Simulating Midlet's Security Claims with Automata Modulo Theory

Fabio Massacci Ida S.R. Siahaan

Department of Information Engineering and Computer Science (DISI), University of Trento - Italy

name.surname@disi.unitn.it

Abstract

Model-carrying code and security-by-contract have proposed to augment mobile code with a claim on its security behavior that could be matched against a mobile platform policy before downloading the code. 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.

Our objective is the run-time matching of the mobile's platform policy against the midlet's security claims expressed as $\mathcal{AMT}$. To this extent the use of on-the-fly product and emptiness test from automata theory may not be effective. In this paper we present an algorithm extending fair simulation between Büchi automata that can be more efficiently implemented.

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory; I.1.2 [*Symbolic and Algebraic Manipulation*]: Algorithms

General Terms Security, Theory, Verification

Keywords automata, security policy, mobile code

1. Introduction

Pervasive services (Bacon 2002) envisions a nomadic user traversing a variety of environments and seamlessly and constantly receiving services from other portables, handhelds, embedded or wearable computers. The nomadic user does not only invoke services in push or pull mode but also download *new* applications that are locally available. These *pervasive client downloads* will appear because service providers will try to exploit the computational power of the nomadic devices to make a better use of the services available in the environment (Dragoni et al. 2007b).

Managing security of services in this scenario is a major challenge as the current security model adopted for mobile phones (the JAVA MIDP 2.0) is the exact negation of this business idea: mobile code is run if its origin is trusted (i.e. digitally signed by a trusted party). The level of trust of determines the privileges of the code and untrusted code is forbidden to have any interaction with the environment.

PLAS'08, June 8, 2008, Tucson, Arizona, USA.

Even if we accept the signature, we still have another problem: there is no semantics attached to the signature. This is a problem for both code producers and consumers. From the point of view of mobile code consumers they must essentially accept the code as-is without the possibility of making informed decisions, while from code producer they produce code with unbounded liability. They cannot declare which security actions the code will do, because by signing the code they essentially declare that they did it. Consequently, injecting an application in the mobile market is a time consuming operation as developers must convince the operators that their code is not harmful.

We can apply a *security reference monitor* which observes execution of a target system and halts that system whenever it is about to violate some security policy of concern (Schneider et al. 2001; Erlingsson 2003). While security monitors remains the bottom-line action, we could be more effective if we start asking some questions about the code.

The first question is whether the code satisfies some pre-defined policy. The Bytecode verifier in Java does exactly this first preliminary check. More advanced techniques based on Proof-Carrying Code (Necula and Lee 1996; Necula 1997) extend the scope of what can be actually checked. One of the limitation of the approaches based on language-based security is that the policy is tied to the programming language, therefore it is difficult to customize the policy on a per-user base.

We need to lift the question to a more flexible one: does the code satisfy a user-defined policy? In general case this is equivalent to arbitrary software verification which is not practical for pervasive downloads. However the idea behind model-carrying code (Sekar et al. 2003) and security by contract (Dragoni et al. 2007a) is that code should come accompanied with a "digest" (a security model or a security contract) that represents its essential security behavior. Then one only needs to check the latter against the user predefined security policies.

The interesting problem which is the focus of our research it matching the security claims of the code with the security desires of the platform. Matching can be done off-line (e.g. a developer checking its claims on a variety of Vodafone's default policies) or on-line (e.g. a user who downloads a midlet and runs it).

In this paper we build on the concept of *Automata Modulo Theory* ($\mathcal{AMT}$) proposed in (Massacci and Siahaan. 2007). $\mathcal{AMT}$ generalize the finite state automata of model-carrying code (Sekar et al. 2003) and extends Büchi Automata (BA). It is suitable for formalizing systems with finitely many states but infinitely many transitions by leveraging on 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* as in (Dragoni et al. 2007a) and model-carrying code (Venkatakrishnan et al. 2002) by capturing

the infinite transition into finite transitions labeled as expressions in defined theories.

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. In (Massacci and Siahaan. 2007) we showed 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 (Holzmann 2004) with oracle queries to a decision procedure solver[1].

This 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 (Vanoverberghe et al. 2007) and the complemented representation that we use for matching.

1.1 The contribution of this paper

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

In the next section we present an overview of security-by-contract framework providing a description of the overall life-cycle of mobile code in this setting and we also describe mobile applications security requirements and contract specification as motivations for $\mathcal{AMT}$. Then, we introduce $\mathcal{AMT}$ and the corresponding automata operation (§3) and some specific issues to be considered in $\mathcal{AMT}$. We describe simulation, symbolic simulation and fair simulation for $\mathcal{AMT}$ (§4). Finally, we present algorithm for lifting finite state tools to $\mathcal{AMT}$ simulation (§5).

2. Intuitions and Motivations

To understand better the motivation behind this work we consider how a midlet-life cycle would be in the security-by-contract (SxC for short) paradigm (Dragoni et al. 2007a).

After, or better during the application development, the mobile code developers are responsible to provide a description of the security behavior that their code finally provides. Such a code can then undergo a formal certification process which can be done by the developer's own company, the mobile operator or any other third party for which the application has been developed. By using suitable techniques such as static analysis or monitor in-lining or proof carrying code the code is certified to comply with the developer's contract. Subsequently the code and the security claims are sealed together with a digital signature and shipped for deployment as shown on Fig.1.

EXAMPLE 2.1. *The Personal Information Management (PIM) system on the 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 was opened no connections are allowed". This contract permits executing* `Connector.open()` *method only if* `PIM.openPIMList()` *method was never called before. This is only a toy example to illustrate a security policy.*

[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 (Henzinger et al. 2005) which would not be suitable for our intended application i.e. checking security claims before a pervasive download on a mobile phone.

