Dynamic Analysis of Usage Control Policies

Yehia Elrakaiby[1] and Jun Pang[2]
[1]*Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Kaiserslautern, Germany*
[2]*University of Luxembourg, 6 rue Richard Coudenhove-Kalergi, Luxembourg, Luxembourg*

Keywords: Usage Control, Access Control, Policy Analysis, Formal Verification.

Abstract: Usage control extends access control by enabling the specification of requirements that should be satisfied before, while and after access. To ensure that the deployment of usage control policies in target domains achieves the required security goals, policy verification and analysis tools are needed. In this paper, we present an approach for the dynamic analysis of usage control policies using formal descriptions of target domains and their usage control policies. Our approach provides usage control management explicit labeled transition system semantics and enables the automated verification of usage control policies using model checking.

1 INTRODUCTION

Access control aims at protecting information systems against unauthorized accesses. Usage control extends access control with the support of pre-access, ongoing-access and post-access controls, enabling the expression of more fine-grained controls on access operations. More specifically, usage controls allows specification of before-access, while-access and after-access requirements, typically in the form of either required user *actions*, e.g. acceptance of an application's usage terms prior to access, or some *state conditions* that should be maintained, e.g. to keep an ad window open during access.

A security policy formally specifies rules needed to satisfy the security requirements of a given information system. Therefore, the overall system security depends on its correct expression. Indeed, misspecified policies could allow undesirable security vulnerabilities and compromise the overall goals that security policies aim to achieve.

Two main analysis types are used for verification of security policies. Static analysis typically checks policy structure to verify policy consistency (Lupu and Sloman, 1999; Simon and Zurko, 1997). Dynamic policy analysis takes into account policy state evolution, i.e. the possible policy states that can be reached after, for example, the execution of policy administration actions (Li and Tripunitara, 2006), access requests (Becker and Nanz, 2007), or general actions (Craven et al., 2009). Dynamic analysis thus checks the satisfaction of the desired policy properties in every possible (reachable) state. Dynamic policy analysis has been used for the verification of access control policies (Li and Tripunitara, 2006; Becker and Nanz, 2007), and obligation policies (Craven et al., 2009). However, the dynamic analysis of usage control policies remains, to the best of our knowledge, unexplored. An overview of formal security policy analysis is presented in Section 6.

This paper introduces an approach for the dynamic analysis of usage control policies that consists of first (1) making a formal description of the target domain and its usage policy using the action language $C+$ (Giunchiglia et al., 2004), a recent member of the family of action languages (Gelfond and Lifschitz, 1998), and then (2) using model checking techniques to verify the properties of the specified usage control policy. Our work also formalizes usage control management, e.g. activation and cancellation of usage controls and the management of usage sessions, clarifying the enforcement of usage controls and enabling the use of standard model-checking techniques for the verification of usage control policy properties.

2 ACTION LANGUAGE $C+$

The action language $C+$ (Sergot, 2004) is a member of the family of action languages (Gelfond and Lifschitz, 1998). $C+$ has explicit transition system semantics and allows a natural expression of many things such as inertia, concurrent actions, non-determinism and ramifications. The practicality of

Elrakaiby Y. and Pang J..
Dynamic Analysis of Usage Control Policies.
DOI: 10.5220/0005046600880100
In *Proceedings of the 11th International Conference on Security and Cryptography* (SECRYPT-2014), pages 88-100
ISBN: 978-989-758-045-1

$C+$ is demonstrated by its use in diverse application domains (Armando et al., 2009; Armando et al., 2012; Artikis et al., 2007; Son et al., 2012; Dworschak et al., 2008; Artikis and Sergot, 2010). Several planning and model checking tools for the automated analysis of $C+$ specifications are available (CCa, ; iCC, ; Lifschitz, 1999; Gebser et al., 2010; Casolary, 2011; Babb and Lee, 2013).

For the description of dynamic domains, $C+$ provides two disjoint non-empty sets of atoms, namely a set of *simple actions* ($\mathcal{A}$) and a set of *fluents* ($\mathcal{F}$). Intuitively, fluents are propositions used to describe the state of the world. Actions are used to represent events that change the state of the world.

2.1 Basic Definitions

A *multi-valued propositional signature* is a set σ of symbols called *constants*, where each constant $c \in \sigma$ is associated with a non-empty set of symbols, disjoint from σ, called the *domain* of c. An *atom* of signature σ is an expression of the form $c = u$, where $c \in \sigma$ and $u \in dom(c)$. For instance, a Boolean constant is one whose domain is the set of truth values $\{t, f\}$. When c is a Boolean constant, $c = t$ is simply written as c and $c = f$ is written $\neg c$. A *formula* ϕ of signature σ is any propositional combination of atoms of σ. An *interpretation* I of σ is a function that maps every constant in σ to an element of its domain. An interpretation I *satisfies* an atom $c = u$ if $I(c) = u$. The satisfaction relation is extended from atoms to formulas, according to the standard truth tables for the propositional connectives. A *model* of a set X of formulas of signature σ is an interpretation of σ that satisfies all formulas in X. If every model of a set X of formulas satisfies some formula ϕ then X *entails* ϕ, written as $X \models \phi$.

2.2 $C+$ Syntax

An *action signature* (σ^f, σ^a) is partitioned into a non-empty set σ^f of *fluent* constants and another non-empty set σ^a of action constants. Fluents are further divided into *statically determined* fluents (σ^{st}) and *simple* fluents (σ^{si}). Fluent constants are symbols characterizing a *state*, whereas action constants characterize *state transitions*.

An *action description* D is a non-empty set of *causal laws*. Causal laws of an action description define a transition system by specifying relationships between fluents and actions. A causal law is a proposition of one of the following forms:

`static law:`	CAUSED F IF G
`dynamic law:`	CAUSED F IF G AFTER H
`action dynamic law:`	CAUSED α IF H

where F and G are formulas of σ^f such that F does not include any statically determined fluent constants, α is an action formula and H is any combination of fluent and action constants. A *static law* specifies that if G is true in a state, then F is caused to be true. Thus, static laws describe relationships between fluent constants in an individual state. A *dynamic law* describes relationships between actions and fluents in state transitions: in a transition from a state s_i to another state s_{i+1}, constants in F and G of a dynamic law are evaluated on s_{i+1}, fluent constants in H are evaluated on s_i and action constants in H are evaluated on the transition itself. An action dynamic law specifies that when G is true in a state, then when the system evolves from that state, it must do so in a way which makes α true, namely α is evaluated on the transition. The formula F of a causal law is called its *head*. In the following, the keyword CAUSED will not be used for readability.

Several common abbreviations of $C+$ propositions are used to model action preconditions, action non-determinism, fluents with default values, inertial and exogenous fluents. Abbreviations of $C+$ propositions used in this paper are shown in Table 1. Note that the *if* part of a rule may be omitted if it is *true*. Also, similarly to fluents in Table 1, action constants can be exogenous. An exogenous action may occur at any state.

Definite action descriptions are the most practical ones since it is straightforward to find runs through transition systems defined by them. An action description D is definite if:

1. the head of every causal law of D is an atom or $\bot$;
2. no atom is the head of infinitely many causal laws.

2.3 $C+$ Semantics

A $C+$ action description D defines a labeled transition system $TS = \langle S, L, \rightarrow\rangle$, where S is a set of states, L is a set of labels and $\rightarrow \subseteq S \times L \times S$ is a transition relation between states. The set of states and the labels of TS are interpretations of σ^f and σ^a, respectively. The states and transitions of TS are constrained by the causal laws in D in the following way:

- A state s of TS is an interpretation of σ^f iff s satisfies the set of formulas $T_{st}(s) \cup Simple(s)$ and there is no other interpretation of σ^f which satisfies this set, i.e.,

$$\{s\} = \{s' \in I(\sigma^f) \mid s' \models T_{st}(s) \cup Simple(s)\}$$

- A label is an interpretation $I(\sigma^a)$ of σ^a.

