This symbolic representation can express the operational semantics of many value-passing processes in terms of finite symbolic transition graphs despite the infinite underlying labeled transitions graph.

In the sequel we will use s to denote states of the application's contract and t to denote state of the platform's policy.

Definition 5.1 (Concrete Fair Compliance Game) *Let A^c and A^p be $\mathcal{AMT}$ with initial states s_0 and t_0 respectively. A* Concrete Fair Compliance Game $G^C_{A^c,A^p}(s_0, t_0)$ *is played by two players,* `Contract` *and* `Policy`*, in rounds.*

1. *In the first round* `Contract` *is on the initial state $s_0 \in S^c$ and* `Policy` *is on the initial state $t_0 \in S^p$.*

2. `Contract` *chooses a transition* $\langle s_i, e_i^c, s_{i+1}\rangle \in \Delta_{\mathcal{T}}^c$ *with a valuation* ν_i *represents* α_i *and* $\mathcal{I}(e_i)$ *such that* $(\mathcal{M}, \alpha_i) \models e_i^c$ *and moves to* s_{i+1}.

3. `Policy` *responds by a transition* $\langle t_i, e_i^p, t_{i+1}\rangle \in \Delta_{\mathcal{T}}^p$ *such that* $(\mathcal{M}, \alpha_i) \models e_i^p$ *and moves to* t_{i+1}.

The winner of the game is determined by the following rules:

- *If the* `Contract` *cannot move then* `Policy` *wins.*
- *If the* `Policy` *cannot move then* `Contract` *wins.*
- *Otherwise there are two infinite concrete runs* $\overrightarrow{s} = \langle s_0\nu_1 s_1\nu_2 s_2 \ldots\rangle$ *and* $\overrightarrow{t} = \langle t_0\nu_1 t_1\nu_2 t_2 \ldots\rangle$ *respectively of* A^c *and* A^p. *If* $\overrightarrow{s} = \langle s_0\nu_1 s_1\nu_2 s_2 \ldots\rangle$ *is an accepting concrete run for* A^c *and* $\overrightarrow{t} = \langle t_0\nu_1 t_1\nu_2 t_2 \ldots\rangle$ *is not an accepting concrete run for* A^p *then* `Contract` *wins. In other cases,* `Policy` *wins.*

Intuitively in the compliance game, the `Contract` tries to make a concrete move and the `Policy` follows accordingly to show that the `Contract` move is allowed. If the `Policy` cannot move then `Contract` is not compliant, meaning there is a move that the `Policy` can not do, that is that particular action is a violation.

Example 5.1 *In a game between the* `Contract` *from Figure 4a and the* `Policy` *from Figure 4b, the* `Contract` *can choose to invoke the url* `http://www.google.com` *and the* `Policy` *can respond by selecting the appropriate expression which is satisfied by that valuation.*

A more complex situation occurs in the infinite case where infinite runs correspond to liveness properties, i.e. something good will eventually happen. An example of this property is shown in Example 2.3. In this case, the `Contract` only wins (i.e. it breaks the Policy) when according to its view of the world there are infinitely many good things but not for the `Policy` which after some initial good things is trapped in an endless sequence of unsatisfactory states.

Example 5.2 *In a game between the* `Contract` *and* `Policy` *from Ex.2.3, the* `Contract` *can choose to invoke the url* `https://sourceforge.net` *in a certain step after in some previous steps it invokes permission* `io.Connector.https`. *The* `Policy` *can respond by selecting the appropriate expression which is also satisfied by the same valuation, which is possible in the game if* `Policy` *has previously requested permission* `io.Connector.https`.

The *concrete strategy* for `Policy` in game $G^C_{A^c,A^p}(s_0, t_0)$ is a partial function that determines its next move given the history of the concrete game up to a certain point.

Definition 5.2 (Concrete Strategy) *A partial function* $f : S^c \times (S^p \times \nu \times S^c)^* \to S^p$ *is a* concrete strategy *if for any sequence* $\langle s_0\nu_1 s_1\nu_2 \ldots s_i\nu_i s_{i+1}\rangle$ *which is a valid concrete run for* A^c

- $f(s_0) = t_0$

- $f(\langle s_0 t_0 \nu_1 s_1 \dots s_i t_i \nu_{i+1} s_{i+1} \rangle) = t_{i+1}$ *such that* $\langle t_i, e_i^p, t_{i+1} \rangle \in \Delta_{\mathcal{T}}^p$ *and* $(\mathcal{M}, \alpha_i) \models e_i^p$, *where* ν_i *represents* α_i *and* $\mathcal{I}(e_i)$.

A concrete strategy f of a game is a `Policy` *winning strategy* if and only if whenever a `Policy` selects the moves of game as in Definition 5.1 according to f then `Policy` wins.

Definition 5.3 ($\mathcal{AMT}$ Concrete Fair Simulation Relation) *An automaton* A^p concretely fair simulates *an automaton* A^c *if and only if there is a concrete winning strategy for* A^p *we denote as* $A^c \sqsubseteq A^p$. *We also say that* A^c *complies with* A^p.

We have now the machinery to generalize the notion of simulation to symbolic level, among expressions.

Definition 5.4 ($\mathcal{AMT}$ Fair Compliance Game) *A* Fair Compliance Game $G_{A^c, A^p}(s_0, t_0)$ *is played by two players,* `Contract` *and* `Policy`, *in rounds.*

1. *In the first round* `Contract` *is on the initial state* $s_0 \in S^c$ *and* `Policy` *is on the initial state* $t_0 \in S^p$.
2. `Contract` *chooses a transition* $\langle s_i, e_i^c, s_{i+1} \rangle \in \Delta_{\mathcal{T}}^c$ *such that* e_i^c *is satisfiable and moves to* s_{i+1}.
3. `Policy` *responds by a transition* $\Delta_{\mathcal{T}}^p(t_i, e_i^p, t_{i+1})$ *such that* $(e_i^c \rightarrow e_i^p)$ *is valid and moves to* t_{i+1} [3].

The winner of the game is determined by the rules as in Definition 5.1 with the difference in run where we define run over expressions instead of assignments.

The intuition is similar to concrete game: `Contract` tries to make a symbolic move and the `Policy` follows suit in order to show that the `Contract` move is allowed. If the `Policy` cannot move this means that the `Contract` may not be compliant because there is a symbolic move that the `Policy` could not do. However, as we shall see this might not imply that at the concrete level the `Contract` is really non-compliant.

Definition 5.5 (Strategy) *A partial function* $f : S^c \times (S^p \times E \times S^c)^* \rightarrow S^p$ *is a* symbolic strategy *if and only if for any sequence* $\langle s_0 e_0^c s_1 e_1^c \dots s_i e_i^c s_{i+1} \rangle$ *which is a valid symbolic run for* A^c

- $f(s_0) = t_0$
- $f(\langle s_0 t_0 e_0^c s_1 t_1 e_1^c \dots s_i t_i e_i^c s_{i+1} \rangle) = t_{i+1}$ *such that* $\Delta_{\mathcal{T}}^p(t_i, e_i^p, t_{i+1})$ *and* $(e_i^c \rightarrow e_i^p)$ *is valid.*

A strategy f of the game is a `Policy` *winning strategy* if and only if whenever a `Policy` select the moves of game as in Definition 5.4 according to f then `Policy` wins.