REMARK 2.1. *We use the word* policy *for a platform security policy. We use the word* contract *for security claims made by a midlet.*

At deployment time the target platform will follow the workflow that we have sketched in Fig.2 (see also (Vanoverberghe et al. 2007). At first it checks that the evidence is correct. Such evidence could be a trusted signature as in standard mobile applications (Yee 1999). An alternative evidence could be a proof that the code satisfies the contract (and then one could use PCC techniques to check it (Necula 1997)).

Once we have evidence that the contract is trustworthy the platform will check that the claimed policy is actually compliant with the policy that our platform would like to be enforced. If this is the case, then the application can be run without further ado Fig.2. This might be a significant saving from in-lining a security monitor.

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* `Connector.open(string url)` *method only if the started connection is a secure one i.e.* `url` *starts with "https://".*

Matching should succeed 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 into the code 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.

Typically the policy will cover 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 rule for sessions regarding PIM and connection is shown in Ex. 2.1, it could be one of the rules of a contract. Another example is a rule for methods invocation of a Java object as shown in Ex. 2.2. This example can be one of the rules of a policy. Both examples describe safety properties, which are the common properties we want to verify.

Although most properties are safety properties, liveness properties also exist as shown in Ex. 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.*

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 (Erlingsson 2003, p.59) and Sekar et al. (Sekar et al. 2003). Then the operation of matching the midlet's claim with platform policy can be mapped into classical problems in automata theory.

One possible alternative 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 is 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

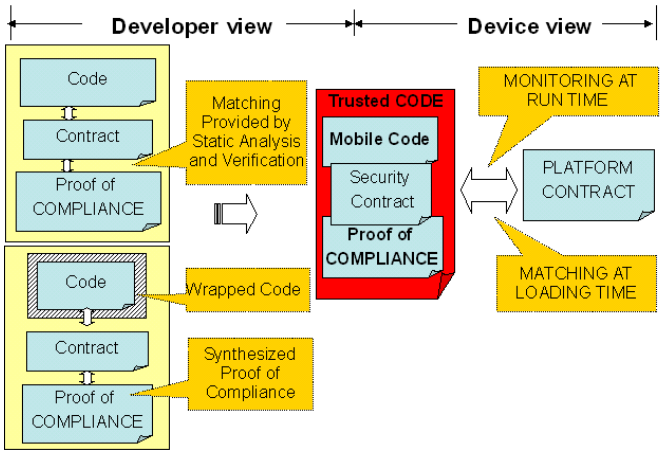


Figure 1. Mobile Code Components with Security-by-Contract

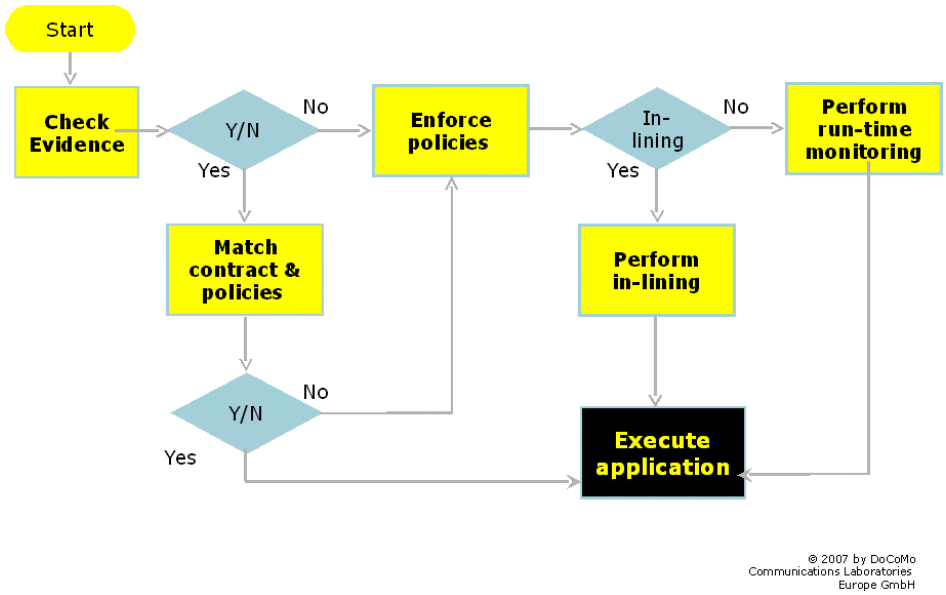


Figure 2. SxC Workflow

corresponds to a security violation. We have pursued this avenue in (Massacci and Siahaan. 2007).

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 behavior of Aut^P. Simulation is usually 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.

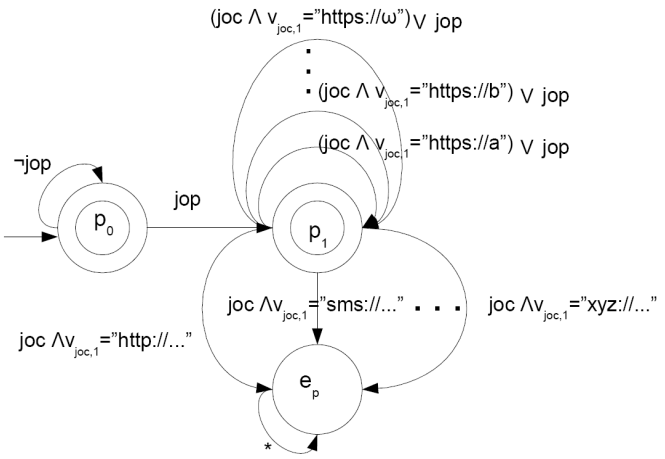
While this idea of the security-digest is almost a decade old (Sekar et al. 2003; Erlingsson 2003) the practical realization has been hindered by a significant technical hurdle: we cannot use the naive encoding into automata for practical policies. Even the basic policies in Ex. 2.1 and Ex. 2.2 will lead to automata with infinitely many transitions.