Table 1: Common Abbreviations for Causal Laws.

Abbreviation	Expanded Form	Informal Meaning
NONEXECUTABLE H IF F	$\bot$ AFTER $H \wedge F$	F is a precondition of H
H CAUSES F IF G	F IF $\top$ AFTER $G \wedge H$	F is true after H is executed in a state in which G is true
INERTIAL F	F IF F AFTER F, $\neg F$ IF $\neg F$ AFTER $\neg F$	F keeps its value after a transition if its value is not affected by the transition
DEFAULT F	F IF F	F is true in the absence of information to the contrary
EXOGENOUS F	DEFAULT F, DEFAULT $\neg F$	F may be true or false in any state
F IFF G	F IF G, DEFAULT $\neg F$	F is true only if G is true

- A transition (s,e,s') is a transition of TS iff both s and s' are states of D, e is label of D and

$$\{s'\} = \{s'' \in I(\sigma^f) \mid s'' \models T_{st}(S') \cup E(s,e,s')\}$$
$$\{e\} = \{e' \in I(\sigma^a) \mid e' \models A(s,e)\}$$

where T_{st}, E, A and $Simple$ are defined as follows:

$$T_{st}(s) = \{F \mid \text{“}F \text{ IF } G\text{”} \in D, s \models G\}$$
$$E(s,e,s') = \{F \mid \text{“}F \text{ IF } G \text{ AFTER } H\text{”} \in D, \quad s' \models G, s \cup e \models H\}$$
$$A(s,e) = \{A \mid \text{“}\alpha \text{ IF } G\text{”} \in D, s \cup e \models G\}$$
$$Simple(s) = \{c = v \mid c \in \sigma^{si}, s \models c = v\}$$

An example description of a labeled transition system using $C+$ is presented in the appendix.

3 POLICY SPECIFICATION

Our approach for the dynamic analysis of usage control policies is composed of two steps: the target domain is first described as an action description D and its usage control policy $\mathcal{P}_U$ is separately specified using the usage control policy language presented in Section 3.1. Given the target action description and its usage policies, a new action description, denoted $D_{\mathcal{P}_U}$, is constructed to model the target domain when the specified usage policies are enforced in it, as explained in Section 4. Given $D_{\mathcal{P}_U}$, we are able to automate the verification of usage control policy properties as discussed in Section 5.

To illustrate our approach, we use as running example a movie player application *iX*. The users of *iX* can turn the application on, play movies and turn the application off. The users can also accept the application usage terms, pay for movies and close an ad window that appears when a movie is played. Users of *iX* may have different roles, e.g., a user can be a *privileged* user or a *blacklisted* user. In the following, this example is referred to as Example 1.

Table 2: Fluent Constants.

Fluent Constant	Informal Meaning
ad_window	the ad-window is open.
payment_window	the payment-window is open.
terms_window	the terms-window is open.
terms_accepted	the application terms are accepted.
user_role	the role of the user requesting access.
state	the movie application state.

Table 3: Action Constants.

Action Constant	Informal Meaning
turn_on	turns on the application.
quit	quits the application.
play	plays a movie.
stop	stops the movie.
accept_terms	accept the application usage terms.
pay	pay for a movie.
close_ad	closes the ad-window.

To formally describe *iX*, we consider an *action description* with a signature that includes the *inertial* fluent constants shown in Table 2 and action constants shown in Table 3. All fluent constants in the signature are Boolean with the exception of *state*, whose domain is $\{$*off, on, playing*$\}$, and *user_role*, whose domain is $\{$*regular, privileged, blacklisted*$\}$. All action constants are exogenous.

Actions of the action description and their executability conditions are described using causal laws. For example, the following rules specify that the execution of the action *accept_terms* makes the fluent *terms_accepted* true and the fluent *ad_window* false; and that the action *accept_terms* cannot be executed if the *payment_window* is not active:

$$accept\_terms \text{ CAUSES } terms\_accepted$$
$$accept\_terms \text{ CAUSES } \neg ad\_window$$
$$\text{NONEXECUTABLE } accept\_terms \text{ IF } \neg terms\_window$$

The full action description is given in the appendices. In the following, the set of fluents and actions of the action description of the target domain will be denoted by $\mathcal{F}$ and $\mathcal{A}$, and will be called the domain fluents and the domain actions, respectively.

3.1 Usage Policy Language

The language for the specification of the usage control policy of the target domain includes permissions, prohibitions, action obligations and state obligations. We first clarify concepts underlying usage control to provide justification for elements of the policy language.

Usage Session. Usage controls specify constraints that should be satisfied before, while and after a usage session. Figure 1 shows the *phases* of a typical usage control session (explained below) and *actions* relevant to each. Actions in a usage session can be classified into *user actions* (the actions *tryaccess* and *endaccess* in Figure 1) and *controller actions* (the actions *denyaccess*, *permitaccess*, *revokeaccess* and *precontrol* in Figure 1).
Admission Control is the first control phase in a usage session; it checks whether the requested access is authorized by the policy. Three possible decisions may be reached after policy evaluation: (1) access should be denied, (2) access should be allowed or (3) access should be allowed provided the satisfaction of some pre-access requirements, e.g., to accept the usage terms of the application.
Pre-usage Control starts after admission control if the requested access further requires the satisfaction of some pre-access requirements. During this phase, the fulfilment of the pre-access requirements is monitored: if they are successfully fulfilled, access is granted, otherwise access is denied.
Ongoing-usage Control starts after the satisfaction of the pre-access requirements. During this phase, ongoing-access requirements are monitored: access continues as long as the requirements are satisfied, otherwise access is revoked. This phase ends when either access is revoked or when the action terminating the usage session is taken.

Usage Controls. The three *decision factors* of usage control are authorizations, obligations and conditions (Sandhu and Park, 2004). They are evaluated before and while access. Our usage policy language aims at covering the expression of these elements. A usage control policy $\mathcal{P}_U$ for an action description D is therefore a set of the following propositions:

$$\textsc{permitted}\ A_h\ \textsc{if}\ F_b \tag{1}$$
$$\textsc{prohibited}\ A_h\ \textsc{if}\ F_b \tag{2}$$
$$\textsc{usage}\ [A_s - A_e]\{\ [\ \textsc{obl}\ ]^*\ \} \tag{3}$$

where F_b is a fluent formula composed using the domain fluents and A_h, A_s and A_e are domain actions. An authorization rule of the form (1) ((2)) specifies that the execution of A_h is allowed (denied) if the fluent formula F_b holds. A usage rule (3) states that a usage session is initiated by the domain action A_s and terminated by A_e. A_s is called the *start action* of the usage session and A_e is called the *end action* of the usage session. The usage controls applicable for a usage session $[A_s, A_e]$ are the set of obligations $\textsc{obl}$. These obligations can define either a pre-obligation or an ongoing-obligation requirement. Pre- and ongoing obligation requirements can be either action or state obligations. Obligation are specified in the language using propositions having one of the following forms:

$$\textsc{pre}_a^d\ A_h\ \textsc{if}\ F_b \qquad \textit{action pre-obligation} \tag{4}$$
$$\textsc{pre}_m\ F_h\ \textsc{if}\ F_b \qquad \textit{state pre-obligation} \tag{5}$$
$$\textsc{On}_a^d\ A_h\ \textsc{if}\ F_b \qquad \textit{ongoing action obligation} \tag{6}$$
$$\textsc{On}_m\ F_h\ \textsc{if}\ F_b \qquad \textit{ongoing state obligation} \tag{7}$$

where A_h is a domain action, F_h and F_b are fluent formulas and d is a positive integer representing the obligation deadline. In a rule, F_h and A_h are called the head and F_b the body of the rule respectively. We impose a restriction that is that any domain action A in D can start only one usage session, i.e., there may be at most one usage rule (3) with A as its start action. Otherwise, it would be unclear which set of usage controls applies for usage sessions initiated by A. Also deadlines are not intended to represent real temporal deadlines but to specify a relative temporal ordering between obligations. Therefore, a limit is imposed on the maximum deadline value. However, this is generally sufficient to correctly model the target domain by ensuring a timely violation of obligations. The management of obligations and their deadlines is explained in Section 4.3.

