A Logic for State-Modifying Authorization Policies

Moritz Y. Becker
Microsoft Research, Cambridge, UK
moritzb@microsoft.com

Sebastian Nanz
Informatics and Mathematical Modelling
Technical University of Denmark
nanz@imm.dtu.dk

March 2007

Technical Report

Microsoft Research
Roger Needham Building
7 J.J. Thomson Avenue
Cambridge, CB3 0FB
United Kingdom

A Logic for State-Modifying Authorization Policies

Moritz Y. Becker
Microsoft Research, Cambridge, UK
moritzb@microsoft.com

Sebastian Nanz
Informatics and Mathematical Modelling
Technical University of Denmark
nanz@imm.dtu.dk

March 2007

Abstract

Administering and maintaining access control systems is a challenging task, especially in environments with complex and changing authorization requirements. A number of authorization logics have been proposed that aim at simplifying access control by factoring the authorization policy out of the hard-coded resource guard. However, many policies require the authorization state to be updated after a granted access request, for example to reflect the fact that a user has activated or deactivated a role. Current authorization languages cannot express such state modifications; these still have to be hard-coded into the resource guard. We present a logic for specifying policies where access requests can have effects on the authorization state. The logic is semantically defined by a mapping to Transaction Logic. Using this approach, updates to the state are factored out of the resource guard, thus enhancing maintainability and facilitating more expressive policies that take the history of access requests into account. We also present a sound and complete proof system for reasoning about sequences of access requests. This gives rise to a goal-oriented algorithm for finding minimal sequences that lead to a specified target authorization state.

1 Introduction

Managing access control in large organizational networks is a challenging task. In the public sector, for instance, authorization is typically governed by a huge number of complex policies, regulations and laws. Perhaps even more challenging, in this regard, are modern decentralized applications that need to provide services and interact with each other over several independent administrative domains, such as web services, peer-to-peer networks and Grid systems. Much research has been done over the last decade on access control models that are more scalable and expressive than simple access control lists or groups. In particular, quite a number of expressive *authorization languages* (such as XACML

[44]) have been developed since the introduction of the trust management approach by Blaze et al. [13].

In this approach (see Figure 1), the authorization policy is factored out of the hard-coded resource guard and written explicitly as a list of declarative rules. When a principal requests access, the resource guard issues an authorization query to the policy evaluator. Access is granted only if the policy evaluator succeeds in proving that the request complies with the local policy and the authorization state. The latter is a database containing relevant environmental facts including knowledge obtained from (submitted or fetched) credentials.

This approach hugely increases the maintainability of complex systems, as modifying the declarative policy rules is much simpler than rewriting and recompiling pieces of procedural code hidden in the resource guard. Furthermore, some authorization languages have a precise semantics that enables implementation-independent rigorous analysis of policies.

However, often the resource guard will not only allow or deny access, but also update the authorization state after a successful request. In a role-based policy, for instance, the fact that a user has activated some role is inserted into the authorization state after a successful role activation request. Similarly, the fact may be removed from the state if the role is deactivated. There are many policies that depend on past interactions; relevant events must therefore be stored in the authorization state. Consider for example the following scenario, where a company policy specifies that payments are only executed if they are initiated and authorized by two different managers. The resource guard could then contain the following two procedures to implement this policy:

```
procedure initPay(X:Principal, P:Payment) {
  if query(isMgr(X)) and
     not(query(hasBeenInit(P))) then {
       insertFact(hasBeenInit(P));
       insertFact(hasInitPay(X, P))
     }
}
procedure authPay(X:Principal, P:Payment) {
  if query(isMgr(X)) and
     query(hasBeenInit(P)) and
     not(query(hasInitPay(X, P))) then
       insertFact(hasBeenAuth(P))
```

Suppose user A attempts to initiate a payment P. Then the resource guard executes `initPay(A, P)`, which issues a query to the policy evaluator to check whether A is a manager, according to the local policy and the current authorization state. Additionally, it is also checked that the payment has not already been initiated. If successful, the facts `hasBeenInit(P)` (payment P has been initiated) and `hasInitPay(A, P)` (A has initiated payment P) are inserted into the authorization state. Subsequently, user B attempts to authorize the same payment P, causing the resource guard to execute `authPay(B, P)`. The request is authorized if B can be proved to be a manager, and if the payment can be proved to have been initiated by someone other than B. If successful, the fact `hasBeenAuth(P)` is inserted into the state, indicating that the payment is authorized to be executed.