[3] $\rightarrow$ in $(e_i^c \rightarrow e_i^p)$ represents implication symbol in first order logic.

Definition 5.6 ($\mathcal{AMT}$ Fair Simulation Relation) *An automaton A^p* fair simulates *an automaton A^c if and only if there is a winning strategy for A^p we denote as $A^c \leq A^p$. We also say that A^c complies with A^p.*

Proposition 5.1 *If $A^c \leq A^p$ is an $\mathcal{AMT}$ fair simulation relation then $A^c \sqsubseteq A^p$ is a concrete fair simulation relation.*

Proof.
Assume that $A^c \leq A^p$ is an $\mathcal{AMT}$ fair simulation relation. By Definition 5.6 there is a winning strategy for A^p, such that whenever a `Policy` select the moves of game defined in Definition 5.4 according to strategy f then `Policy` wins the game. We construct a concrete strategy f' from f.

By Definition 5.4 there are two cases where `Policy` wins the game:

- Finite game: If the `Contract` cannot move then `Policy` wins.
  `Contract` moves by choosing a transition $\langle s_i, e_i^c, s_{i+1}\rangle \in \Delta_{\mathcal{T}}^c$ such that e_i^c is satisfiable. `Contract` cannot move means that there exists no valuations and by Definition 5.1 in concrete game `Contract` cannot move either.

- Infinite game: There are infinitely many j such that $t_j \in F^p$ or there are only finitely many i such that $s_i \in F^c$.
  The compliance game has infinitely many j such that $t_j \in F^p$ when `Policy` is able to respond infinitely often by a transition $\Delta_{\mathcal{T}}^p(t_j, e_j^p, t_{j+1})$ where $(e_j^c \rightarrow e_j^p)$ is valid, meaning for all α_j, $(\mathcal{M}, \alpha_j) \models (e_j^c \rightarrow e_j^p)$. And by Definition 5.1 with $(\mathcal{M}, \alpha_j) \models e_j^p$, `Policy` can respond by a transition $\left\langle t_j, e_j^p, t_{j+1}\right\rangle \in \Delta_{\mathcal{T}}^p$.
  Finitely many i occurs when there is some k such that $\forall i > k, s_i \notin F^c$, meaning `Contract` moves by choosing a transition $\langle s_i, e_i^c, s_{i+1}\rangle \in \Delta_{\mathcal{T}}^c$ such that e_i^c is satisfiable, i.e. there exist α_i where $(\mathcal{M}, \alpha_i) \models e_i^c$ and by Definition 5.1 `Contract` can also move in concrete game.

It is clear that the constructed concrete strategy f' is a winning strategy for A^p in concrete compliance game, hence by Definition 5.3 $A^c \sqsubseteq A^p$. □

In contrast to the language inclusion approach discussed in Section 3.3, where symbolic language inclusion coincides with concrete language inclusion, and also the simulation notions of [22], the converse of Proposition 5.1 does not hold in general.

Proposition 5.2 *$\mathcal{AMT}$ fair simulation is stronger than $\mathcal{AMT}$ language inclusion.*

Proof. The pair of automata in Figure 7b and Figure 7a is a simple counter example. We can see that both automata coincide with the same concrete automaton as in Figure 7c. Thus in concrete level the same automaton having not just simulation but also bi-simulation to itself. However, the symbolic $\mathcal{AMT}$ on Figure 7a cannot simulate the symbolic $\mathcal{AMT}$ on Figure 7b. For example if we have policy represented as Figure 7b and contract represented as Figure 7a, where both automata accept the same language but according to simulation $VALID(e^2 \rightarrow e^{11})$ does not hold nor $VALID(e^2 \rightarrow e^{12})$, thus we do not have simulation (see abbreviation in Figure 7d). □

In order to show that $\mathcal{AMT}$ simulation coincides with concrete simulation we must impose some additional syntactic constraints on the automaton.

Definition 5.7 (Normalized $\mathcal{AMT}$) *$A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ is a* normalized automaton *modulo theory $\mathcal{T}$ if and only if for every $s, s_1 \in S$ there is at most one expression $e_1 \in E$ such that $s_1 \in \Delta_{\mathcal{T}}(s, e_1)$.*

For example Figure 7a is a normalized automaton while Figure 7b is not normalized.

Lemma 5.1 *It is possible to normalize an $\mathcal{AMT}$ automaton $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ when theory $\mathcal{T}$ is convex and closed under disjunction.*

Proof. A theory $\mathcal{T}$ is convex [42] if all the conjunctions of literals are convex in theory $\mathcal{T}$. A conjunction of $\mathcal{T}$-literals in a theory $\mathcal{T}$ is convex if for each disjunction $(\mathcal{M}, \alpha) \models \bigvee_{i=1}^{n} e_i$ if and only if $(\mathcal{M}, \alpha) \models e_i$ for some i, where e_i are equalities between variables occurring in $(\mathcal{M}, \alpha)$. If a theory $\mathcal{T}$ is convex then we can normalize an automaton by considering the disjunction of all expressions going to the same state.

A theory $\mathcal{T}$ is called closed under disjunction if disjunctions of $\mathcal{T}$-formulas $\bigvee_{i=1}^{n} e_i$, where e_i are $\mathcal{T}$-formulas, is also a $\mathcal{T}$-formula. For most theories this closure holds. An example where the closure does not hold is when a $\mathcal{T}$ consists of only Horn-formulas that allows at most one positive literal. Suppose we have two Horn-formulas e_1 and e_2, where $e_1 \doteq p_1 \wedge p_2 \rightarrow p$ and $e_2 \doteq q_1 \wedge q_2 \rightarrow q$, then $e_1 \vee e_2 \doteq p_1 \wedge p_2 \wedge q_1 \wedge q_2 \rightarrow p \vee q$ which is not a Horn-formula. □

Lemma 5.2 *Normalization preserves the determinism of an $\mathcal{AMT}$.*

Proof. By Definition 3.4 $A = \langle E, \mathcal{T}, \Sigma, S, \mathbf{s_0}, \Delta, F\rangle$ is a *deterministic automaton* modulo theory $\mathcal{T}$, if and only if, for every $s \in S$ and every $s_1, s_2 \in S$ and every $e_1, e_2 \in E$, if $(s, e_1, s_1) \in \Delta$ and $(s, e_2, s_2) \in \Delta$, where $s_1 \neq s_2$ then the expression $(e_1 \wedge e_2)$ is unsatisfiable in the Σ-theory $\mathcal{T}$.