The following rules specify a usage control policy for the application iX: a user is allowed to play a movie if he is not blacklisted. The action $play$ starts a usage session that requires the fulfilment of a conditional pre-obligation to accept the usage terms within two minutes (unless they have been already accepted) and an unconditional obligation to pay for watching the movie within three minutes. The usage policy also specifies a conditional ongoing obligation specifying that the user has to keep the ad window open during the session (unless he is a privileged user). The usage session is ended when the action $stop$ is taken.

$$\textsc{permitted}\ play\ \textsc{if}\ \neg user\_role = blacklisted$$
$$\textsc{usage}\ [play - stop]\{$$
$$\textsc{pre}_a^2\ accept\_terms\ \textsc{if}\ \neg terms\_accepted$$

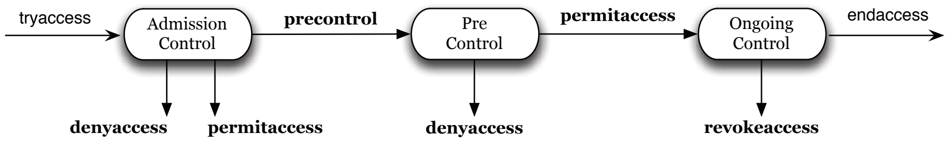

Figure 1: Usage Control Session.

PRE_a^3 *pay* IF *true*

ON_m *ad_window* IF $\neg$*user_role* = *privileged* }

In the following, we use $\Gamma(X)$ to denote the body of a rule whose head is X. It will be also called the activation condition of the corresponding rule. The set of pre-obligations and ongoing obligations of some usage session $[A_s, A_e]$ will be denoted as $p(A_s, A_e)$ and $o(A_s, A_e)$, respectively.

4 USAGE-CONTROLED DESCRIPTIONS

To study the effects of the enforcement of a usage policy $\mathcal{P}_U$ on the target domain, the action description of the target domain D is extended into a usage-controled action description $D_{\mathcal{P}_U}$, modeling the target domain D after the enforcement of the policy $\mathcal{P}_U$ in it. The extended action description is constructed as follows: First, every domain action A in D is replaced by two actions in $D_{\mathcal{P}_U}$: a *tryaccess(A)* action (to represent a request to execute A) and a *permitaccess(A)* action (to represent the execution of A). The *tryaccess(A)* action has the same executability conditions of A, and the execution of the action *permitaccess(A)* has the same effects of A. The *tryaccess(A)* actions are called *user actions* in the following discussions.

A number of *policy fluents* and *policy actions* are also added to the signature to model usage policy enforcement. These elements are necessary to reason about the effects of the application of $\mathcal{P}_U$ on the domain D. In particular, policy fluents are used to describe the *policy state*, e.g., they describe the state of usage sessions and usage controls in the system. On the other hand, the policy actions define the operations used to update and manage the usage control policy state. The effects of policy actions on policy fluents is formally specified using causal laws, clarifying the update of the usage control policy after the execution of policy management actions. The policy management causal laws are called the *policy management rules*. The action description extended with the aforementioned actions, fluents and causal laws define the usage-controlled action description $D_{\mathcal{P}_U}$.

Table 4: Fluent and Action Constants.

Fluent Constant	**Domain**
session(play-stop)	{initial, requesting, waiting, accessing, ended, denied, revoked}
session(stop-none)	{initial, requesting, waiting, accessing, ended, denied, revoked}

Action Constant	**Description**
denyaccess(play)	deny access
permitaccess(play)	start ongoing-usage controls
precontrol(play)	activate pre-usage controls
revokeaccess(play)	revoke access

The sets of policy fluents and actions of $D_{\mathcal{P}_U}$ are denoted by $\mathcal{F}_P$ and $\mathcal{A}_P$, respectively.

4.1 Extending the Language Signature

Controller Actions and Usage Sessions: Figure 2 shows an abstract representation of a usage session. A usage session may be in one of the states {*idle, requesting, waiting, accessing, revoked, denied, ended*}. Change in the state of a usage session is caused by either a user action $A \in$ {*tryaccess, endaccess*}, i.e. actions that start or end a usage session; or a session management action $A \in$ {*permitaccess, precontrol, denyaccess, revokeaccess*}, i.e. actions that update the usage session state.

Every domain action A in D is associated with a usage session, represented using a policy fluent.When A is the start action of a usage rule (6) in the specified usage control policy, e.g., the action *play*, then this policy fluent has the form *session(A-A′)*, where A' is the action that ends the usage session, e.g., *stop*. When A is not the start action of any usage rule (6), e.g., the action *stop*, then the policy fluent has the form *session(A,none)*. A policy action is also added for every session management action appearing in Figure 2. Table 4 shows the action and fluent constants added for the domain actions *play* and *stop*.

Authorization Fluents: Authorization rules of the form 4 or 5 in the policy are represented using statically determined fluents that hold only when their ac-

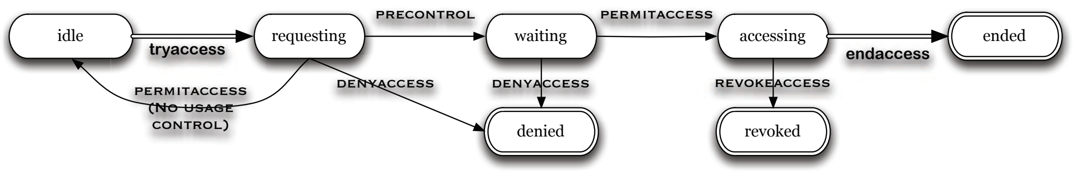

Figure 2: Usage Control State Model

tivation conditions hold:

$$perm(A_h) \text{ IFF } \Gamma(A_h) \qquad proh(A_h) \text{ IFF } \Gamma(A_h)$$

For example, the permission to execute the action *play* in Example 1 is represented as follows:

$$perm(play) \text{ IFF } \neg user\_role = blacklisted$$

Obligation Fluents: Every action obligation in the usage policy $\mathcal{P}_U$ is denoted using a separate policy fluent of the form $obl_a^d(A)$[1]. The domain of this fluent is {*inactive, active(X), violated, fulfilled*} where $X \in \{1,\ldots,d\}$, representing the possible states of the obligation. When the obligation is a state obligation, its policy fluent is of the form $obl_s(F)$ and its domain includes the values {*inactive, active*}. Violated and fulfilled state obligations are represented using two statically determined boolean fluents of the form $obl_s(F)$*=fulfilled* and $obl_s(F)$*=violated*, respectively. Action and state obligations are represented differently since the fulfilment/violation of a state obligation is defined in terms of state conditions whereas the fulfilment/violation conditions of action obligations is caused by action occurrences. This will be clarified in Section 4.3. Table 5 shows the fluents used corresponding to obligations in Example 1.