Fig.3(a) represents an automaton for Ex. 2.2. Starting from state p_0, we stay in this state while PIM is not accessed (jop). As PIM is accessed we move to state p_1 and we stay in state p_1 only if the started connection `Connector.open(string url)` method is a secure one i.e. `url` starts with "https://" or we keep accessing PIM (jop). We enter state e_p if we start an unsecure connection `Connector.open(string url)` e.g. `url` starts with "http://"or "sms://" etc. These examples are from a Java VM. Since we do not consider useful to invent our own names for API calls we use the `javax.microedition` APIs (though a bit verbose) for the notation that is shown in Fig.3(b).

3. Automata Modulo Theory

The theory of *Automata Modulo Theory* ($\mathcal{AMT}$ for short) is a combination of the formal notion of Büchi Automata (BA) with the notion of Satisfiability Modulo Theories (SMT).

The intuition is that we represent a security policy as BA automaton where edges are not labeled by atomic actions but rather by expressions in a suitable theory. We prefer to use BA, rather than classical security automata, as there are some liveness properties which have to be verified, e.g. Ex 2.3.



(a) Infinite Transitions Security Policies

$$
\begin{aligned}
joc(v_{joc,1}) &\doteq \texttt{io.Connector.open}(url)\\
jop &\doteq \texttt{pim.PIM.openPIMList(...)}\\
q &\doteq \texttt{io.Connector.}type\\
&\quad \text{is protocol type e.g. “http”}\\
pr(q) = type &\doteq \text{permission } q\text{is for protocol } type\\
p(url) = type &\doteq url\texttt{.startsWith}(type)
\end{aligned}
$$

(b) Abbreviations for Java APIs

Figure 3. Infinite Transitions Example

The real scientific trick is the use of satisfiability modulo theory for reasoning about allowed APIs.

EXAMPLE 3.1. *When comparing a policy asking that* `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 enough that we deal with* $protocol$ *and* $port$ *as uninterpreted functions and apply the theory of equality and uninterpreted functions* $\mathcal{EUF}$*.*

Return to our examples Ex. 2.1 and Ex. 2.2. Figure4(b) shows an automaton modulo theory corresponding to policy of Ex. 2.2 and the automaton with infinitely many transitions from Fig.3(a). Fig. 4(a) corresponds to the contract from Ex. 2.1. The notation is the same from Fig.3(b).

EXAMPLE 3.2. *We can use the quantifier-free fragment of Linear Arithmetic over the integers* $\mathcal{LA}(\mathbb{Z})$ *when the actions of the policy or the contract sets limits on resources such as downloading a file of at most 50KB as opposed to to 100KB.*

Some theories of interest are the theory of difference logic $\mathcal{DL}$ the theory $\mathcal{EUF}$ of equality and uninterpreted functions, $\mathcal{LA}(\mathbb{Q})$ and the integers $\mathcal{LA}(\mathbb{Z})$. As in (Bozzano et al. 2005) we are particularly interested in the combination of two or more simpler theories. While this is a not complete list, our only requirement for a theory $\mathcal{T}$ is that the $\mathcal{T}$-satisfiability of conjunctions of ground literals is decidable by a $\mathcal{T}$-solver (Nieuwenhuis et al. 2006).

We assume the usual notion of signature Σ with variables $V = \{x, y, z, v, ...\}$, function symbols $\mathcal{F} = \{c, d, f, g, ...\}$ and predicate symbols $\mathcal{P} = \{p, q, ...\}$. Terms and formulas are defined in the usual way over the boolean connectives $\neg, \vee, \wedge$. A first-order Σ-structure $\mathcal{A}$ consists of a set A of elements as *domain*, a mapping of each n-ary function symbol $f \in \Sigma$ to a total function $f^{\mathcal{A}} : A^n \to A$, a mapping of each n-ary predicate symbol $p \in \Sigma$ to a relation $p^{\mathcal{A}} \subseteq A^n$.

Let $\mathcal{A}$ denote a structure, ϕ a formula, and $\mathcal{T}$a theory, all of signature Σ. We use the notation $(\mathcal{A}, \alpha) \models \phi$ when ϕ evaluates to true in $\mathcal{A}$ under the variable assignment $\alpha : V \to A$. We say that ϕ is satisfiable in $\mathcal{A}$ if there exists some α such that $(\mathcal{A}, \alpha) \models \phi$. We denote by E as a set of formulas.

DEFINITION 3.1 (Automaton Modulo Theory). *A tuple* $A = \langle E, S, q_0, \Delta_{\mathcal{T}}, F\rangle$ *is an automaton modulo theory* $\mathcal{T}$ *where* E *is a set of formulas in the language of the theory* $\mathcal{T}$*,* S *is a finite set of states,* $q_0 \in S$ *is the initial state,* $\Delta_{\mathcal{T}} : S \times E \to 2^S$ *is labeled transition function, and* $F \subseteq S$ *is a set of accepting states.*

We say that $(s, e, t) \in \Delta_{\mathcal{T}}$ when $t \in \Delta_{\mathcal{T}}(s, e)$. The intuition is that variables represent parameters over invoked methods. For example a guard $x < 3$ where x is some external parameter of a Java method means that this edge will be taken each time the Java method is invoked with a value of x smaller than 3. This is different from traditional state variables in classical hybrid automata for state variable x where the ”same” guard means that after taking the transition x must be smaller than 3.