Let $(s, e_{1j}, s_1) \in \Delta$ where $j \in \{1, \ldots, m\}$, and let $(s, e_{2k}, s_2) \in \Delta$ where $k \in \{1, \ldots, n\}$, and $s_1 \neq s_2$. Thus, each expression $(e_{1j} \wedge e_{2k})$ is unsatisfiable in the Σ-theory $\mathcal{T}$. By normalization we have $(\bigvee_{j=1}^{m} e_{1j})$ and $(\bigvee_{k=1}^{n} e_{2k})$, where $(\bigvee_{j=1}^{m} e_{1j}) \wedge (\bigvee_{k=1}^{n} e_{2k}) \Leftrightarrow \forall j \in \{1, \ldots, m\}, \forall k \in \{1, \ldots, n\}, (e_{1j} \wedge e_{2k})$. If each expression $(e_{1j} \wedge e_{2k})$ is unsatisfiable then $(\bigvee_{j=1}^{m} e_{1j}) \wedge (\bigvee_{k=1}^{n} e_{2k})$ is also unsatisfiable when the theory $\mathcal{T}$ is convex. Thus, normalization preserves the determinism of an $\mathcal{AMT}$. □

Proposition 5.3 *For normalized $\mathcal{AMT}$ if $A^c \sqsubseteq A^p$ is a concrete fair simulation relation then $A^c \leq A^p$ is an $\mathcal{AMT}$ fair simulation relation.*

Proof.
Assume that $A^c \sqsubseteq A^p$ is a concrete fair simulation relation. By Definition 5.3 there is a winning strategy for A^p, such that whenever a `Policy` select the moves of game defined in Definition 5.1 according to strategy f then `Policy` wins the game. We construct a concrete strategy f' from f.

By Definition 5.1 there are two cases where `Policy` wins the game:

- Finite game: If the `Contract` cannot move then `Policy` wins.
  `Contract` moves by choosing a transition $\langle s_i, e_i^c, s_{i+1}\rangle \in \Delta_{\mathcal{T}}^c$ with a valuation ν_i represents α_i and $\mathcal{I}(e_i)$ such that $(\mathcal{M}, \alpha_i) \models e_i^c$, meaning e_i^c is satisfiable. `Contract` cannot move means that there exists no valuations and by Definition 5.4 in compliance game `Contract` cannot move either.

- Infinite game: There are infinitely many j such that $t_j \in F^p$ or there are only finitely many i such that $s_i \in F^c$.
  The concrete compliance game has infinitely many j such that $t_j \in F^p$ when `Policy` is able to respond infinitely often by a transition $\Delta^p_{\mathcal{T}}(t_j, e^p_j, t_{j+1})$ where for all valuations ν_j represents α_j and $\mathcal{I}(e_j)$ such that $(\mathcal{M}, \alpha_j) \models (e^c_j \rightarrow e^p_j)$, meaning $(e^c_j \rightarrow e^p_j)$ is valid. And by Definition 5.4 `Policy` can respond by a transition $\left\langle t_j, e^p_j, t_{j+1} \right\rangle \in \Delta^p_{\mathcal{T}}$ with a valuation ν_j represents α_j and $\mathcal{I}(e_j)$ such that $(\mathcal{M}, \alpha_j) \models e^p_j$.
  Finitely many i occurs when there is some k such that $\forall i > k, s_i \notin F^c$, meaning `Contract` moves by choosing a transition $\langle s_i, e^c_i, s_{i+1} \rangle \in \Delta^c_{\mathcal{T}}$ with a valuation ν_i represents α_i and $\mathcal{I}(e_i)$ such that $(\mathcal{M}, \alpha_i) \models e^c_i$ and by Definition 5.4 `Contract` can also move in concrete game.

It is clear that the constructed strategy f' is a winning strategy for A^p in compliance game, hence by Definition 5.6 $A^c \leq A^p$. □

If automata are in normalized form then we have the following theorem from [33]:

Theorem 5.1 *For normalized $\mathcal{AMT}$ $A^c \leq A^p$ is an $\mathcal{AMT}$ fair simulation if and only if $A^c \sqsubseteq A^p$ is a concrete fair simulation.*

Proof.
"$\supseteq$" By Proposition 5.1.
"$\subseteq$" If a normalization that preserves automata determinism (Lemma 5.2) is possible (Lemma 5.1), then By Proposition 5.3. □

6 Simulation Matching

In this section we describe a different algorithm for matching from Section 4 that uses the concepts of language inclusion. Here we use fair simulation for matching and adapts the Jurdziński's algorithm on parity games [28]. The simulation algorithm Algorithm 5 takes as input two automata Aut^C and Aut^P representing respectively the formal specification of a contract and of a policy. A match is obtained when every APIs invoked by Aut^C can also be invoked by Aut^P. In other words, every behavior of Aut^C is also a behavior of Aut^P.

At the first step (line 1) a compliance game graph $G = \langle V_1, V_0, E, l \rangle$ is constructed out of automata Aut^C and Aut^P. A compliance game graph can be formally defined as follows:

Definition 6.1 (Compliance Graph) *Given $\langle E^c, \mathcal{T}^c, \Sigma^c, S^c, \mathbf{s_0}^c, \Delta^c_{\mathcal{T}}, F^c \rangle$ and $\left\langle E^p, \mathcal{T}^p, \Sigma^p, S^p, \mathbf{s_0}^p, \Delta^p_{\mathcal{T}}, F^p \right\rangle$, construct a $\langle V_1, V_0, E, l \rangle$ as follows:*

- $V_1 = \{v_{(s^c, s^p)} | s^c \in S^c, s^p \in S^p\}$
- $V_0 = \{v_{(s^c, s^p, e^c)} | s^c \in S^c, s^p \in S^p, \exists r^c . s^c \in \Delta^c_{\mathcal{T}}(r^c, e^c)\}$
- $E = \{(v_{(s^c, s^p, e^c)}, v_{(s^c, t^p)}) | t^p \in \Delta^c_{\mathcal{T}}(s^p, e^p) \wedge VALID(e^c \rightarrow e^p)\} \cup$
  $\{(v_{(s^c, s^p)}, v_{(t^c, s^p, e^c)}) | t^c \in \Delta^c_{\mathcal{T}}(s^c, e^c)\}$

- $$l(v) = \begin{cases} 0 & \textit{if } v = v_{(s^c,s^p)} \textit{ and } s^p \in F^p \\ 1 & \textit{if } v = v_{(s^c,s^p)} \textit{ and } s^c \in F^c \textit{ and } s^p \notin F^p \\ 2 & \textit{otherwise} \end{cases}$$

A compliance graph G *is the tuple* $\langle V_1, V_0, E, l\rangle$

Intuitively the compliance level l(v) is 0 when the simulating automaton accepts, 1 when the simulated automaton accepts (but the simulating automaton has not accepted yet) and 2 when neither of them accepts. V_1 consists of $v_{(s^c,s^p)}$ where Aut^C is on s^c and Aut^P is on s^p and it is `Contract` turn to move. V_0 consists of $v_{(s^c,s^p,e^c)}$ where Aut^C is on s^c and Aut^P is on s^p, `Contract` just made a move e^c and it is `Policy` turn to move such that $VALID(e^c \to e^p)$ by querying to an oracle for the SMT solver.