Table 5: Obligation Fluent Constants.

Fluent Constant	Domain
obl_a^2*(accept_terms)*	{inactive, active(2), active(1), fulfilled, violated}
obl_a^3*(pay)*	{inactive, active(3), active(2), active(1), fulfilled, violated}
obl_s*(ad_window)*	{inactive, active}
obl_s*(ad_window)=violated*	boolean
obl_s*(ad_window)=fulfilled*	boolean

Pre-usage and Ongoing Usage Controls: To link pre-obligations and ongoing obligations to usage sessions, we use a boolean statically determined fluent that holds only when the activation conditions of the obligation hold. This fluent is defined for each obligation in $\mathcal{P}_U$ according to the obligation type as follows:

$$\begin{array}{ll} preobl([A_s,A_e],A_h) \text{ IFF } \Gamma(A_h) & \textit{action pre-obligation} \\ preobl([A_s,A_e],F_h) \text{ IFF } \Gamma(F_h) & \textit{state pre-obligation} \\ onobl([A_s,A_e],A_h) \text{ IFF } \Gamma(A_h) & \textit{action on-obligation} \\ onobl([A_s,A_e],F_h) \text{ IFF } \Gamma(F_b) & \textit{state on-obligation} \end{array}$$

Obligations in Example 1 are linked to the usage session [*play,stop*] as follows:

$$\begin{array}{l} preobl([play,stop],pay) \text{ IFF } \top \\ preobl([play,stop],accept\_terms) \text{ IFF } \neg terms\_accepted \\ onobl([play,stop],ad\_window) \text{ IFF } \\ \qquad \neg user\_role = privileged \end{array}$$

4.2 Usage Control Management

Usage policy management rules are causal laws, included into the usage controlled action description $D_{\mathcal{P}_U}$. In this section, the notation *obl(X)* is used to represent both an action obligation $obl_a^d(X)$ and a state obligation $obl_s(X)$ when their management rules are specified similarly. Figure 3 illustrates the main policy management operations discussed in this section.

Admission Control: It starts after an access request, i.e., when a *tryaccess(A)* action occurs, and consists of an evaluation of the policy: the request is denied by taking the action *denyaccess* if the action is not authorized and, consequently, the usage session state becomes *denied*. Otherwise, if the action does not start a usage session, then access is immediately granted (the action *permitaccess(A)* is taken) and the session state returns to *idle*. If the action starts a usage session, then the action *precontrol* is initiated to start the pre-usage control phase, making the usage session *waiting*. The following causal laws define this policy enforcement paradigm, illustrated in Figure 3. Note that Figure 3 does not show the effects of the policy management actions *denyaccess(A)* and *revokeaccess(A)* due to space limitation.

[1]We slightly abuse the notation and use subscripts and superscripts when writing obligation fluents for clarity.

tryaccess(A) CAUSES *session(A,$\overline{A}$)=requesting*
  IF *session(A,$\overline{A}$)=idle*
denyaccess(A) IF *session(A,$\overline{A}$)=requesting* &
  ¬allowed(A)
denyaccess(A) CAUSES *session(A,$\overline{A}$)=denied*
precontrol(A,A') IF *session(A,A')=requesting* &
  allowed(A)
precontrol(A) CAUSES *session(A,A')=waiting*
  IF *session(A,A')=requesting*
permitaccess(A) IF *session(A,none)=requesting* &
  allowed(A)
permitaccess(A) CAUSES *session(A,none)=idle*
  IF *session(A,none)=requesting*

where A and A' are variables ranging over the set of domain actions, $\overline{A}$ is a variable over the set $A \cup \{none\}$ and $allowed(A)$ is a statically determined fluent defining the admission control policy evaluation strategy. The following strategies may be considered:

$allowed(A)$ IFF $perm(A)$	*closed-policy*
$allowed(A)$ IFF $\neg proh(A)$	*open-policy*
$allowed(A)$ IFF $perm(A)$ & $\neg proh(H)$	*precedence*

The *closed-policy* (*open-policy*) strategy states that an action is allowed only if there is an explicit permission (prohibition) authorizing (forbidding) it. The *precedence* strategy specifies that an action is authorized only if it is explicitly permitted and not prohibited.

Pre-usage Controls Activation: The start of the pre-usage control phase, i.e., the execution of the action *precontrol(A)*, activates the pre-obligations defined for the usage session that is started by *A*. Action and state obligations are activated as follows:

precontrol(A,A') CAUSES *obl$_s$(P)=active* IF *preobl([A,
  A'],P)* & *obl$_s$(P)=inactive where P $\in$ p(A,A')*
precontrol(*A,A'*) CAUSES $obl_a^d(A_h)$*=active(d)* IF *preo-
  bl*$([A,A'],A_h)$ & $^d_a(A_h)$*=inactive where* $A_h \in$ *p(A,A')*

The previous rules specify that *precontrol(A)* activates a pre-obligation $X \in p(A,A')$ if X is required, i.e., $preobl([A,A'],X)$ holds and the obligation is *inactive*.

Pre-usage Controls Monitoring: Activated pre-obligations are monitored in the state *waiting* as follows: if any of the pre-obligations is violated, then *denyaccess(A)* is initiated to end the usage session. Access is granted whenever all the *required* pre-obligations are satisfied. This occurs when all pre-obligations are in either the *fulfilled* or *inactive* state. This is specified using the following rules:

$$denyaccess(A) \text{ IF } session(A,A') = waiting \;\& \bigvee_{P \in p(A,A')} obl(P) = violated$$

$$permitaccess(H) \text{ IF } session(A,A') = waiting \;\& \bigwedge_{P \in p(A,A')} (obl(P) = inactive \vee obl(P) = fulfilled)$$

Pre-usage Control Satisfaction/Violation: The satisfaction of the pre-usage controls starts the ongoing-usage control phase, i.e. session state becomes *accessing*. Active pre-obligations are canceled, i.e. their state becomes *inactive* after the end of the pre-usage control phase as follows:

permitaccess(A) CAUSES *session(A,A')=accessing*
  IF *session(A,A')=waiting*
denyaccess(A) CAUSES *obl(P)=inactive*
  IF *¬obl(P)=inactive*
permitaccess(A) CAUSES *obl(P)=inactive*
  IF *¬obl(P)=inactive where* $P \in p(A,A')$

Ongoing Usage Controls Activation: The *permitaccess(A)* action starts the ongoing-usage control phase and activates the session's ongoing obligations $O \in o(A,A')$ as follows:

permitaccess(A) CAUSES *obl$_s$(O)=active*
  IF *onobl([A,A'],O)* & *obl$_s$(O)=inactive*
permitaccess(A) CAUSES $obl_a^d(O)$*=active(d)*
  IF *onobl([A,A'],O)* & $obl_a^d(O) = inactive$

Ongoing Usage Controls Monitoring: In state *accessing*, ongoing obligations are monitored. If any of the ongoing obligations is violated, then the usage session is revoked.

$$revokeaccess(A) \text{ IF } session(A,A')=accessing \;\& \bigvee_{O \in o(A,A')} obl(O)=violated$$

Ending of the Usage Session: The usage session is ended if either access is revoked or the end action of the usage session is executed. In the former case, the usage session state becomes *revoked*. In the latter case, the usage session state becomes *ended*. After the end of a usage session, all ongoing-obligations