This example highlights a deficiency in current authorization languages: they cannot express updates of the authorization state, as required for many role-

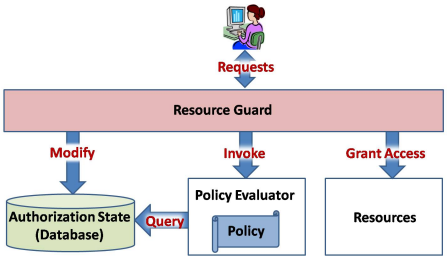

Figure 1: Model of a policy-based authorization system

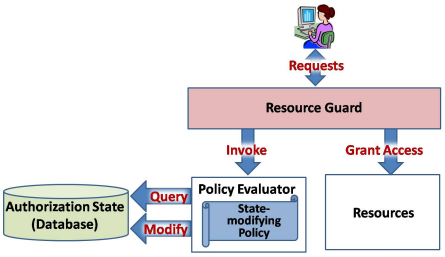

Figure 2: Factoring out the state manipulations.

based, separation-of-duties and other history-dependent policies. Instead, updates have to be hard-coded into the resource guard, which leads to maintainability problems. Moreover, as the state changes happen outside the policy and are written in a Turing-complete language, rigorous analysis is difficult.

In this paper, we address this problem by introducing *SMP*, a logic for specifying policies with state-modifying user requests. State changes are thus factored out of the resource guard, as in Figure 2. For example, in our approach the above scenario could be modelled by the following two policy rules:

$$\begin{aligned}\mathsf{initPay}(X,P) \leftarrow\ & \mathsf{isMgr}(X) \wedge \neg\mathsf{hasBeenInit}(P) \otimes \\ & +\mathsf{hasBeenInit}(P) \otimes +\mathsf{hasInitPay}(X,P)\\ \mathsf{authPay}(X,P) \leftarrow\ & \mathsf{isMgr}(X) \wedge \mathsf{hasBeenInit}(P) \wedge \neg\mathsf{hasInitPay}(X,P) \otimes \\ & +\mathsf{hasBeenAuth}(P)\end{aligned}$$

Intuitively, $+P$ specifies an insertion of a fact P into the authorization state, and the connective $\otimes$ expresses a serial execution of state changes.

Our logic is not intended to replace existing authorization languages. Rather, it should demonstrate how existing languages can be orthogonally extended to support state changes.

The complexity of state-modifying policies calls for analysis tools that support policy authors in debugging policies. We present a proof system that describes all possible sequences of access requests which yield a certain outcome. This proof system is proved sound and correct with respect to the logic, and we describe a sound and complete algorithm for finding minimal sequences in the propositional case.

The remainder of this paper is structured as follows. In §2 we introduce *SMP* and motivate its syntax and semantics. In §3 we present a proof system for reasoning about sequences of access requests, and describe the related algorithm. In §4 we present a larger case study for using state-modifying authorization policies. We conclude in §5 with a discussion on related works and future work. Full proofs are included in Appendix A.

2 A Logic for State-Modifying Policies

This section introduces the syntax and semantics of *SMP*. The logic can express updates to the authorization state: facts may be inserted into or removed from the state as a result of evaluating an access request. Furthermore, it can specify

non-monotonic prerequisite conditions on the authorization state: certain facts may be required to hold or not to hold in the current state.

SMP is based on Datalog, but extends it with statements for state modification, henceforth called *effects*, and a simple form of negation. Datalog [17, 2] is a declarative language for deductive databases (essentially Horn clauses without function symbols). It is well-suited for specifying authorization policies, as these can often be expressed as statements of the form "if ⟨*condition*⟩ then ⟨*permission*⟩". Consequently, Datalog has been chosen as the semantic basis for many authorization languages: Cassandra [8], Binder [23], RT [37, 36], and Delegation Logic [35] are semantically defined by a translation to Datalog or constrained Datalog; SecPAL [6], and significant fragments of XACML [44], XrML [20] and Lithium [27] can also be translated into Datalog or constrained Datalog. We assume a denumerable set of variables $\mathcal{X}$ and a first-order signature with constants $\mathcal{C}$ and disjoint sets of predicate symbols, *extensional* ($\mathcal{Q}_{ext}$) and *intensional* ($\mathcal{Q}_{int}$) ones, as in standard Datalog. In addition, we have a third set of so-called *command* predicate symbols $\mathcal{Q}_{cmd}$, intended to represent access requests. As usual, (extensional, intensional and command) *atoms* are formed by applying predicate symbols to ordered lists of constants or variables. A *literal* is either an atom P or a negated atom $\neg P$.