Lemma 6.1 *Let* $Aut^C = \left\langle E^{Aut^C}, \mathcal{T}^{Aut^C}, \Sigma^{Aut^C}, S^{Aut^C}, \mathbf{s_0}^{Aut^C}, \Delta_{\mathcal{T}}^{Aut^C}, F^{Aut^C} \right\rangle$ *and* $Aut^P = \left\langle E^{Aut^P}, \mathcal{T}^{Aut^P}, \Sigma^{Aut^P}, S^{Aut^P}, \mathbf{s_0}^{Aut^P}, \Delta_{\mathcal{T}}^{Aut^P}, F^{Aut^P} \right\rangle$ *be* $\mathcal{AMT}$ *automata and let the theory* $\mathcal{T} = \mathcal{T}^{Aut^C} \cup \mathcal{T}^{Aut^P}$ *be decidable with an oracle for the SMT problem in the complexity class* $\mathcal{C}$

1. $|G = \langle V_1, V_0, E, l\rangle|$ *constructed out of automata* Aut^C *and* Aut^P *by Definition 6.1 is in* $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)^{\mathcal{C}}$
2. $\left|l^{-1}(1)\right|$ *defined as in Definition 6.1 is in* $O(|S^c| \,.\, |S^p|)$

Proof. We prove part 1 by computing the vertices and edges of $\langle V_1, V_0, E, l\rangle$

- $|V_1|$ is in $O(|S^c| \,.\, |S^p|)$
- $|V_0|$ is in $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)$
- $|E|$ is in $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)^{\mathcal{C}}$ because an edge exists from a node in V_0 to a node in V_1 when $VALID(e^c \to e^p)$ that needs a call to oracle for the SMT solver.

Thus, we can conclude that $|G = \langle V_1, V_0, E, l\rangle|$ is in $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)^{\mathcal{C}}$

For part 2 vetices with $l = (1)$ are contained in V_1, thus $\left|l^{-1}(1)\right|$ is in $O(|S^c| \,.\, |S^p|)$ □

A compliance game $P(G, v_0)$ on G starting at $v_0 \in V$ is played by two players `Policy` (for Aut^P) and `Contract` (for Aut^C). The game starts by placing pebble on v_0. At round i with pebble on v_i, $v_i \in V_0(V_1)$, `Policy` (`Contract` resp.) plays and moves the pebble to v_{i+1} such that $(v_i, v_{i+1}) \in E$. The player who cannot move loses. For infinite play $\pi = v_0 v_1 v_2 \ldots$, the winner defined as the minimum compliance level that occurs infinitely often, namely if the minimum compliance level is 0 or 2 then `Policy` wins, otherwise `Contract` wins.

Next, we define a *compliance measure* $\mu : V \to \left\{x | x \leq |l^{-1}(1)|\right\} \cup \{\infty\}$. μ ranges from 0 to $|l^{-1}(1)|$ because at l(v)=1 the simulated automaton (contract) accepts but the simulating automaton (policy) has not accepted yet. Thus, progressing the measure has the analogy of computing the pre-fixed point where the `Contract` remains winning and ∞ shows that the μ keeps progressing beyond this limit, meaning `Contract` wins

the game. If $l(v) = 1$, then $\mu(v) > \mu(w)$, where $|l^{-1}(1)| + 1 = \infty$. If $l(v) = 2$ or $l(v) = 0$, then $\mu(v) \geq \mu(w)$.

The compliance measure for each node is the number of potential bad nodes, namely nodes where the contract accepts but the policy does not, that it can reach. Thus, $\mu(v) = \infty$ means that there is an infinite path where policy cannot return to compliance level 0. We slighty modify the Jurdziński progress measure [28] to *compliance measure* where instead of a pair $(0, x)$ we only use x. This is due to our observation of our domain where we only have three priorities, namely $l(v) \in 0, 1, 2$ thus for ordering $(0, x) \geq_{l(v)} (0, x')$ the first component will not effect the ordering.

Jurdziński's algorithm on parity games [28] defines that `Policy` has a winning strategy from precisely the vertices v where after its lifting algorithm halts has $\mu(v) < \infty$. However, in contract-policy matching we are interested when there is a winning strategy from the initial vertex $v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}$, depicted in Algorithm 5 as $\mu(v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}) < \infty$.

Proposition 6.1 *Let G be a parity game constructed from two $\mathcal{AMT}$ automata Aut^C and Aut^P constructed as in Definition 6.1.* `Policy` *has a winning strategy from the initial vertex $v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}$ when Algorithm 5 halts with $\mu(v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}) < \infty$.*

Proof. **Correctness.**
The correctness derived from Jurdziński's algorithm on parity games [28]. Jurdziński defined a parity game between two players and defining and even player (in our case `Policy`) wins when the lowest priority occuring infinitely often in the play is even (in our case `Policy` can return to compliance level 0 infinitely often). He proposed computing the game using progress measure which is defined as $M_G = [1] \times [n_1 + 1]\,[1] \times [n_3 + 1] \times \ldots \times [1] \times [n_{d-1} + 1]$, where d is the maximum priority in the game. In our setting, we slighty modify the Jurdziński progress measure [28] to *compliance measure* where instead of a pair $(0, x)$ we only use x. As we have mentioned afore, this is due to our observation of our domain where we only have 3 priorities, namely $l(v) \in 0, 1, 2$ thus for ordering $(0, x) \geq_{l(v)} (0, x')$ the first component will not effect the ordering.

Jurdziński reasoned that each vertex can only be lifted $|M_G|$ times. This lifting procedure is implemented in Algorithm 5 presented as a loop where compliance measure progressing until reaching a pre-fixed point ($\mu = \mu_{\texttt{new}}$). He also defined that Even has a winning strategy from precisely the vertices v where after its lifting algorithm halts has $\mu(v) < \infty$. However, in contract-policy matching we are interested when there is a winning strategy from the initial vertex $v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}$. Thus, in Algorithm 5 `Policy` wins when $\mu(v_{(\mathbf{s_0}^c, \mathbf{s_0}^p)}) < \infty$.

Termination. This parity game terminates because each vertex can only be lifted $|M_G|$ times.

Complexity. Lifting procedure in Jurdziński [28] has time complexity $O\left(\sum_{v \in V} d.od(v).|M_G|\right) = O(d.m.|M_G|)$ where d is the maximum priority in the game, m the number of edges, $od(v)$ the degree outgoing edges from v, and V is the set of vertices in the game graph. He reasoned that for every vertex v with outgoing edges degree $od(v)$ and the tuple of progress measure has the length of maximum priority d can only be lifted $|M_G|$ times:

$|M_G| = \prod_{i=1}^{\lfloor d/2 \rfloor}(n_{2i-1}+1) \leq \left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}$, where d is the maximum priority in the game.

In our setting, d equals to two, because our compliance measure is in $\{0, 1, 2\}$. Thus, $|M_G| = [n_1 + 1] = |l^{-1}(1)| + 1 \leq |V_1|$ and from Lemma 6.1 $|V_1| = O(|S^c| \,.\, |S^p|)$. In addition, the number of edges $|E|$ is in $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)^{\mathcal{C}}$ (from Lemma 6.1). Thus, the time complexity of Algorithm 5 is $O(2. \,|E| \,.|M_G|)$

Lifting procedure in Jurdziński [28] has space complexity $O(dn)$ where d is the maximum priority in the game and n the number of vertices in the game graph. He reasoned that every vertex v in the game graph only needs to keep the compliance measure, which is a d-tuple of integers. In our setting, d equals to two because our compliance measure is in $\{0, 1, 2\}$, however our compliance measure only use an integer x instead of a 2-tuple $(0, x)$. As we have mentioned afore, this is due to our observation of our domain where we only have 3 priorities, namely $l(v) \in 0, 1, 2$ thus for ordering $(0, x) \geq_{l(v)} (0, x')$ the first component will not effect the ordering. In addition, from Lemma 6.1 $|V_1| = O(|S^c| \,.\, |S^p|)$ and $|V_0|$ is in $O(|S^c| \,.\, |S^p| \,.\, |\Delta_{\mathcal{T}}^c|)$ where the total number of vertices equals to $V = |V_1| + |V_0|$. Thus, the space complexity of Algorithm 5 is $O(|V|)$. □