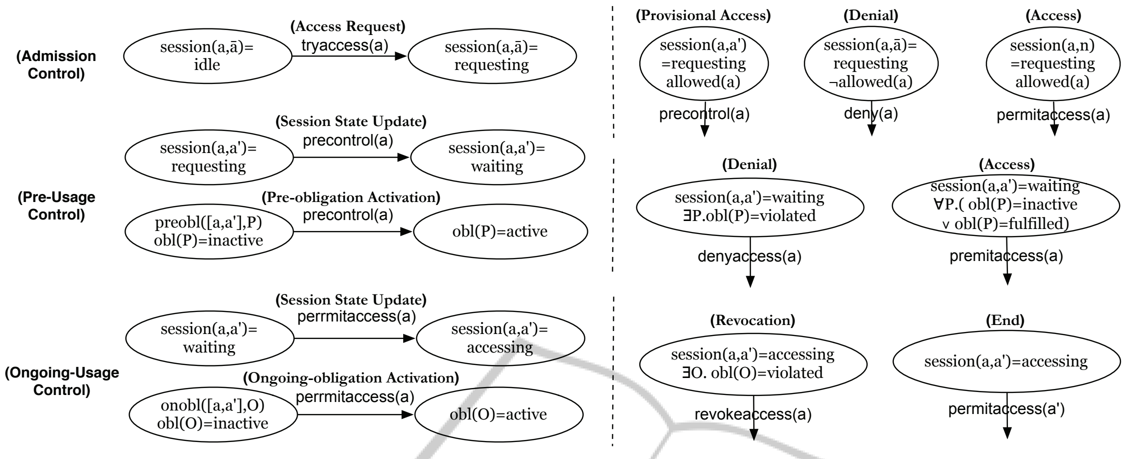

Figure 3: Usage Control Enforcement.

$O \in o(A,A')$ are canceled.

revokeaccess(A) CAUSES *session(A,A')=revoked*
  IF *session(A,A')=accessing*
revokeaccess(A) CAUSES *obl(O)=inactive*
  IF $\neg$*obl(O)=inactive*
permitaccess(A') CAUSES *session(A,A')=ended*
  IF *session(A,A')=accessing*
permitaccess(A') CAUSES *obl(O)=inactive*
  IF $\neg$*obl(O)=inactive*

4.3 Obligation Management

Usage controls are either action or state obligations. This section formalizes the violation/fulfilment conditions of obligations and the update of their state.

State Obligation Fulfilment/Violation: State obligations are fulfilled (violated) if they are *active* and their required state conditions hold (don't hold). The fulfilment and violation of a state obligation of the form (8) or (10) is defined in terms of the fluents $obl(F_h)$*=violated* and $obl(F_h)$*=fulfilled* as follows:

$obl_s(F_h)$*=fulfilled* IFF $obl_s(F_h)$*=active* & F_h
$obl_s(F_h)$*=violated* IFF $obl_s(F_h)$*=active* & $\neg F_h$

The following rules define the fulfilment and violation of the ongoing obligations in Example 1.

obl_s*(ad_window)=fulfilled*
  IF obl_s*(ad_window)=active* &*ad_window*
obl_s*(ad_window)=violated*
  IF obl_s*(ad_window)=active* &$\neg$*ad_window*

State Obligation Cancellation: Obligations are contextual. Therefore, an obligation is canceled when its conditions no longer hold. This is specified as follows for state obligations of the form (8) or (10):

$obl_s(F_h)$*=inactive* IF $\neg\Gamma(F_h)$

For example, the following rule specifies when the ongoing obligation in Example 1 is canceled.

obl_s*(ad_window)=inactive* IF $\neg$*user_role=privileged*

Action Obligation Fulfilment: An *active* action obligation of the form (7) and (9) is fulfilled after the execution of its action as follows:

$obl_a^d(A_h)$*=fulfilled* AFTER $obl_a(A_h)$*=active(N)* &
  permitaccess(A_h) *where* $N \in \{1,\ldots,d\}$

The fulfilment of action obligations in Example 1 is defined as follows:

obl_a^2*(accept_terms)=fulfilled* IF *true*
  AFTER obl_a^2*(accept_terms)=active*(N_1) &
  permitaccess(accept_terms) where $N_1 \in \{1,2\}$
obl_a^3*(pay)=fulfilled* IF *user_role=privileged*
  AFTER obl_a^3*(pay)=active*(N_2) &
  permitaccess(pay) where $N_2 \in \{1,2,3\}$

Action Obligation Cancellation: An action obligation is canceled if its conditions do not hold and the obligation was not yet fulfilled.

$obl_a^d(A_h)$*=inactive* IF $\neg\Gamma(A_h)$
  AFTER $\neg$*permitaccess*(A_h)

The cancellation of the action obligations of Example 1 is defined as follows:

obl_a^2*(accept_terms)=inactive* IF $\neg$*true*
  AFTER $\neg$*permitaccess(accept_terms)*
$obl_a^3(pay) = inactive$ IF $user\_role = privileged$
  AFTER $\neg permitaccess(pay)$

Note that $\neg\neg f$ is equated with f and that the obligation to accept terms can not be canceled since its cancellation conditions can never be true, namely $\neg true$.

Action Obligation Violation: An active action obligation is violated after the elapse of the period specified in its deadline. An action obligation can be in different *active* states according to the time that is left before its deadline elapses. In particular, an obligation $Obl_a^d(A)$ is in the state *active(d)* when it is activated, as presented in Section 4.2. The passage of time is simulated using an *exogenous* action *clock*, whose occurrence updates an obligation in the state $active(n)$ to the state $active(n')$ by decreasing the value of n by one, i.e., $n' = n - 1$. The obligation is violated after an occurrence of the action *clock* when the state of the obligation is $active(1)$ and the obligation has not been canceled nor fulfilled. This is specified using the following rules for action obligations of the form (7) or (9):

$$obl_a^d(A_h){=}violated \text{ IF } \Gamma(A_h) \text{ AFTER } obl_a^d(A_h){=}active(1) \,\&\, clock \,\&\, \neg permitaccess(A_h)$$

$$clock \text{ CAUSES } obl_a^d(A_h){=}active(d') \text{ IF } obl_a^d(A_h){=}active(d) \text{ where } d > 1, d' \text{ is } d-1$$

State Update. The update of state as a result of access enables, for example, to change a subject's attributes after access revocation (Sandhu and Park, 2004). In our framework, state update is specified as post effects of the usage session management actions. For example, access revocation may cause the role of the user to become *blacklisted* as follows:

$$revoke(play) \text{ CAUSES } user\_role{=}blacklisted$$

Similarly, the state can be updated after access acceptance and denial using the actions $permitaccess(A)$ and $denyaccess(A)$ respectively. Note that it is similarly possible to activate obligations after the execution of usage session management actions. This allows specification of post-obligations in our framework, i.e. requirements that should be satisfied after the end or revocation of access. The support of post-obligations is however not discussed in this paper due to space limitations.

Multiple Sessions. In Figure 2, the states *denied*, *revoked* and *ended* are terminal states. To support multiple usage sessions, the session state is reset to the state *idle* when a terminal state is reached by initiating the policy management action *reset_session*:

$$reset\_session(A,A') \text{ IF } session(A,A'){=}denied \vee session(A,A'){=}ended \vee session(A,A'){=}revoked$$

$$reset\_session(A,A') \text{ CAUSES } session(A,A'){=}idle$$

5 POLICY ANALYSIS

The use of $C+$ enables the use of model checking techniques for the verification of properties of labeled transition systems described by action language descriptions. This section identifies and formalizes some important usage control policy properties using Action-based Computational Tree Logic (ACTL) (Bouali et al., 1994), enabling their verification using standard ACTL-model checkers. The automated verification of usage control policies ensures the correctness of usage control policies before their deployment. For example, it enables to ensure resource *availability*, i.e. that the specified security controls do not make system resources inaccessible. It also ensures that security controls cannot be circumvented, i.e. that resources can only be accessed after the satisfaction of the specified security controls. The complexity of model checking algorithms for ACTL formulas is linear in the state space of the system model and the formula.