The extensional predicates are defined by an *extensional database* (EDB), a set of ground extensional atoms (*facts*). The validity of an extensional literal can thus be checked simply by inspecting the database. In the context of an authorization system, the database contains environmental facts that are relevant for authorization, e.g. $\mathsf{hasInitPay}(\mathtt{Alan}, \mathtt{P123})$, $\mathsf{isUser}(\mathtt{Alan})$ and $\mathsf{dateOfBirth}(\mathtt{Alan}, \mathtt{23/06/1912})$. As we shall see later, facts may be inserted or removed from the database after evaluating an access request. The database thus constitutes the transient state of an authorization system; hence we also call it the *authorization state*.

Definition 2.1 An *authorization state* **B** is a finite set of *facts* (ground extensional atoms).

Intensional predicate symbols are defined by *rules*. A rule consists of an intensional atom P_{int} (the *head*) and a conditional *body*, a (possibly empty) conjunction of extensional or intensional literals: $P_{int} \leftarrow L_1 \wedge \ldots \wedge L_m$. For example, the intensional predicate symbol isMgr may be defined by a rule specifying that X is a manager if X is a user and if someone has registered X as a manager:

$$\mathsf{isMgr}(X) \leftarrow \mathsf{isUser}(X) \wedge \mathsf{hasRegisteredAsMgr}(Y, X)$$

Negation is restricted to extensional atoms: $\neg P_{ext}$ holds if P_{ext} is not in the current authorization state. This is the simplest form of negation that is sufficient for our purposes.

Access requests, or *commands*, are defined by *command rules*, i.e. rules with a command atom as head. In addition to the body, a command rule contains a (possibly empty) sequence of *effects* on the authorization state which are executed if all conditions in the body have been satisfied. An effect K is either an insertion ($+P_{ext}$) of a fact P_{ext} into the authorization state or a removal ($-P_{ext}$). Effects are sequentially composed by the operator $\otimes$ from Transaction Logic. A command rule is thus of the form $P_{cmd} \leftarrow L_1 \wedge \ldots \wedge L_m \otimes K_1 \otimes \ldots \otimes K_n$.

Term	t	::=	$X \mid a$	where $X \in \mathcal{X}, a \in \mathcal{C}$
Atom	P_τ	::=	$p(t_1, \ldots, t_k)$	where $p \in \mathcal{Q}_\tau, \ \tau \in \{ext, int, cmd\}$
Literal	L	::=	$P_{int} \mid P_{ext} \mid \neg P_{ext}$	
Effect	E	::=	$+P_{ext} \mid -P_{ext}$	
Rule	Rl	::=	$P_{int} \leftarrow L_1 \wedge \ldots \wedge L_m$	where $m \geq 0$
		$\mid$	$P_{cmd} \leftarrow L_1 \wedge \ldots \wedge L_m \otimes E_1 \otimes \ldots \otimes E_n$	where $m, n \geq 0$

Table 1: Syntax of *SMP*

We sometimes write $\vec{L}$ to abbreviate $L_1 \wedge \ldots \wedge L_m$, and $\tilde{K}$ for $K_1 \otimes \ldots \otimes K_n$. Table 1 shows the complete *SMP* syntax.

Definition 2.2 (Well-formed Policy) A *policy* is a finite set of rules.

A rule is *well-formed* if all variables of its effects also occur in the head; furthermore, if effects $+P_1$ and $-P_2$ occur in the same rule, then P_1 and P_2 are non-unifiable. A policy $\mathcal{P}$ is *well-formed* iff all rules in $\mathcal{P}$ are well-formed, and furthermore, whenever two ground instances of rules in $\mathcal{P}$ have the same head, their effects are identical.