We are now in the position to state our contract-policy matching's result using fair simulation:

Proposition 6.2 *Let the theory $\mathcal{T}$ be decidable with an oracle for the SMT problem in the complexity class $\mathcal{C}$ then:*

1. *The contract-policy matching problem for $\mathcal{AMT}$ using fair simulation is decidable in time $O(2. \,|E| \,.|M_G|)$.*
2. *The contract-policy matching problem for $\mathcal{AMT}$ using fair simulation is decidable in space $O(|V|)$.*

Proof. The matching between a contract with a security policy problem can be reduced to a fair simulation between a contract with a security policy. A combination of an algorithm based on Jurdziński's algorithm on parity games [28] with a decision procedure for SMT given in Algorithm 5 can solve this problem in time $O(2. \,|E| \,.|M_G|)$ and in space $O(|V|)$. The algorithm takes as input the midlet's claim and the mobile platform's policy and constructs compliance game graph $G = \langle V_1, V_0, E, l \rangle$. The correctness and complexity follow from Proposition 6.1. □

7 Related Work and Conclusions

Security monitors were implemented in several systems, for example the PoET / PSLang toolkit [13], that can enforce security policies whose transitions pattern-match event symbols using regular expressions. Edit automata [3] are another model for achieving this. Edit automata are implemented in the Polymer system [4] to dynamically compose security automata. Most recently, the Mobile system [21] implements a linear decision algorithm that verifies that annotated .NET binaries satisfy a class of policies, which includes security automata and edit automata. All mentioned approaches focus on the

relations between code and security claims on the code (which we call contracts); $\mathcal{AMT}$ focuses between the security claims of the code and the desired security behavior for the platform. Other works fit into in-lining and run-time monitoring in our workflow, while $\mathcal{AMT}$ falls into matching a contract and policies.

Model-carrying code [43] and security-by-contract [11] proposed to augment mobile code with a claim on its security behavior that can be matched against a mobile platform policy before downloading the code. In [43] and in other companion papers only finite automata have been proposed and they are too simple to express even the most basic security requirement occurring in practice: a basic security policy such as only allows connections starting with "https://" already generates an infinite automaton.

Automata Modulo Theory ($\mathcal{AMT}$) [32] enables systems formalization with finitely many states but infinitely many transitions. As we already showed in [32], it is possible to define very expressive (essentially infinite) policies that can capture realistic scenarios, while keeping the task of matching computationally tractable. $\mathcal{AMT}$ maps the problem into a variant of on-the-fly product and emptiness tests from automata theory, without symbolic manipulation procedures of zones and regions nor finite representation of equivalence classes. The tractability limit is essentially the complexity of the satisfiability procedure for the theories, called as subroutines. The prototype for matching policies with security claims of mobile applications using $\mathcal{AMT}$ appeared in [6].

Infinite numbers of transitions in security policies, by labeling each transition with a computable predicate instead of an atomic symbol, have been studied in [40]. Security automata á la Schneider can also be mapped to a particular form of $\mathcal{AMT}$ (with all accepting states and an error absorbing state) for which particular optimizations are possible. Security automata specify transitions as a function of the input symbols that can be the entire system state. However, $\mathcal{AMT}$ dif and only ifers from security automata in transitions which are environmental parameters rather than system states.

A theory of symbolic bi-simulation for the π-calculus was proposed in [22] which has the advantage of expressing the operational semantics of many value-passing processes in terms of finite symbolic transition graphs despite the infinite underlying labeled transitions graph.

A new view of fair simulation by extending the local definition of simulation to account for fairness [23] proposed the notion of simulation game. A system fairly simulates another system if and only if in the simulation game, there is a strategy that matches with each fair computation of the simulated system a fair computation of the simulating system. Efficient algorithms for computing a variety of simulation relations on the state space of a Büchi automaton were proposed in [14] using parity game framework, based on solving parity games using small progress measures as appeared in [28]. An algorithm based on the notion of fair simulation was presented in [18] applied for the minimization of Büchi automata.

7.1 Conclusions

In order to capture realistic scenarios with potentially infinite transitions (e.g. "only connections to urls starting with https") we have proposed to represent those policies with the notion of *Automata Modulo Theory* ($\mathcal{AMT}$), an extension of Büchi Automata (BA), with edges labeled by expressions in a decidable theory and defined the theory

and algorithm for extending simulation results to $\mathcal{AMT}$.

Matching using $\mathcal{AMT}$ language inclusion as in [32] has a limitation in the structure of the policy automaton (only deterministic automaton). The constraint arises from the $\mathcal{AMT}$ complementation. As BA complementation, the nondeterminic complementation is complicated and demonstrates exponential blow-up in the state space [8]. Safra in [39] gives a better lower bound ($2^{O(n \log n)}$) for nondeterministic BA complementation, however it is still exponential(see [49]).

The determinism constraint complies to our domain of interest because the security policies in our application domain are naturally deterministic, as the platform owner should have a clear idea on what to allow or disallow. Furthermore, to cope with non-deterministic $\mathcal{AMT}$, we can use the approach as in [33].

$\mathcal{AMT}$ makes it possible to match the mobile's policy and the midlet's contract by mapping the problem into a variant of the on-the-fly product and emptiness test from automata theory, without symbolic manipulation procedures of zones and regions nor finite representation of equivalence classes. The tractability limit is essentially the complexity of the satisfiability procedure for the theories, called as subroutines, where most practical policies require only polynomial time decision procedures.

A known problem with security automata and infinity yet to be addressed is the encoding of policies such as "we must allow certain strings that we have seen in the past". If the set of strings is unbounded, then it is difficult (if not impossible) to encode it with finite states. Finding a suitable approximation is the subject of current investigations.

In our current implementation of the matcher that runs on a mobile phone, security states of the automata are represented by variables over finite domains e.g. smsMessagesSent ranges between 0 to 5. [2, 6]. A possible solution could be to extend the work on finite-memory automata [29] by Kaminski and Francez or other works [36] that studied automata and logics on strings over infinite alphabets.

An approach to address scalability (if our smart-phone must cope with the webapplications of its internal web server) is to give up soundness of the matching and use algorithms for simulation and testing. A challenge to be addressed is how to measure the coverage of approximate matching. Which value should give a reasonable assurance about security? Should it be an absolute value? Should it be in proportion of the number of possible executions? In proportion to the likely executions? An interesting approach could be to recall to life a neglected section on model checking by Courcoubetis et al [10] in which they traded off a better performance of the algorithm in change for the possibility of erring with a small probability.

A second approach is to use the contract as a model of the application in order to generate security tests by applying techniques from Model Based Testing [50]. Losing soundness is a major disadvantage: an application may pass all the generated tests and still turn out to violate the contract once fielded. However, the advantages are also important: no annotations on the application source code are needed, and the tests generated from the contract can be easily injected in the standard platform testing phase, thus making this approach very practical. A challenge to be addressed here is how to measure the coverage of such security tests. When are there enough tests to give a reasonable assurance about security?