The runs of the system are the traces of actual values of invoked APIs, represented by assignments.

DEFINITION 3.2 ($\mathcal{AMT}$ concrete run). *Let* $A = \langle E, S, q_0, \Delta_{\mathcal{T}}, F\rangle$ *be an automaton modulo theory* $\mathcal{T}$*. A* concrete run modulo $\mathcal{T}$ *of* A *is a sequence of states alternating with assignments* $\gamma = \langle s_0\alpha_0 s_1\alpha_1 s_2\alpha_2, \ldots\rangle$*, such that:*

1. $s_0 = q_0$
2. *there exists expressions* $e_i \in E$ *such that* $s_{i+1} \in \Delta_{\mathcal{T}}(s_i, e_i)$ *and* $(\mathcal{A}, \alpha_i) \models e_i$ *holds for all* $i \in [0 \ldots |w|]$ *(resp.* $i \in N$*).*

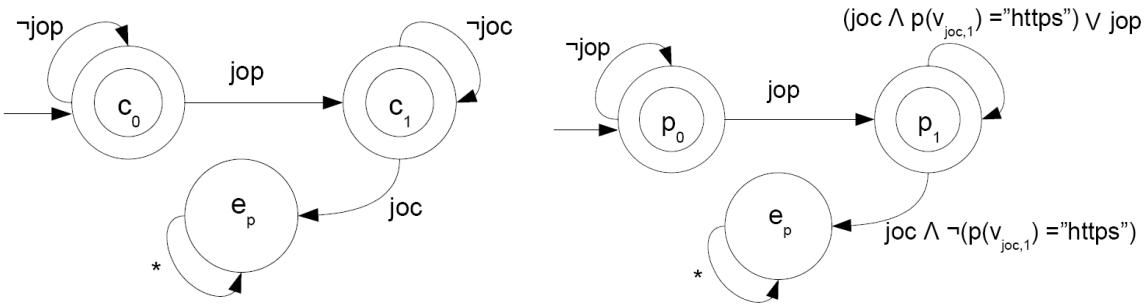
The trace associated with γ *is sequence of assignments* $w = \langle \alpha_0, \alpha_1, \alpha_2, \ldots\rangle$*. A finite run is accepting if* $s_{|w|}$ *goes through some accepting states. An infinite run is accepting if the automaton goes through some accepting states infinitely often as in BA.*

We use definition of run as in (Etessami et al. 2005) which is slightly different from the one we use in (Massacci and Siahaan. 2007), where we use only states, in order to accommodate simulation.

The notion of symbolic run is what would correspond to the traditional notion of run in automata.

DEFINITION 3.3 ($\mathcal{AMT}$ symbolic run). *Let* $A = \langle E, S, q_0, \Delta_{\mathcal{T}}, F\rangle$ *be an automaton modulo theory* $\mathcal{T}$*. A* symbolic run modulo $\mathcal{T}$ *of* A *is a sequence of states alternating with expressions* $\sigma = \langle s_0 e_0 s_1 e_1 s_2 e_2, \ldots\rangle$*, such that:*

1. $s_0 = q_0$
2. $\langle s_i, e_i, s_{i+1}\rangle \in \Delta_{\mathcal{T}}$ *and* $\mathcal{A}, \alpha_j \models e_i$ *holds for some* j



(a) $\mathcal{AMT}$ rule of a contract from Ex. 2.1

(b) $\mathcal{AMT}$ rule of a policy from Ex. 2.2

Figure 4. $\mathcal{AMT}$ Examples

The trace associated with σ is sequence of assignments $w = \langle e_0, e_1, e_2, \ldots \rangle$.

REMARK 3.1. *The condition that $\mathcal{A}, \alpha_j \models e_i$ holds for some j implies that every expression in the trace must be satisfiable and is necessary to guarantee that symbolic traces correspond to at at least one real, concrete execution.*

In order to understand better the semantics of an automaton modulo theory we can consider the corresponding concrete automaton which is constructed by replacing each transition labeled with an expression from the theory with the infinitely many transitions labeled by the corresponding satisfying assignments. Automata that are different at the theory level might have the same concrete representation.

For example the two automata modulo theory from Fig.5(a) have the same concrete model Fig.5(b).

Such equivalence is obvious because at the concrete level if the assignment α_{1i} is such that $(\mathcal{A}, \alpha_{1i}) \models joc \wedge protocol(url) =$ "$http''$ or $(\mathcal{A}, \alpha_{2i}) \models joc \wedge protocol(url) =$ "$https''$ then clearly $(\mathcal{A}, \alpha_i) \models joc \wedge (protocol(url) =$ "$http'' \vee protocol(url) =$ "$https''$). In other words, $\vee$ has the maximal model and thus in the transitions corresponding to the disjunction in the theory it is the union of all assignments in the concrete automaton.

4. Simulation

At first we introduce the notion of simulation at the concrete level, among assignments i.e. API calls and then we give the notion of symbolic simulation as in (Hennessy and Lin 1995). The actual notion of fair simulation is adapted from (Etessami et al. 2005; Gurumurthy et al. 2002; Henzinger et al. 1997).