The well-formedness conditions ensure that every ground command uniquely determines a sequence of ground effects, and furthermore, that the order of the sequence is irrelevant.

As command literals cannot occur inside a body, effects can only occur at the top level: they are effectively decoupled from recursion. Recursive effects would not only be harder to compute and to comprehend, but worse, they would be non-deterministic.

Conceptually, the command predicates provide an interface between the resource guard and the policy, and are indeed the only predicates exported externally. Upon an access request (e.g. initiating a payment), the resource guard issues a query consisting of an instantiated command predicate (e.g. $\mathsf{initPay}(\texttt{Alan}, \texttt{P123})$). Depending on the policy and the authorization state, the query could succeed or fail. If it succeeds, the authorization state is updated automatically (e.g. by inserting $\mathsf{hasInitPay}(\texttt{Alan}, \texttt{P123})$ into the authorization state) and the access request is authorized.

Example 2.3 We would like to write a policy for an online movie store. Informally, we would like to express that users can buy a movie online, and are then allowed to play it twice. This is expressed by the following transaction policy:

$$\begin{aligned}
\mathsf{buy}(X, M) &\leftarrow +\mathsf{bought}(X, M) \\
\mathsf{play1}(X, M) &\leftarrow \mathsf{bought}(X, M) \wedge \neg\mathsf{played1}(X, M) \otimes +\mathsf{played1}(X, M) \\
\mathsf{play2}(X, M) &\leftarrow \mathsf{played1}(X, M) \wedge \neg\mathsf{played2}(X, M) \otimes +\mathsf{played2}(X, M)
\end{aligned}$$

The semantics of *SMP* is formalized by modelling it as a fragment of Transaction Logic [15]. Transaction Logic is a general framework that incorporates database updates and transactions into first order logic. A Herbrand-style model theory of Transaction Logic is presented in detail in [15]. Based on this semantics, we define an entailment relation $\tilde{\mathbf{B}} \models_{\mathcal{P}} \phi$ between a sequence of authorization states $\tilde{\mathbf{B}}$ and a formula ϕ, in the context of a well-formed policy $\mathcal{P}$. This relation is presented in Table 2.

(pos)	$\mathbf{B} \vDash_{\mathcal{P}} P_{ext}$	iff	$P \in \mathbf{B}$
(neg)	$\mathbf{B} \vDash_{\mathcal{P}} \neg P_{ext}$	iff	$P \notin \mathbf{B}$
(and)	$\tilde{\mathbf{B}} \vDash_{\mathcal{P}} \phi \wedge \psi$	iff	$\tilde{\mathbf{B}} \vDash_{\mathcal{P}} \phi$ and $\tilde{\mathbf{B}} \vDash_{\mathcal{P}} \psi$
(seq)	$\mathbf{B}_1, ..., \mathbf{B}_k \vDash_{\mathcal{P}} \phi \otimes \psi$	iff	$\mathbf{B}_1, ..., \mathbf{B}_i \vDash_{\mathcal{P}} \phi$ and $\mathbf{B}_i, ..., \mathbf{B}_k \vDash_{\mathcal{P}} \psi$ for some $i \in \{1, ..., k\}$
(plus)	$\mathbf{B}_1, \mathbf{B}_2 \vDash_{\mathcal{P}} +P$	iff	$\mathbf{B}_2 = \mathbf{B}_1 \cup \{P\}$
(min)	$\mathbf{B}_1, \mathbf{B}_2 \vDash_{\mathcal{P}} -P$	iff	$\mathbf{B}_2 = \mathbf{B}_1 \setminus \{P\}$
(impl)	$\tilde{\mathbf{B}} \vDash_{\mathcal{P}} Q$ where Q is not extensional	iff	$Q \leftarrow \phi$ is a ground instantiation of a rule in $\mathcal{P}$ and $\tilde{\mathbf{B}} \vDash_{\mathcal{P}} \phi$