Another interesting problem for future work is a scenario when the claimed security

contract is missing (as is the case for current MIDP applications). In that case, based on the platform security policy, the "claimed" security contract could be inferred by static analysis as an approximation automaton. If such an approximation is matched, then monitoring the code becomes unnecessary. The feasibility of this approach depends on the cost of inferring approximation automata on-the-fly.

Acknowledgement

The EU-FP6-IST-STREP-S3MS project for partly supporting this research.

R. Sebastiani for his insightful comments in the beginning of our work.

References

[1] W. Ackermann. *Solvable Cases of the Decision Problem.* North Holland Pub. Co., 1954.

[2] I. Aktug and K. Naliuka. Conspec - a formal language for policy specification. In *Proc. of the 1st Int. Workshop on Run Time Enforcement for Mobile and Distributed Systems (REM 2007)*, Dresden, Germany, 2007.

[3] L. Bauer, J. Ligatti, and D. Walker. More enforceable security policies. In *Found. of Comp. Security*, 2002.

[4] L. Bauer, J. Ligatti, and D. Walker. Composing security policies with polymer. In *Proc. of the ACM SIGPLAN 2005 Conf. on Prog. Lang. Design and Implementation*, pages 305–314. ACM Press, 2005.

[5] N. Bielova, N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Matching in security-by-contract for mobile code. *J. of Logic and Algebraic Programming*, 78:340–358, May-June 2009.

[6] N. Bielova, M. Dalla Torre, N. Dragoni, and I. Siahaan. Matching policies with security claims of mobile applications. In *Proc. of the 3rd Int. Conf. on Availability, Reliability and Security (ARES'08)*. IEEE Press, 2008.

[7] M. Bozzano, R. Bruttomesso, A. Cimatti, T. Junttila, S. Ranise, P.v. Rossum, and R. Sebastiani. Efficient satisfiability modulo theories via delayed theory combination. In K. Etessami and S.K. Rajamani, editors, *Proc. of the 17th Int. Conf. on Computer Aided Verification (CAV'05)*, volume 3576 of *LNCS*, pages 335–349. Springer-Verlag, 2005.

[8] J.R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel et al., editor, *Int. Cong. on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

[9] B.V. Cherkassky and A.V. Goldberg. Negative-cycle detection algorithms. *Mathematical Programming*, 85(2):277–311, 1999.

[10] C. Courcoubetis, M.Y. Vardi, P. Wolper, and M. Yannakakis. Memory-efficient algorithms for the verification of temporal properties. *Formal Methods in Syst. Design*, 1(2-3):275–288, 1992.

[11] N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code. In *Proc. of the 4th European PKI Workshop Theory and Practice (EUROPKI'07)*, page 297. Springer-Verlag, 2007.

[12] U. Erlingsson. *The Inlined Reference Monitor Approach to Security Policy Enforcement*. PhD thesis, Department of Computer Science, Cornell University, 2004.

[13] U. Erlingsson and F.B. Schneider. IRM enforcement of Java stack inspection. In *Proc. of the 2000 IEEE Symp. on Security and Privacy*, pages 246–255, 2000.

[14] K. Etessami, T. Wilke, and R. Schuller. Fair simulation relations, parity games, and state space reduction for büchi automata. *SIAM J. on Comp.*, 34(5):1159–1175, 2005.

[15] M. Fitting. *First-order logic and automated theorem proving*. Springer-Verlag, 1996.

[16] P. Gastin, B. Moro, and M. Zeitoun. Minimization of counterexamples in SPIN. In *Proc. of the 11th Int. SPIN Workshop*, volume 2989 of *LNCS*, pages 92–108. Springer-Verlag, 2004.

[17] S. Ghilardi. Model-theoretic methods in combined constraint satisfiability. *J. of Autom. Reas.*, 33(3):221–249, 2004.

[18] S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Proc. of the 14th Int. Conf. on Computer Aided Verification (CAV'02)*, pages 610–624. Springer-Verlag, 2002.

[19] L.G. Hacijan. A polynomial algorithm in linear programming. In *Dokl. Akad. Nauk SSSR*, volume 244, pages 1093–1096, 1979.

[20] K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Program. Lang. Syst.*, 28(1):175–205, 2006.

[21] K.W. Hamlen, G. Morrisett, and F.B. Schneider. Certified in-lined reference monitoring on .net. In *Proc. of the 2006 workshop on Prog. Lang. and analysis for security*, pages 7–16. ACM Press, 2006.

[22] M. Hennessy and H. Lin. Symbolic bisimulations. In *MFPS'92: Selected papers of the meeting on Math. Foundations of Programming Semantics*, pages 353–389. Elsevier Sci. Publishers B. V., 1995.

[23] T.A. Henzinger, O. Kupferman, and S.K. Rajamani. Fair simulation. In *Proc. of of the 8th Int. Conf. on Concurrency Theory*, pages 273–287. ACM Press, 1997.

[24] T.A. Henzinger, R. Majumdar, and J.F. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1):1–32, 2005.

[25] G. J. Holzmann, D. Peled, and M. Yannakakis. On nested depth first search. In *Proc. of the 2nd Int. SPIN Workshop*, pages 23–32. American Mathematical Society, 1996.

[26] G.J. Holzmann. *The Spin Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, 2004.

[27] N.D. Jones. Space-bounded reducibility among combinatorial problems. *J. of Comp. and Syst. Sci.*, 11(1):68–85, 1975.

[28] M. Jurdzinski. Small progress measures for solving parity games. In *STACS '00: Proc. of the 17th Annual ACM Symposium on Theoretical Aspects of Computer Science*, pages 290–301. Springer-Verlag, 2000.

[29] M. Kaminski and N. Francez. Finite-memory automata. *Theor. al Comp. Sci.*, 134(2):329–363, 1994.

[30] S. Krstic, A. Goel, J. Grundy, and C. Tinelli. Combined satisfiability modulo parametric theories. In *Proc. of the 13th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424, page 602. Springer-Verlag, 2007.

[31] S.K. Lahiri and M. Musuvathi. An efficient decision procedure for UTVPI constraints. In *Proc. of the 5th Int. Workshop on Frontiers of Combining Systems (FroCoS'05)*, volume 3717. Springer-Verlag, 2005.

[32] F. Massacci and I. Siahaan. Matching midlet's security claims with a platform security policy using automata modulo theory. In *Proc. of the 12th Nordic Workshop on Secure IT Systems (NordSec'07)*, 2007.

[33] F. Massacci and I. Siahaan. Simulating midlet's security claims with automata modulo theory. In *Proc. of the 2008 workshop on Prog. Lang. and analysis for security*, pages 1–9, 2008.

[34] G.C. Necula. Proof-carrying code. In *Proc. of the 24th ACM SIGPLAN-SIGACT Symp. on Princ. of Prog. Lang.*, pages 106–119. ACM Press, 1997.

[35] G. Nelson and D.C. Oppen. Simplification by cooperating decision procedures. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 1(2):245–257, 1979.

[36] F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *TOCL*, 5(3):403–435, 2004.

[37] R. Nieuwenhuis, A. Oliveras, and C. Tinelli. Solving SAT and SAT Modulo Theories: from an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). *J. of the ACM*, 53(6):937–977, 2006.

[38] C.H. Papadimitriou. On the complexity of integer programming. *J. of the ACM*, 28(4):765–768, 1981.