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 4.1 (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 $\langle s_i, e^c_i, s_{i+1} \rangle \in \Delta^c_{\mathcal{T}}$ and an assignment α_i such that $(\mathcal{A}, \alpha_i) \models e^c_i)$ and moves to s_{i+1}.*
3. `Policy` *responds by a transition $\langle t_i, e^p_i, t_{i+1} \rangle \in \Delta^p_{\mathcal{T}}$ such that $(\mathcal{A}, \alpha_i) \models e^p_i)$ 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 $\vec{s} = \langle s_0 \alpha_0 s_1 \alpha_1 s_2 \alpha_2, \ldots \rangle$ and $\vec{t} = \langle t_0 \alpha_0 t_1 \alpha_1 t_2 \alpha_2, \ldots \rangle$ respectively of A^c and A^p. If $\vec{s} = \langle s_0 \alpha_0 s_1 \alpha_1 s_2 \alpha_2, \ldots \rangle$ is an accepting concrete run for A^c and $\vec{t} = \langle t_0 \alpha_0 t_1 \alpha_1 t_2 \alpha_2, \ldots \rangle$ 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: there is a move that the `Policy` could not do, i.e. that particular action is a violation.

EXAMPLE 4.1. *In a game between the* `Contract` *from Fig.4(a) and the* `Policy` *from Fig.4(b), the* `Contract` *can choose to invoke the url* `http://www.google.com` *and the* `Policy` *can respond by selecting the appropriate expression which is also satisfied by the same assignment.*

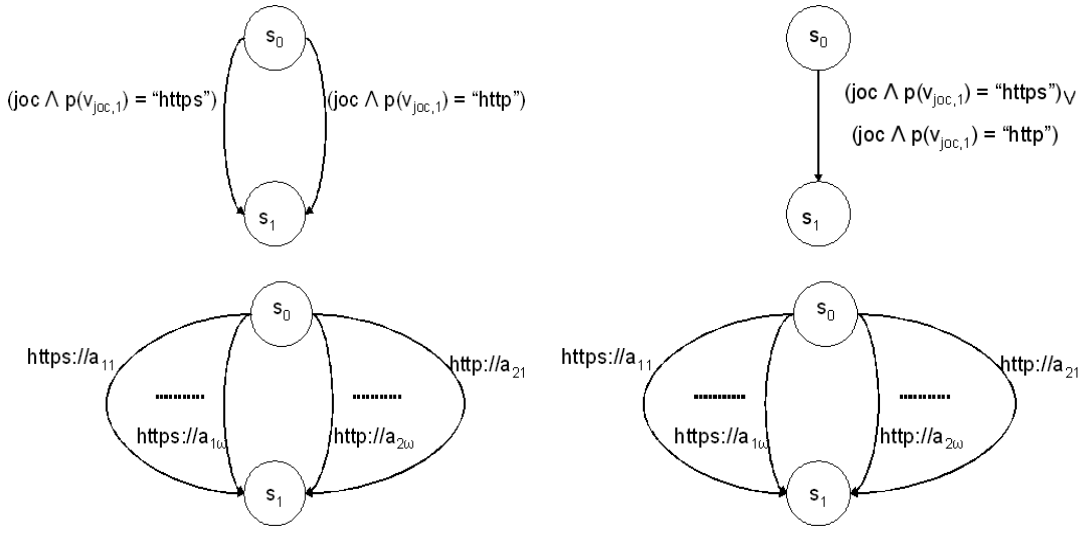
A more complex situation presents itself in the infinite case. Infinite runs correspond to liveness properties, e.g. something good happens infinitely often, for example Ex. 2.3. Therefore, the `Contract` only wins (i.e. it breaks the `Policy`) when according to its view of the world there are infinitely many good things but not for the `Policy` which after some initial good things is trapped in an endless sequence of unsatisfactory states.

EXAMPLE 4.2. *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 assignment, which is possible in the game if* `Policy` *has requested permission* `io.Connector.https` *in some previous steps.*

Now we can introduce the notion of *concrete strategy* for `Policy` in game $G^C_{A^c,A^p}(s_0, t_0)$ which is just a partial function which determines the next move of `Policy` given the history of the concrete game up to a certain point.

DEFINITION 4.2 (Concrete Strategy). *A partial function $f : S^c \times (S^p \times \alpha \times S^c)^* \to S^p$ is a* concrete strategy *if for any sequence $\langle s_0 \alpha_0 s_1 \alpha_1 \ldots s_i \alpha_i s_{i+1} \rangle$ which is a valid concrete run for A^c*

- *$f(s_0) = t_0$*
- *$f(\langle s_0 t_0 \alpha_0 s_1 \ldots s_i t_i \alpha_i s_{i+1} \rangle) = t_{i+1}$ such that $\langle t_i, e^p_i, t_{i+1} \rangle \in \Delta^p_{\mathcal{T}}$ and $(\mathcal{A}, \alpha_i) \models e^p_i$.*



(a) Splitting Edges (b) Disjuncting Expressions

Figure 5. Symbolic vs Concrete Automaton

A concrete strategy f of the game is a `Policy` *winning strategy* if and only if whenever a `Policy` select the moves of game defined in Definition 4.1 according to f then `Policy` wins the game.

DEFINITION 4.3 ($\mathcal{AMT}$ 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 4.4 ($\mathcal{AMT}$ 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}.*

The winner of the game is determined by the rules as in Definition 4.1 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 4.5 (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 $\langle s_0 e_0^c s_1 e_1^c \ldots s_i e_i^c s_{i+1}\rangle$ which is a valid symbolic run for A^c*

- $f(s_0) = t_0$
- $f(\langle s_0 t_0 e_0^c s_1 t_1 e_1^c \ldots s_i t_i e_i^c s_{i+1}\rangle) = t_{i+1}$ *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 defined in Definition 4.4 according to f then `Policy` wins the game.