Table 2: *SMP* semantics

Intuitively, $\mathbf{B}_0, \ldots, \mathbf{B}_n \vDash_{\mathcal{P}} \phi$ means that the goal ϕ can be derived in the context of policy $\mathcal{P}$, starting from an initial authorization state $\mathbf{B}_0$. The evaluation of ϕ leads (via the intermediate states) to a final state $\mathbf{B}_n$. Rule (pos) and (neg) state that extensional literals are checked by inspecting the authorization state. This involves no effects, so the initial and final state are both identical. Rule (and) states that a database sequence $\tilde{\mathbf{B}}$ entails the conjunction of two formulae ϕ and ψ iff each of them is independently entailed by the same $\tilde{\mathbf{B}}$. In contrast, a serial composition $\phi \otimes \psi$ is entailed by $\mathbf{B}_1, ..., \mathbf{B}_k$ iff ϕ can be derived starting from $\mathbf{B}_1$ and ending in some intermediate state $\mathbf{B}_i$, and ψ can be derived starting from $\mathbf{B}_i$ and ending in $\mathbf{B}_k$. Rules (plus) and (min) straightforwardly describe the insertion and deletion of facts. Finally, rule (impl) defines the derivation for non-extensional literals.

For example, starting with a database $\{q(1)\}$ the evaluation of formula $+p(0)$ leads to database $\{q(1), p(0)\}$ (using (plus)):

$$\{q(1)\}, \{q(1), p(0)\} \vDash_{\mathcal{P}} +p(0)$$

Likewise one can argue with (min) that the following holds:

$$\{q(1), p(0)\}, \{p(0)\} \vDash_{\mathcal{P}} -q(1)$$

Taking both results together gives $\{q(1)\}, \{q(1), p(0)\}, \{p(0)\} \vDash_{\mathcal{P}} p(0) \otimes -q(1)$by rule (seq).

3 Reasoning about User Requests

Authorization policies are hard to get right. It is even harder in the case of state-modifying policies. Authorization decisions may depend on access requests in the past, and in general, completing a task involves executing a command sequence (i.e. a sequence of access requests).

The increased complexity of state-modifying policies calls for proof techniques and tools to support the policy writer in establishing the correctness of a policy. For instance, in Example 2.3 a policy writer might like to determine the answers to the questions "Is there a command sequence that enables a user to play a movie without purchase?" and "Can a movie be played at least twice after purchase?". In this section, we develop a sound and complete proof system

which determines the command sequences which yield a certain target authorization state, and also motivates an algorithm for deriving a finite abstraction of all such possible sequences.

3.1 State Constraints

In analyzing state-modifying policies, we are usually not interested in whether a specific authorization state is reachable. Rather, we wish to reason about the reachability of a family of target states that all satisfy some constraint; for example, all states containing some ground instantiation of the atom $\mathsf{played1}(X, M)$ but not the corresponding instantiation of $\mathsf{bought}(X, M)$. This would capture all states in which a movie has been played without purchase.

The following definition allows us to specify classes of authorization states by stating which atoms it must or must not contain.

Definition 3.1 (State Constraints) A *state constraint* D is a set of extensional literals. The notation D^+ (and D^-, respectively) is used to refer to the set of atoms derived from the positive (negative) literals in D.

A state $\mathbf{B}$ *satisfies* a ground state constraint D iff $D^+ \subseteq \mathbf{B}$ and $D^- \cap \mathbf{B} = \emptyset$. A state constraint D is *consistent* iff $D^+ \cap D^- = \emptyset$.

For example, the state $\mathbf{B}_0 = \{p(0), q(1), r(2)\}$ satisfies the state constraint $D = \{p(0), q(1), \neg r(1)\}$ because $D^+ = \{p(0), q(1)\} \subseteq \mathbf{B}_0$ and $D^- = \{r(1)\} \cap \mathbf{B}_0 = \emptyset$. In this way D characterizes a family of states, namely the set of states that satisfy D. The state constraint $E = \{p(0), q(1), \neg p(0)\}$ is not consistent because $E^+ \cap E^- = \{p(0)\} \neq \emptyset$.

Example 3.2 Continuing Example 2.3, a policy writer might be interested in establishing that there is no command sequence which leads to a state satisfying the constraint $\{\neg\mathsf{bought}(X, M), \mathsf{played1}(X, M)\}$; we will establish later that there exists no such sequence. On the other hand, we will be able to determine that the command sequence $\mathsf{buy}(X, M) \otimes \mathsf{play1}(X, M)$ leads from an arbitrary state to a state satisfying $\{\mathsf{bought}(X, M), \mathsf{played1}(X, M)\}$.