Labeled Transition System of an Action Description. An action description D defines a labeled transition system $TS = \langle S, L, \rightarrow \rangle$ as described in Section 2.3:

- If $q \in S$ is a state and $A \in L$ is an action, $R_A(q)$ denotes the set of actions that the state q can take, i.e., the set $\{q' \mid \textit{there exists } \alpha \in A \textit{ such that } (q, \alpha, q') \in \rightarrow\}$. $R(q)$ denotes $R_L(q)$.
- A *path* in TS is a finite or infinite sequence $q_1, q_2, \ldots$ such that every $q_{i+1} \in R(q_i)$;
- We denote the set of paths starting from a state q by $\Pi(q)$, and use $\sigma, \sigma', \ldots$ to range over paths;
- A path is *maximal* if it is infinite, or finite and its last state q' has no successor states, i.e., $R(q') = \emptyset$;
- If σ is infinite then $|\sigma| = \omega$; if $\sigma = q_1, q_2, \ldots, q_n$, then $|\sigma| = n - 1$. Moreover, if $|\sigma| \geq i - 1$, then $\sigma(i)$ denotes the i^{th} state in the sequence.

Action Formulas. Let $b \in L$. The language $\mathcal{L}(L)$ of action formulas on L is defined as follows:

$$\mathcal{X} ::= \top \mid b \mid \neg\mathcal{X} \mid \mathcal{X} \vee \mathcal{X}$$

where b ranges over L. The satisfaction relation $\models$ for action formulas is defined as:

$$\begin{array}{lll} a \models \top & & always \\ a \models b & iff & a = b \\ a \models \neg\mathcal{X} & iff & not\ a \models \mathcal{X} \\ a \models \mathcal{X} \vee \mathcal{X} & iff & a \models \mathcal{X}\ or\ a \models \mathcal{X}' \end{array}$$

Let $\mathcal{X}$ be an action formula, the set of actions satisfying $\mathcal{X}$ is characterized by the function $k : \mathcal{L}(L) \rightarrow 2^L$

as follows:

$$\begin{array}{lcl} k(\top) & = & L \\ k(b) & = & \{b\} \\ k(\neg \mathcal{X}) & = & L \setminus k(\mathcal{X}) \\ k(\mathcal{X} \vee \mathcal{X}') & = & k(\mathcal{X}) \cup k(\mathcal{X}') \end{array}$$

ACTL Syntax. ACTL is a branching-time temporal logic of state formulas (denoted by ϕ), in which a path quantifier prefixes an arbitrary path formula (denoted by π). The syntax of ACTL formulas is given by the following grammar[2]:

$$\begin{array}{lcl} \phi & ::= & \top \mid \phi \wedge \phi \mid \neg\phi \mid E\pi \mid A\pi \\ \pi & ::= & X_{\mathcal{X}}\phi \mid \phi_{\mathcal{X}}U\phi \mid \phi_{\mathcal{X}}U_{\mathcal{X}'}\phi \end{array}$$

where $\mathcal{X}$ and $\mathcal{X}'$ range over action formulas, E and A are the existential and universal path quantifiers, respectively. X and U are the next and until operators, respectively. The following is a set of common abbreviations of other modalities: the eventuality operators $EF\phi \leftrightarrow E(\top_{\top}U_{\mathcal{X}}\phi)$ and $AF\phi \leftrightarrow A(\top_{\top}U_{\mathcal{X}}\phi)$; the always operators $EG\phi \leftrightarrow \neg AF\neg\phi$ and $AG\phi \leftrightarrow \neg EF\neg\phi$.
ACTL Semantics. The satisfaction relation for ACTL formulas is defined as follows:

$s \models \top$		*always*;
$s \models \phi \wedge \phi'$	*iff*	$s \models \phi$ *and* $s \models \phi'$;
$s \models \neg\phi$	*iff*	*not* $s \models \phi$;
$s \models E\pi$	*iff*	*there exists a path* $\sigma \in \Pi(s)$ *such that* $\sigma \models \pi$;
$s \models A\pi$	*iff*	*for all maximal paths* $\sigma \in \Pi(s)$, $\sigma \models \pi$;
$s \models X_{\mathcal{X}}\phi$	*iff*	$\lvert\sigma\rvert \geq 1$ *and* $\sigma(2) \in R_{k(\mathcal{X})}(\sigma(1))$ *and* $\sigma(2) \models \phi$;
$s \models \phi_{\mathcal{X}}U\phi'$	*iff*	*there exists* $i \geq 1$ *such that* $\sigma(i) \models \phi'$ *and for all* $i \leq j \leq i-1, \sigma(j) \models \phi$ *and* $\sigma(j+1) \in R_{k(\mathcal{X})}(\sigma(j))$;
$s \models \phi_{\mathcal{X}}U_{\mathcal{X}'}\phi'$	*iff*	*there exists* $i \geq 2$ *such that* $\sigma(i) \models \phi'$ $\sigma(i-1) \models \phi, \sigma(i) \in R_{k(\mathcal{X}')}(\sigma(i-1))$ *and for all* $i \leq j \leq i-2$, $\sigma(j+1) \in R_{k(\mathcal{X})}(\sigma(j))$.

Analysis. Table 6 identifies and summarizes some important usage control policy properties in the form of ACTL formulas, enabling the automation of their formal verification.

System Accountability: A system is said to be *accountable* if obligations assigned by the system can be fulfilled (Irwin et al., 2006). In our work, two factors impact system accountability, namely authorization policies and action executability conditions.

- An action obligation can only be fulfilled if it is *allowed* by the specified authorization policies. An obligation action can be allowed throughout or only partially during the activation period of the obligation. These properties can be specified using the ACTL formulas (1) and (2) in Table 6 respectively for the action A_h of a pre-obligation of the form (7) in some usage rule $[A,A']$ in the usage control policy.
- Action Executability: An action A is *executable* at a state if the action *tryaccess*(A) can be taken in this state, i.e., that the action preconditions hold. Similarly to action allowance, the action of an obligation may be executable throughout or only partially during the activation period of the obligation. Formulas (3) and (4) in Table 6 enable the verification of these properties for the action A_h of a pre-obligation of the form (7) in a usage rule $[A,A']$ in the policy.

A system is strongly accountable only if the action of every obligation in the policy is both allowed and executable during the obligation lifetime. It is weakly accountable if every obligation action is both allowed and executable sometime during the obligation lifetime. The formulas (5) and (6) in Table 6 define when the system is accountable with respect to the action A_h of a pre-obligation of the form (7) in a usage rule $[A,A']$ in the usage control policy. Global system accountability is checked by verifying these properties for every pre-obligation action and every ongoing-obligation action in the usage control policy. We do not present the formulas for verifying global accountability nor formulas corresponding to ongoing obligations due to space limitation.