DEFINITION 4.6 ($\mathcal{AMT}$ 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.*

THEOREM 4.1. *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. We sketch a proof of Prop. 4.1 by showing the correctness of our construction. In order to show the correctness of our construction, we first assume $A^c \leq A^p$ is an $\mathcal{AMT}$ fair simulation relation. By Definition 4.6 there is a winning strategy for A^p, such that whenever a `Policy` select the moves of game defined in Definition 4.4 according to strategy f then `Policy` wins the game. By Definition 4.4 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 assignments and by Definition 4.1 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 $\Delta_{\mathcal{T}}^p(t_j, e_j^p, t_{j+1})$ such that $e_j^c \rightarrow e_j^p$ is valid, meaning for all α_j, $(\mathcal{A}, \alpha_j) \models e_j^c \rightarrow e_j^p$ and by Definition 4.1 with $(\mathcal{A}, \alpha_j) \models e_j^p$), `Policy` can respond by a transition $\left\langle t_j, e_j^p, t_{j+1}\right\rangle \in \Delta_{\mathcal{T}}^p$. Furthermore, finitely many i such that $s_i \in F^c$ 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 $(\mathcal{A}, \alpha_i) \models e_i^c$) and by Definition 4.1 `Contract` can also move in concrete game.

The concrete strategy f' constructed is a winning strategy for A^p in concrete compliance game, hence by Definition 4.3 $A^c \sqsubseteq A^p$. □

In contrast to the simulation notions of (Hennessy and Lin 1995) the converse of Theorem 4.1 does not hold in general.

PROPOSITION 4.1. *$\mathcal{AMT}$ fair simulation is stronger than $\mathcal{AMT}$ language inclusion.*

The pair of automata in Figure5(b) and Fig.5(a) is a simple counter example. We can see this from concrete automata in Fig.5(a) and Fig.5(b) where both are the same thus having not just simulation but also bi-simulation. However, the $\mathcal{AMT}$ on Fig.5(a) cannot simulate the $\mathcal{AMT}$ on Fig.5(b). The second consequence of this maximal model is that in $\mathcal{AMT}$ simulation is stronger than language inclusion. For example if we have policy represented as Fig. 5(b) and contract represented as Fig. 5(a), where both automata accept the same language but according to simulation $VALID(e^1 \rightarrow e^2)$ does not hold, thus we do not have simulation. Technically this is a consequence of the maximal model for $\vee$.

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

DEFINITION 4.7 (Normalized $\mathcal{AMT}$). *$A = \langle E, S, q_0, \Delta_\mathcal{T}, F\rangle$ is a* normalized automaton *modulo theory $\mathcal{T}$ if and only if for every $q, q_1 \in S$ there is at most one expression $e_1 \in E$ such that $q_1 \in \Delta_\tau(q, e_1)$.*

For example Fig. 5(a) is a normalized automaton while Fig. 5(b) is not normalized.

We cannot always normalize an automaton by considering the disjunction of all expressions going to the same state as it may change nondeterministic automata into deterministic automata (see Fig. 6). However, if automata are in normalized form then we have the following theorem:

THEOREM 4.2. *For normalized $\mathcal{AMT}$ $A^c \leq A^p$ is a $\mathcal{AMT}$ fair simulation if and only if $A^c \sqsubseteq A^p$ is a concrete fair simulation.*

Proof. The first direction of Theorem 4.2 follows from Proof 4. We sketch proof for the second direction by first assuming $A^c \sqsubseteq A^p$ is a concrete fair simulation relation. We have to show that for every step in the game where `Policy` in concrete game moves according to a transition $\langle t_i, e_i^p, t_{i+1}\rangle \in \Delta_\mathcal{T}^p$ such that $(\mathcal{A}, \alpha_i) \models e_i^p)$ and moves to t_{i+1}, then in symbolic level $VALID(e^c \rightarrow e^p)$ holds. This follows as for normalized $\mathcal{AMT}$ by Definition4.7 there is at most one expression $e_1 \in E$ such that $q_1 \in \Delta_\tau(q, e_1)$, such that concrete level represents maximal model for $\vee$. □

5. A Decision Procedure for Run-Time Simulation

In order to check the compliance of the policy on the actual device as defined on Fig.2 a number of preliminary steps is necessary: we must check the digital signature on the device, compare the various security rules of the contract and the policy in order to identify the correct pair of $\mathcal{AMT}$ that need to be matched. We refer to (Dragoni et al. 2007a) for the overall algorithm and to (Massacci and Siahaan. 2007; Bielova et al. 2008) for the matching implementation using language inclusion.

However, the procedure for matching using language inclusion among $\mathcal{AMT}$ proposed in (Massacci and Siahaan. 2007) limited the generality of the policy in order to avoid the exponential blow-up that occurs when one complements a BA.

So here we propose a different algorithm that uses the concepts of fair simulation for matching and adapts the Jurdzinski's algorithm on parity games (Jurdzinski 2000).