[39] S. Safra. On the Complexity of omega-Automata. In *IEEE Symp. on Found. Comp. Science (FOCS'88)*, pages 319–327, White Plains, New York, USA, 1988. IEEE Press.

[40] F.B. Schneider. Enforceable security policies. *ACM Trans. on Inf. and Syst. Security*, 3(1):30–50, 2000.

[41] S. Schwoon and J. Esparza. A note on on-the-fly verification algorithms. Technical Report 2004/06, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, November 2004.

[42] R. Sebastiani. Lazy satisfiability modulo theories. *J. on Satisfiability, Boolean Modeling and Computation*, 3:141–224, 2007.

[43] R. Sekar, V.N. Venkatakrishnan, S. Basu, S. Bhatkar, and D.C. DuVarney. Model-carrying code: a practical approach for safe execution of untrusted applications. In *Proc. of the 19th ACM Symp. on Operating Syst. Princ.*, pages 15–28. ACM Press, 2003.

[44] C. Talhi, N. Tawbi, and M. Debbabi. Execution monitoring enforcement under memory-limitation constraints. *Inform. and Comp.*, 206(2-4):158–184, 2007.

[45] C. Tinelli and M.T. Harandi. A new correctness proof of the Nelson-Oppen combination procedure. *Proc. of the 1st Int. Workshop on Frontiers of Combining Systems (FroCoS'96)*, 3:103–120, 1996.

[46] C. Tinelli and C.G. Zarba. Combining nonstably infinite theories. *J. of Autom. Reas.*, 34(3):209–238, 2005.

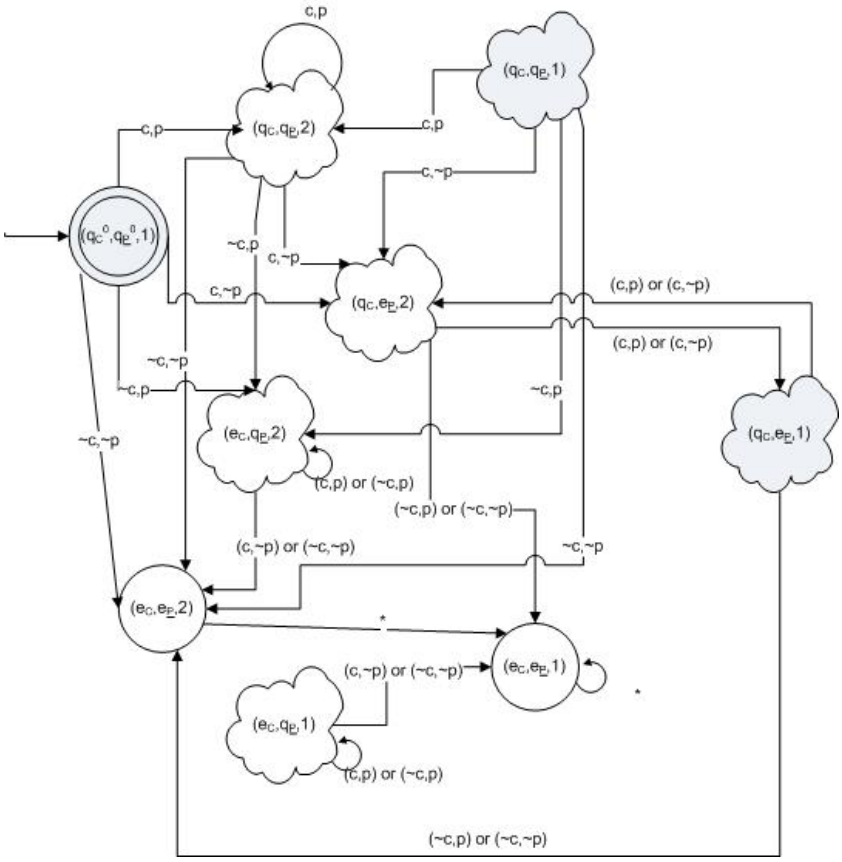
[47] D. Vanoverberghe, P. Philippaerts, L. Desmet, W. Joosen, F. Piessens, K. Naliuka, and F. Massacci. A flexible security architecture to support third-party applications on mobile devices. In *Proc. of the 1st ACM Comp. Sec. Arch. Workshop*, 2007.

[48] M. Vardi. An automata-theoretic approach to linear temporal logic. In *Proc. of the 8th Banff Higher order workshop conference on Logics for concurrency : structure versus automata*, LNCS, pages 238–266. Springer-Verlag, 1996.

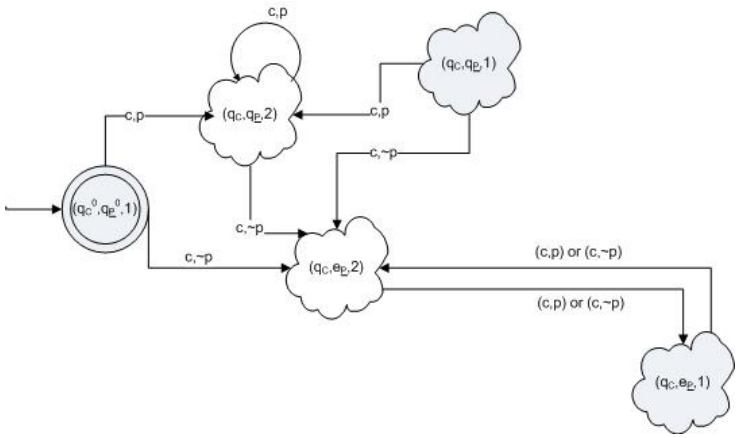
[49] M.Y. Vardi. Büchi complementation a 40-year saga, March 2006.

[50] M. Veanes, C. Campbell, W. Schulte, and N. Tillmann. Online testing with model programs. In *Proc. of the 10th Eur. Softw. Eng. Conf. held jointly with 13th ACM SIGSOFT Int. Symp. on Found. of Softw. Eng.*, pages 273–282. ACM Press, 2005.

[51] B.S. Yee. A sanctuary for mobile agents. In J. Vitek and C.D. Jensen, editors, *Secure Internet Programming*, pages 261–273. Springer-Verlag, 1999.



(a) Automata Intersection without Optimization



(b) Automata Intersection with Optimization

$c \doteq$ valid contract transition

$\neg c \doteq$ invalid contract transition

$\neg p \doteq$ invalid policy transition

shaded areas are accepting states

(c) Abbreviations

Figure 6: Automata Intersection

Algorithm 1 check_safety(s) Procedure

Input: state s;

```
1: map(s) := in_current_path;
2: for all ((s, e, t) ∈ Δ) do
3:    if (DecisionProcedure(e) = SAT) then
4:       if (map(t) = in_current_path ∧ ((s ∈ F) ∨ (t ∈ F))) then
5:          report non-empty;
6:       else if (map(t) = safe) then
7:          check_safety(t);
8: if (s ∈ F) then
9:    check_availability(s);
10:   map(s) := availability_checked;
11: else
12:   map(s) := safety_checked;
```

Algorithm 2 check_availability(s) Procedure

Input: state s;

```
1: for all ((s, e, t) ∈ Δ) do
2:    if (DecisionProcedure(e) = SAT) then
3:       if (map(t) = in_current_path) then
4:          report non-empty;
5:       else if (map(t) = safety_checked) then
6:          map(t) := availability_checked
7:          check_availability(t);
```