State Obligation Satisfiability and Violation: It is sometimes necessary to verify that a state obligation is satisfied at the moment of its activation and that the obligation may later be violated. For example, the activation of an ongoing obligation to keep an ad window open may be unreasonable if the ad window is not open at the moment the obligation is activated; also there should a path where the obligation is violated after its activation; otherwise the obligation is unnecessary (void). The ACTL formulas (7) and (8) respectively in Table 6 check whether the condition F_h required by an ongoing state obligation of the form (10), in a usage rule $[A,A']$ in the usage control policy, is satisfied after the activation of the obligation and that this condition may later become false during the obligation lifetime respectively.

[2]Note that we omit $X_{\tau}\phi$ from its original syntax description, as actions τ do not play a role in our formulation.

Table 6: ACTL Formulas for Verification of Usage Control Policy Properties.

	Property	**ACTL Formula**
1	*Pre-Obl. Action Fully Allowed*	$AG((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow allowed(A_h))$
2	*Pre-Obl. Action Partially Allowed*	$AG\ ((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow$ $E(session(A,A') = waiting_{\top}\ U\ allowed(A_h))$
3	*Pre-Obl. Action Fully Executable*	$AG\ ((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow EX_{tryaccess(A_h)} \top)$
4	*Pre-Obl. Action Partially Executable*	$AG\ ((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow$ $E(session(A,A') = waiting_{\top}\ U\ EX_{tryaccess(A_h)} \top))$
5	*System Strong Accountability*	$AG\ ((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow$ $(allowed(A_h) \land EX_{tryaccess(A_h)} \top))$
6	*System Weak Accountability*	$AG\ ((session(A,A')=waiting \land obl_a^d(A_h)=active) \rightarrow$ $E(session(A,A') = waiting_{\top}\ U\ (allowed(A_h) \land EX_{tryaccess(A_h)} \top)))$
7	*State On-Obl. Satisfaction*	$AX_{permitaccess(A)}(F_h)$
8	*State On-Obl. Violation*	$AG\ ((session(A,A')=accessing \land obl_s(F_h)=active) \rightarrow$ $E(session(A,A') = accessing_{\top}\ U\ \neg F_h))$

We have identified and formalized other important usage control specific properties such as consistency of controls, usage session termination and access denial and revocation. They are however not discussed in this paper due to space limitations.

6 RELATED WORK

A classification of policy analysis techniques can be made according to the type of security policies considered, e.g., access control, obligations or usage control policies, and the type of analysis targeted, i.e., static or dynamic. The static analysis of access control policies is by far the most studied. An overview of access control policy conflicts and their resolution techniques is presented in (Lupu and Sloman, 1999; Samarati and de Vimercati, 2001).

The dynamic analysis of access control policies is supported in (Becker and Nanz, 2007), which introduces a non-monotonic extension of *datalog* allowing specification of action effects in access control policy rules. A goal-oriented algorithm is used to find minimal action sequences that lead to a specified target authorization state. In (Li and Tripunitara, 2006), the *RBAC* policy scheme (Ferraiolo et al., 1995; Sandhu et al., 1996) is modeled as a state-transition system, where changes occur via administrative operations. Security analysis techniques answer questions such as whether an undesirable state is reachable and whether every reachable state satisfies some safety or availability properties. A framework for the dynamic analysis of security policies that include authorizations and obligations is presented in (Craven et al., 2009). In this framework, domains regulated by policies are formalized using the event calculus (Shanahan, 1999), enabling formal analysis of the target domain and its security policy. A meta-model for the analysis of authorization systems with obligations is introduced in (Irwin et al., 2006). This paper focuses on the study of the *system accountability* problem, i.e., ensuring that assigned obligations will be fulfilled if system subjects are diligent. The aforementioned works do *not* consider usage control systems where authorizations, obligations and usage sessions interact.

$UCON_{ABC}$ (Sandhu and Park, 2004) is formalized in (Zhang et al., 2005) using Lamport's Temporal Logic of Actions (TLA). The safety analysis of $UCON_A$ positive authorization policies is presented in (Zhang et al., 2006). This work is generalized in (Ranise and Armando, 2012). In comparison, we support analysis of usage policies that include authorizations, prohibitions and obligations. The model checking of policies specified using the Obligation Specification Language (OSL) (Hilty et al., 2007) is studied in (Pretschner et al., 2009). OSL is a temporal logic with explicit operators for cardinality and permissions whose semantics is defined over traces of parameterized events. In comparison, our work enables the formal description of the target domain and the automated verification of properties of usage control policies in their target domain. Furthermore, our work formalizes the management of usage controls and usage sessions, concepts that are not considered in (Pretschner et al., 2009). The *XACML* language and architecture are extended in (Li et al., 2012) to add support for access control policies with obligations and a framework for the specification and enforcement of data handling policies is presented in (Ardagna et al., 2008). These works do not address the dynamic analysis of usage control policies.

7 CONCLUSION

This paper introduces an approach for the formal analysis of usage control policies and study of their application on target domains. Thus, our work is a first step in providing policy officers a valuable means to formally verify the correctness of specified policies before their deployment. Furthermore, we have identified and formalized several usage control specific properties and automated their verification.

This work can be extended to support structured policies with pre-defined sets of basic entities (Elrakaiby et al., 2012). We also intend to investigate the use of $C+_{timed}$ (Craven and Sergot, 2005) to provide a more elegant representation of delays and deadlines, and to design heuristics for our approach to make model checking scale better to real-world scenarios.

REFERENCES

CCalc. http://www.cs.utexas.edu/∼tag/cc/.

iCCalc. http://www.doc.ic.ac.uk/∼rac101/iccalc/.

Ardagna, C. A., Cremonini, M., De Capitani di Vimercati, S., and Samarati, P. (2008). A privacy-aware access control system. *JCS*, 16(4):369–397.

Armando, A., Giunchiglia, E., Maratea, M., and Ponta, S. E. (2012). An action-based approach to the formal specification and automatic analysis of business processes under authorization constraints. *Journal of Computer and System Sciences*, 78(1):119–141.

Armando, A., Giunchiglia, E., and Ponta, S. E. (2009). Formal specification and automatic analysis of business processes under authorization constraints: An action-based approach. In *TrustBus*, volume 5695 of *LNCS*, pages 63–72. Springer.

Artikis, A. and Sergot, M. (2010). Executable specification of open multi-agent systems. *Logic Journal of IGPL*, 18(1):31–65.

Artikis, A., Sergot, M. J., and Pitt, J. (2007). An executable specification of a formal argumentation protocol. *Artificial Intelligence*, 171(10-15):776 – 804.

Babb, J. and Lee, J. (2013). Cplus2asp: Computing action language c+ in answer set programming. *Logic Programming and Nonmonotonic Reasoning*, page 122.

Becker, M. Y. and Nanz, S. (2007). A logic for state-modifying authorization policies. In *ESORICS*, volume 4734 of *LNCS*, pages 203–218. Springer.

Bouali, A., Gnesi, S., and Larosa, S. (1994). The integration project for the JACK environement. *Bulletin of the EATCS*, 54:207–223.

Casolary, M. (2011). *Representing the language of the causal calculator in answer set programming*. PhD thesis, Arizona State University.

Craven, R., Lobo, J., Ma, J., Russo, A., Lupu, E., and Bandara, A. (2009). Expressive policy analysis with enhanced system dynamicity. In *ASIACCS*, pages 239–250. ACM.

Craven, R. and Sergot, M. (2005). Distant causation in C+. *Studia Logica*, 79(1):73–96.

Dworschak, S., Grell, S., Nikiforova, V., Schaub, T., and Selbig, J. (2008). Modeling biological networks by action languages via answer set programming. *Constraints*, 13(1-2):21–65.

Elrakaiby, Y., Cuppens, F., and Cuppens-Boulahia, N. (2012). Formal enforcement and management of obligation policies. *DKE*, 71(1):127 – 147.