DEFINITION 5.1 (Compliance Graph). *Let V_0 and V_1 be two disjoint sets, a* compliance graph *G is a tuple $\langle V_1, V_0, E, l\rangle$, where*

Algorithm 1 Simulation Algorithm

Require: two $\mathcal{AMT}$ automata (contract `Contract`, policy `Policy`)
1: Create compliance game graph $G = \langle V_1, V_0, E, l\rangle$
2: $\mu(v) := 0$ for all $v \in V$
3: **while** $\mu(v) \neq \mu_{\texttt{new}}(\mu, v)$ for some $v \in V$ **do**
4: $\quad \mu := \mu_{\texttt{new}}(\mu, v)$
5: **if** $\mu < \infty$ **then**
6: $\quad$ Simulation exists

- $V = V_0 \cup V_1$
- $E \subseteq V \times V$
- $l : V \rightarrow \{0, 1, 2\}$

where l is the compliance level of the game.

Intuitively the compliance level lv is 0 when `Policy` accepts, 1 when `Contract` accepts (but `Policy` have not accepted yet) and 2 when neither of them accepts.

A compliance game $P(G, v_0)$ on G starting at $v_0 \in V$ is played by two players `Policy` and `Contract`. 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.

We apply this compliance game to $\mathcal{AMT}$ such that given an $\mathcal{AMT}$ $\langle E, S^c, q_0^c, \Delta_\mathcal{T}^c, F^c\rangle$ and an $\mathcal{AMT}$ $\langle E, S^p, q_0^p, \Delta_\mathcal{T}^p, F^p\rangle$, we construct a $\langle V_1, V_0, E, l\rangle$ as follows:

- $V_1 = \{v_{(q^c, q^p, e^c, c)} | q^c \in S^c, q^p \in S^p, \exists q'^c.q^c \in \Delta_\mathcal{T}^c(q'^c, e^c)\}$
- $V_0 = \{v_{(q^c, q^p, e^p, p)} | q^c \in S^c, q^p \in S^p, \exists q'^p.q^p \in \Delta_\mathcal{T}^p(q'^p, e^p)\}$
- $E = \{(v_{(q^c, q^p, e^p, p)}, v_{(q^c, q'^p, e^c, c)}) | q'^p \in \Delta_\mathcal{T}^p(q^p, e^p) \wedge VALID(e^c \rightarrow e^p)\} \cup \{(v_{(q^c, q^p, e^c, c)}, v_{(q'^c, q^p, e^p, p)}) | q'^c \in \Delta_\mathcal{T}^c(q^c, e^c)\}$
- $$l(v) = \begin{cases} 0 & \text{if } v = v_{(q^c, q^p, e^c, c)} \text{ and } q^p \in F^p \\ 1 & \text{if } v = v_{(q^c, q^p, e^c, c)} \text{ and } q^c \in F^c \text{ and } q^p \notin F^p \\ 2 & \text{otherwise} \end{cases}$$

Next, we define a compliance value as $M_G = \{x | x \leq |V_1|\} \cup \{\infty\}$ and a *compliance measure* $\mu : V \rightarrow M_G$. We use ordering relation $>$ that depends on l of the current vertex v. If $l(v) = 1$ then $\mu(v) > \mu(w)$. If $l(v) = 2$ or $l(v) = 0$ then $\mu(v) \geq \mu(w)$.

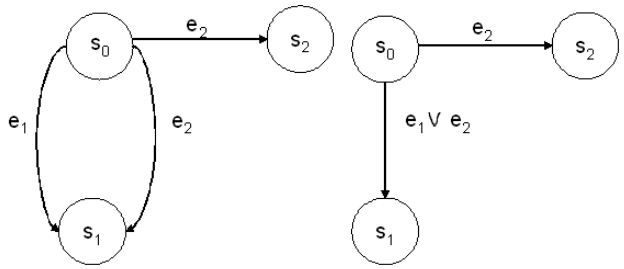
The update procedure $\mu_{\texttt{new}}(\mu, v)(u)$ is defined as follows:

- $\mu(u)$ if $u \neq v$
- $\mu(v)$ if $u = v$ and $l(v) = 0$
- $max\{\mu(v), max\{\mu(u) \oplus 1\}\}$ if $u = v$ and $l(v) = 1$
- $max\{\mu(v), min\{\mu(u)\}\}$ if $u = v$ and $l(v) = 2$

where $i \oplus 1 = i + 1$ if $i < |V_1|$ and ∞ otherwise.

We are now in the position to state our algorithmic results:

THEOREM 5.1. *Let the theory $\mathcal{T}$ be decidable with an oracle for the SMT problem in the complexity class $\mathcal{C}$ then Alg.1 decides fair simulation for $\mathcal{AMT}$ in time $POL - TIME^\mathcal{C}$ and in space $O(|S^c| \,.\, |S^p| \,.\, |\Delta_\mathcal{T}^p + \Delta_\mathcal{T}^c|)^\mathcal{C}$.*



(a) Nondeterministic Automaton (b) Deterministic Automaton after Normalization

Figure 6. Normalization changes Determinism of an Automaton

6. Conclusion and Related Work

Model-carrying code (Sekar et al. 2003) and security-by-contract (Dragoni et al. 2007a) proposed to augment mobile code with a claim on its security behavior that can be matched against a mobile platform policy before code downloading. In (Sekar et al. 2003) 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 ($\mathcal{AMT}$) (Massacci and Siahaan. 2007) enables systems formalization with finitely many states but infinitely many transitions. As we already showed in (Massacci and Siahaan. 2007), 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 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. The prototype for matching policies with security claims of mobile applications using $\mathcal{AMT}$ appeared in (Bielova et al. 2008).