Algorithm 3 check_safety$(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x)$ Procedure

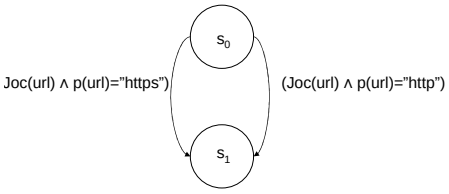
Input: state $s^{\mathbf{C}}$, state $s^{\overline{\mathbf{P}}}$, marker x;

1: $map(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x) :=$ `in_current_path`;
2: **for all** $((s^{\mathbf{C}}, e^{\mathbf{C}}, t^{\mathbf{C}}) \in \Delta^{\mathbf{C}})$ **do**
3: **for all** $((s^{\overline{\mathbf{P}}}, e^{\overline{\mathbf{P}}}, t^{\overline{\mathbf{P}}}) \in \Delta^{\overline{\mathbf{P}}})$ **do**
4: **if** $(DecisionProcedure(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}}) = SAT)$ **then**
5: $y := condition(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x, S^{\mathbf{C}}, S^{\overline{\mathbf{P}}})$
6: **if** $(map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) =$ `in_current_path` $\wedge ((\mathtt{s}^{\mathbf{C}} \in \mathtt{S}^{\mathbf{C}} \wedge \mathtt{s}^{\overline{\mathbf{P}}} = \mathtt{err}^{\overline{\mathbf{P}}} \wedge \mathtt{x} = 1) \vee (\mathtt{t}^{\mathbf{C}} \in \mathtt{S}^{\mathbf{C}} \wedge \mathtt{t}^{\overline{\mathbf{P}}} = \mathtt{err}^{\overline{\mathbf{P}}} \wedge \mathtt{y} = 1)))$ **then**
7: report policy violation;
8: **else if** $(map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) =$ `in_current_path` $\wedge ((\mathtt{s}^{\mathbf{C}} \in \mathtt{S}^{\mathbf{C}} \wedge \mathtt{s}^{\overline{\mathbf{P}}} \in (\mathtt{S}^{\overline{\mathbf{P}}} \backslash \{\mathtt{err}^{\overline{\mathbf{P}}}\}) \wedge \mathtt{x} = 1) \vee (\mathtt{t}^{\mathbf{C}} \in \mathtt{S}^{\mathbf{C}} \wedge \mathtt{t}^{\overline{\mathbf{P}}} \in (\mathtt{S}^{\overline{\mathbf{P}}} \backslash \{\mathtt{err}^{\overline{\mathbf{P}}}\}) \wedge \mathtt{y} = 1)))$ **then**
9: report availability violation;
10: **else if** $(map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) =$ `safe`$)$ **then**
11: **check_safety**$(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y)$;
12: **if** $(s^{\mathbf{C}} \in S^{\mathbf{C}} \wedge s^{\overline{\mathbf{P}}} \in S^{\overline{\mathbf{P}}} \wedge x = 1)$ **then**
13: **check_availability**$(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x)$;
14: $map(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x) :=$ `availability_checked`;
15: **else**
16: $map(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x) :=$ `safety_checked`;

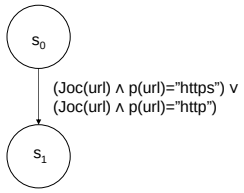
Algorithm 4 check_availability$(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x)$ Procedure

Input: state $s^{\mathbf{C}}$, state $s^{\overline{\mathbf{P}}}$, marker x;

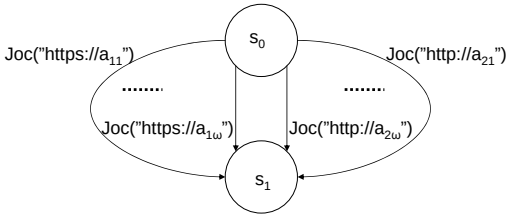
1: **for all** $((s^{\mathbf{C}}, e^{\mathbf{C}}, t^{\mathbf{C}}) \in \Delta^{\mathbf{C}})$ **do**
2: **for all** $((s^{\overline{\mathbf{P}}}, e^{\overline{\mathbf{P}}}, t^{\overline{\mathbf{P}}}) \in \Delta^{\overline{\mathbf{P}}})$ **do**
3: **if** $(DecisionProcedure(e^{\mathbf{C}} \wedge e^{\overline{\mathbf{P}}}) = SAT)$ **then**
4: $y := condition(s^{\mathbf{C}}, s^{\overline{\mathbf{P}}}, x, S^{\mathbf{C}}, S^{\overline{\mathbf{P}}})$
5: **if** $(map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) =$ `in_current_path`$)$ **then**
6: **if** $(t^{\overline{\mathbf{P}}} = err^{\overline{\mathbf{P}}})$ **then**
7: report policy violation;
8: **else**
9: report availability violation;
10: **else if** $(map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) =$ `safety_checked`$)$ **then**
11: $map(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y) :=$ `availability_checked`
12: **check_availability**$(t^{\mathbf{C}}, t^{\overline{\mathbf{P}}}, y)$;



(a) Splitting Edges



(b) Disjuncting Expressions

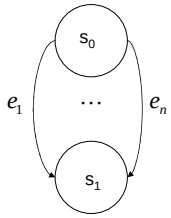


(c) Concrete Automaton

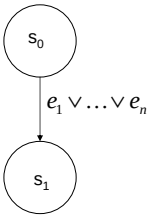
$$
\begin{aligned}
e_{11} &\doteq (Joc(url) \wedge p(url) = \text{“https”}) \\
e_{12} &\doteq (Joc(url) \wedge p(url) = \text{“http”}) \\
e_2 &\doteq (Joc(url) \wedge p(url) = \text{“https”}) \\
&\quad \vee (Joc(url) \wedge p(url) = \text{“http”})
\end{aligned}
$$

(d) Abbreviations for expressions

Figure 7: Symbolic vs Concrete Automaton



(a) Automaton before normalization



(b) Automaton after normalization

Figure 8: Normalization of an automaton

Algorithm 5 Simulation Algorithm

Input: two $\mathcal{AMT}$ automata Aut^C and Aut^P

Construct compliance game graph $G = \langle V_1, V_0, E, l \rangle$
for all $v \in V$ **do**
    $\mu(v) := \mu_{\mathtt{new}}(v) := 0$
repeat
    $\mu := \mu_{\mathtt{new}}$
    for all $v \in V_0$ **do**
        $\mu_{\mathtt{new}}(v) := \begin{cases} \infty & \text{if} \quad \{\mu(w)|(v,w)\} = \emptyset \\ min\,\{\mu(w)|(v,w)\} & \text{otherwise} \end{cases}$
    for all $v \in V_1$ **do**
        $max_v := max\,\{\mu(w)|(v,w) \in E\}$
        $\mu_{\mathtt{new}}(v) := \begin{cases} \infty & \text{if} \quad max_v = \infty \\ 0 & \text{if} \quad l(v) = 0 \\ max_v + 1 & \text{if} \quad l(v) = 1 \\ max_v & \text{if} \quad l(v) = 2 \end{cases}$
until $\mu = \mu_{\mathtt{new}}$
if $\mu(v_{(\mathbf{s_0}^C, \mathbf{s_0}^P)}) < \infty$ **then**
    Simulation exists