Ferraiolo, D., Cugini, J., and Kuhn, D. R. (1995). Role-based access control (rbac): Features and motivations. In *ACSAC*, pages 241–48. ACM.

Gebser, M., Grote, T., and Schaub, T. (2010). Coala: a compiler from action languages to ASP. In *Logics in Artificial Intelligence*, pages 360–364. Springer.

Gelfond, M. and Lifschitz, V. (1998). Action languages. *Electronic Transactions on AI*, 3(16).

Giunchiglia, E., Lee, J., Lifschitz, V., McCain, N., Turner, H., and Lifschitz, J. L. V. (2004). Nonmonotonic causal theories. *Artificial Intelligence*, 153:49–104.

Hilty, M., Pretschner, A., Basin, D., Schaefer, C., and Walter, T. (2007). A policy language for distributed usage control. In *ESORICS*, volume 4734 of *LNCS*, pages 531–546. Springer.

Irwin, K., Yu, T., and Winsborough, W. H. (2006). On the modeling and analysis of obligations. In *CCS*, pages 134–143. ACM.

Li, N., Chen, H., and Bertino, E. (2012). On practical specification and enforcement of obligations. In *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, CODASPY ’12, pages 71–82, New York, NY, USA. ACM.

Li, N. and Tripunitara, M. V. (2006). Security analysis in role-based access control. *TISSEC*, 9(4):391–420.

Lifschitz, V. (1999). Action languages, answer sets, and planning. In *The Logic Programming Paradigm*, pages 357–373. Springer.

Lupu, E. C. and Sloman, M. (1999). Conflicts in policy-based distributed systems management. *TSE*, 25(6):852–869.

Pretschner, A., Ruesch, J., Schaefer, C., and Walter, T. (2009). Formal analyses of usage control policies. In *ARES*, pages 98–105. IEEE.

Ranise, S. and Armando, A. (2012). On the automated analysis of safety in usage control: a new decidability result. In *NSS’12*, pages 15–28, Berlin, Heidelberg. Springer-Verlag.

Samarati, P. and de Vimercati, S. C. (2001). Access control: Policies, models, and mechanisms. In *Founda-*

tions of Security Analysis and Design, pages 137–196. Springer.

Sandhu, R., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2):38–47.

Sandhu, R. and Park, J. (2004). The UCON ABC usage control model. *TISSEC*, 7(1):128–174.

Sergot, M. (2004). An action language for modelling norms and institutions. Technical Report 2004/8, Imperial College London.

Shanahan, M. (1999). The event calculus explained. In *Artificial Intelligence Today*, pages 409–430. Springer.

Simon, R. T. and Zurko, M. E. (1997). Separation of duty in role-based environments. In *Proceedings of the 10th Computer Security Foundations Workshop*, pages 183–194. IEEE.

Son, T. C., Pontelli, E., and Sakama, C. (2012). Formalizing commitments using action languages. In *Proceedings of the 9th Conference on Declarative Agent Languages and Technologies*, volume 7169 of *LNCS*, pages 67–83. Springer.

Zhang, X., Parisi-Presicce, F., Sandhu, R., and Park, J. (2005). Formal model and policy specification of usage control. *TISSEC*, 8(4):351–387.

Zhang, X., Sandhu, R., and Parisi-Presicce, F. (2006). Safety analysis of usage control authorization models. In *ASIACCS*, pages 243–254. ACM.

APPENDIX

Action Description Example

Consider a signature that includes the two simple fluent constants $\{p, q\}$, the statically determined fluent constant *r* and the two actions $\{a1, a2\}$. The causal laws below specify that the fluent *q* is inertial, i.e., its value persists unless it is changed by an action occurrence; *p* is false by default but holds after the occurrence of *a1*; *r* is a statically determined fluent, i.e., its truth depends on other fluents, that holds only when *p* and *q* are true. The action *a1* is exogenous, i.e., it may occur at any state; *a2*, on the other hand, has to occur in every transition from a state where *r* is true. The execution of *a2* causes *p* & ¬*q* to be true. Finally, we specify that a void transition, i.e., the transition not including *a1* nor *a2*, is not executable.

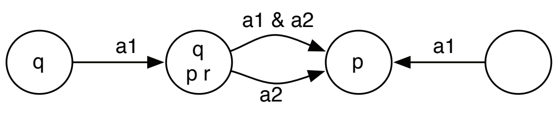

Figure 4: C+ Labeled Transition System Example.

Figure 4 shows the labeled transition system described by this action description. Note that we consider boolean fluent constants. A state in Figure 4 is labeled with the set of fluents that are true in it.

INERTIAL q.	DEFAULT $\neg p$.
DEFAULT $\neg r$.	r IF $p \& q$.
EXOGENOUS $a1$.	DEFAULT $\neg a2$.
$a2$ IF r.	$p \& \neg q$ AFTER $a2$.
p AFTER $a1$.	NONEXECUTABLE $\neg a1 \& \neg a2$.

Running Example

The description of the *iX* movie player example consists of a declaration of sorts and variables, fluent constants, action constants and causal laws.

Sorts	Fluents	Actions
$state(ready)$.	$fc(state)$.	$ac(turn\_on)$.
$state(off)$.	$domain(state, PS)$.	$ac(turn\_off)$.
$state(playing)$.	$fc(accept\_terms\_window)$.	$ac(play)$.
$PS :: state(PS)$.	$fc(payment\_window)$.	$ac(accept\_terms)$.
	$fc(ad\_window)$.	$ac(pay)$.
	$fc(terms\_accepted)$.	$ac(close\_ad)$.
	$fc(payment\_accepted)$.	

Causal Laws:

$exogenous(A)\ :-\ ac(A)$.
$inertial\ FC\ :-\ fc(FC)$.
CAUSED $state = ready$ AFTER $turn\_on$.
CAUSED $accept\_terms\_window$ IF $-terms\_accepted$ AFTER $turn\_on$.
CAUSED $payment\_window$ IF $-payment\_accepted$ AFTER $turn\_on$.
CAUSED ad_window AFTER $turn\_on$.
NONEXECUTABLE $turn\_on$ IF $-state = off$.
CAUSED $state = playing$ AFTER $play$.
NONEXECUTABLE $play$ IF $-state = ready$.
CAUSED $state = off$ AFTER $turn\_off$.
CAUSED $-payment\_window$ AFTER $turn\_off$.
CAUSED $-accept\_terms\_window$ AFTER $turn\_off$.
CAUSED $-ad\_window$ AFTER $turn\_off$.
NONEXECUTABLE $turn\_off$ IF $state = off$.
CAUSED $terms\_accepted$ AFTER $accept\_terms$.
CAUSED $-accept\_terms\_window$ AFTER $accept\_terms$.
NONEXECUTABLE $accept\_terms$ IF $-accept\_terms\_window$.
CAUSED $payment\_accepted$ AFTER pay.
CAUSED $-payment\_window$ AFTER pay.
NONEXECUTABLE pay IF $-payment\_window$.
CAUSED $accept\_terms\_window$ IF $-terms\_accepted$ AFTER $turn\_on$.
CAUSED $payment\_window$ IF $-payment\_accepted$ AFTER $turn\_on$.
CAUSED $-ad\_window$ AFTER $close\_ad$.
NONEXECUTABLE $close\_ad$ IF $-ad\_window$.
CAUSED $false$ IF $ad\_window \& state = off$.
CAUSED $false$ IF $payment\_window \& state = off$.
CAUSED $false$ IF $accept\_terms\_window \& state = off$.