Matching using language inclusion as in (Massacci and Siahaan. 2007) 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 nondeterminic complementation is complicated and exponentially blow-up in the state space (Büchi 1962). Safra in (Safra 1988), gives a better lower bound ($2^{O(n \log n)}$) for nondeterministic BA complementation, however it is still exponential(see (Vardi 2006)). This limitation does not evolve in matching using simulation as presented in this paper, because using simulation approach we can also deal with nondeterministic automata.

Infinite numbers of transitions in security policies by labeling each transition with a computable predicate instead of an atomic symbol has been studied in (Schneider 2000). 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 specified transitions as a function of the input symbols which can be the entire system state. However, $\mathcal{AMT}$ differs from security automata in transitions which are environmental parameters rather than system states.

Security monitors were implemented in several systems for example PoET/PSLang toolkit (Erlingsson and Schneider 2000), which can enforce security policies whose transitions pattern-match event symbols using regular expressions. Edit automata (Bauer et al. 2002) are another model for achieving this. Edit automata was implemented in the Polymer system (Bauer et al. 2005) to dynamically compose security automata. Most recently, the Mobile system (Hamlen et al. 2006) implements a linear decision algorithm that verifies that annotated .NET binaries satisfy a class of policies that includes security automata and edit automata. All mentioned approaches focus on the relations between code and security claims on the code (which we call contract), while $\mathcal{AMT}$ focuses between the security claims of the code and the platform desired security behavior. Other works fit into in-lining and run-time monitoring in our workflow while $\mathcal{AMT}$ falls into matching contract and policies.

A theory of symbolic bi-simulation for the π-calculus was proposed in (Hennessy and Lin 1995) 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 (Henzinger et al. 1997) 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 (Etessami et al. 2005) using parity game framework, based on solving parity games using small progress measures as appeared in (Jurdzinski 2000). An algorithm based on the notion of fair simulation was presented in (Gurumurthy et al. 2002) applied for the minimization of Büchi automata.

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 $\mathcal{AMT}$.

Our final objective is do the run-time matching of the mobile's platform policy against the midlet's security claims expressed as $\mathcal{AMT}$. We have already done this for the matching with language inclusion (we have a working .NET prototype working on a PDA HTC P3600 the algorithm extending fair simulation between Büchi automata that we have presented in the paper.

Acknowledgments

Research partly supported by the Projects EU-FP6-IST-STREP-S3MS, EU-FP6-IP-SENSORIA, and EU-FP7-IP-MASTER

References

J. Bacon. Toward pervasive computing. *IEEE Pervasive Comp. Magazine*, 1(2):84, 2002. ISSN 1536-1268.

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

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. ISBN 1-59593-056-6.

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.

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 CAV'05*, volume 3576 of *LNCS*, pages 335–349. Springer-Verlag, 2005.

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.

N. Dragoni, F. Massacci, K. Naliuka, and I. Siahaan. Security-by-Contract: Toward a Semantics for Digital Signatures on Mobile Code. In *Proc. of EuroPKI'07*. Springer-Verlag, 2007a.

N. Dragoni, F. Massacci, C. Schaefer, T. Walter, and E. Vetillard. A security-by-contracts architecture for pervasive services. In *Proc. of the 3rd Int. Workshop on Security, Privacy and Trust in Pervasive and Ubiquitous Computing*. IEEE Press, 2007b.

U. Erlingsson. *The Inlined Reference Monitor Approach to Security Policy Enforcement*. Technical report 2003-1916, Department of Computer Science, Cornell University, 2003.

U. Erlingsson and F.B. Schneider. Irm enforcement of java stack inspection. In *Proc. of Symp. on Sec. and Privacy*, page 246. IEEE Press, 2000. ISBN 0-7695-0665-8.

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. ISSN 0097-5397.

S. Gurumurthy, R. Bloem, and F. Somenzi. Fair simulation minimization. In *Proc. of CAV'02*, pages 610–624. Springer-Verlag, 2002. ISBN 3-540-43997-8.

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.

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.

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. Springer-Verlag, 1997.

T.A. Henzinger, R. Majumdar, and J.F. Raskin. A classification of symbolic transition systems. *ACM Trans. Comput. Logic*, 6(1):1–32, 2005. ISSN 1529-3785.

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

M. Jurdzinski. Small progress measures for solving parity games. In *STACS '00: Proc. of the 17th Annual Symposium on Theoretical Aspects of Computer Science*, pages 290–301. Springer-Verlag, 2000. ISBN 3-540-67141-2.

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.

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. ISBN 0-89791-853-3.

G.C. Necula and P. Lee. Safe kernel extensions without run-time checking. In *Proc. of the 7th USENIX symposium on Operating systems design and implementation*, pages 229–243. ACM Press, 1996. ISBN 1-880446-82-0.

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.

S. Safra. On the Complexity of omega-Automata. In *IEEE Symp. on Found. Comp. Science (FOCS'88)*, pages 319–327. IEEE Press, 1988.

F.B. Schneider. Enforceable security policies. *TISSEC*, 3(1):30–50, 2000.

F.B. Schneider, J.G. Morrisett, and R. Harper. A language-based approach to security. In *Informatics - 10 Years Back. 10 Years Ahead.*, pages 86–101. Springer-Verlag, 2001. ISBN 3-540-41635-8.

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 Sys. Princ.*, pages 15–28. ACM Press, 2003. ISBN 1-58113-757-5.

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.

M.Y. Vardi. Büchi complementation a 40-year saga. March 2006.

V. N. Venkatakrishnan, Ram Peri, and R. Sekar. Empowering mobile code using expressive security policies. In *Proc. of The 10th New Security Paradigms Workshop*, 2002.

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.