Composing state constraints allows us to specify their transformation when a command is performed on them.

Definition 3.3 (Composition Operator) The operator $\circ$ defines a notion of composition for state constraints E and D:

$$\begin{aligned}(E \circ D)^+ &= (D^+ \cup E^+) \setminus E^- \\ (E \circ D)^- &= (D^- \cup E^-) \setminus E^+\end{aligned}$$

Intuitively, E can be interpreted as a set of effects, where positive literals in E are interpreted as insertions into D, and negative ones as removals. Then $E \circ D$ can be interpreted as the result of applying the effects in E to D. More precisely: it is the strongest constraint that is satisfied by states obtained by applying E to states satisfying D. For example, let $D = \{p(0), q(1), \neg r(1)\}$ as before and $E = \{\neg p(0), r(1), s(2)\}$ then $E \circ D = \{q(1), r(1), s(2), \neg p(0)\}$. This will enable us later to formulate the strongest postcondition when applying some command Q with effect E to a state constraint D. The following lemma is immediate:

Lemma 3.4 *If D is consistent, then $E \circ D$ is consistent for any state constraint E.*

Proof. Assume $E \circ D$ is inconsistent, i.e. $Q \in (E \circ D)^+ \cap (E \circ D)^-$. Then $Q \in (D^+ \cup E^+) \setminus E^-$ and $Q \in (D^- \cup E^-) \setminus E^+$. From the first relation we have $Q \notin E^-$, and from the second $Q \notin E^+$. Hence $Q \in D^+ \cap D^-$, a contradiction to the consistency of D. □

Recall that the well-formedness conditions of policies ensure that every ground command atom is uniquely associated with a sequence of its effects. We can now define a mapping *eff* that maps a ground command atom to a state constraint that is equivalent to its effects:

Definition 3.5 (Specification of *eff*) Let Q be a ground command atom, and let $\tilde{K}$ be the (possibly empty) sequence of ground effects uniquely determined by it (in the context of a well-formed policy). Then $\mathit{eff}(Q)$ is defined as the state constraint

$$\mathit{eff}(Q) = \{P \ : \ +P \in \tilde{K}\} \cup \{\neg P \ : \ -P \in \tilde{K}\}$$

3.2 Preconditions and Effects

We wish to relate command sequences of the form $\tilde{S} = Q_1 \otimes \ldots \otimes Q_k$ to state constraints D that represent initial authorization states from which $\tilde{S}$ can successfully execute, and to state constraints D' that represent authorization states after executing the sequence. In other words, for all authorization states $\mathbf{B}$ and $\mathbf{B}'$ such that $\mathbf{B}$ satisfies D and $\mathbf{B}'$ satisfies D', $\mathbf{B}, \ldots, \mathbf{B}' \vDash_{\mathcal{P}} \tilde{S}$ should hold. We start by considering the case $k = 1$, where the command sequence consists of just one single command. The proof system developed in §3.3 then deals with command sequences of arbitrary lengths and captures exactly the above relationship between D, $\tilde{S}$ and D'.

In order to decide from which initial states a command successfully executes, we need to compute its *preconditions* as a state constraint. To determine the target states, we need to find its *effects*. For example, suppose a command predicate q is defined by a single policy rule (where p, r and s are all extensional):

$$q(X, Y) \leftarrow \neg p(X) \wedge r(Z) \otimes +p(X) \otimes -s(Y),$$

Then in order for the command $q(0, 1)$ to execute, the starting authorization state must satisfy the state constraint $\{\neg p(0), r(Z)\}$ containing the preconditions of the command. Likewise, due to the effects of the command, the end state contains $p(0)$ but cannot contain $s(1)$. Moreover, the atom $r(Z)$ is left untouched by the command, hence the end state satisfies $\{r(Z), p(0), \neg s(1)\}$.

The effects are easily obtained using the function *eff* defined in Definition 3.5 that reinterprets the sequence of effects as a state constraint (insertions $+P$ corresponds to positive literals P, and removals $-P$ to negative literals $\neg P$).

In the following, we define a function *pre* which yields the information about the (extensional) preconditions of a ground atom P. In the case where P is entirely defined by rules whose body literals are all extensional, it is easy to determine the preconditions. For instance, in the rule for $\mathsf{play1}(X, M)$ of Example 2.3 both bought and played1 are extensional predicate symbols. Furthermore, play1

is defined by only one single rule. Therefore the preconditions for executing $\mathsf{play1}(X, M)$ are given by a set containing only one state constraint:

$$pre(\mathsf{play1}(X, M)) = \{\{\mathsf{bought}(X, M) \wedge \neg \mathsf{played1}(X, M)\}\}$$

In general however, command predicates may be defined by multiple rules, and the body literals of those rules may be intensional, as in the following example.

Example 3.6 We modify Example 2.3 by removing the $\mathsf{buy}(X, M)$ rule, and adding the following two rules (where bought is now an intensional predicate):

$$\begin{array}{l} \mathsf{bought}(X, M) \leftarrow \mathsf{bank}(Y), \mathsf{cardPayment}(X, Y, M) \\ \mathsf{bought}(X, M) \leftarrow \mathsf{freeTrial}(X) \end{array}$$

These rules express that customer X has bought a movie M if he either paid for it using his credit card issued by a bank Y, or has signed up for a free trial offer of the movie store. In this case, $pre(\mathsf{play1}(X, M))$ is a set containing the two preconditions under which the command $\mathsf{play1}$ can execute:

$$\begin{array}{l} F_1 = \{\mathsf{bank}(Y), \mathsf{cardPayment}(X, Y, M)\} \\ F_2 = \{\mathsf{freeTrial}(X)\} \end{array}$$

To specify *pre* formally, note that the bodies of the rules defining P determine its preconditions; in particular, the effects have no influence. We therefore introduce the following operation on policies that erases the effects:

Definition 3.7 Let $\mathcal{P}$ be a well-formed policy, and

$$Rl \equiv P_{cmd} \leftarrow L_1 \wedge \ldots \wedge L_m \otimes E_1 \otimes \ldots \otimes E_n$$

be one of its rules. Then $Rl^\star$ denotes the rule $P_{cmd} \leftarrow L_1 \wedge \ldots \wedge L_m$, where all effects have been removed. We write $\mathcal{P}^\star$ to denote the policy obtained from $\mathcal{P}$ by applying the $\star$-operation to all its rules.

We are now ready to specify *pre*:

Definition 3.8 (Specification of *pre*) The function *pre* is a mapping between ground atoms and sets of state constraints. It is defined by the following axiom which holds for any state $\mathbf{B}$ and ground atom P:

$$\exists\, F, \theta \;:\; F \in pre(P) \wedge \mathbf{B} \text{ satisfies } F\theta \quad \text{iff} \quad \mathbf{B} \vDash_{\mathcal{P}^\star} P$$

Read in the "only if"-direction, the axiom states that every authorization state satisfying a ground instantiation of a state constraint in $pre(P)$ indeed suffices as a precondition of P. In the "if"-direction, the axiom states that every extensional ground precondition is subsumed by some state constraint in $pre(P)$.

Note that literals in preconditions can also contain free variables, such as Y in the first rule of Example 3.6. These variables are instantiated by some substitution θ.

The function *pre* can be computed using *abduction* [48], a dual to *deduction*, where explanatory facts, e.g. F_1 in the example, are inferred from a desired result, e.g. $\mathsf{bought}(X, M)$, using some general (effect-free) logic program, $\mathcal{P}^\star$. Abduction algorithms for logic programs are documented in [32].

3.3 Proof System

Based on the notion of state constraints and the functions *pre* and *eff* defined above, we now present a Hoare-style proof system in which we can reason about pre- and postconditions of general command sequences. The proof system allows the inference of triples of the form $\{D_1\}\,\tilde{S}\,\{D_2\}$. Intuitively, this holds if $\tilde{S}$ can be executed from any authorization state satisfying D_1, and terminates in a state satisfying D_2.

Definition 3.9 (Proof System)

$$\text{(eps)}\ \frac{G \text{ is ground and consistent}}{\{G\}\,\varepsilon\,\{G\}} \qquad \text{(seq)}\ \frac{\{G_1\}\,\tilde{S}\,\{H\} \qquad \{H\}\,P\,\{G_2\}}{\{G_1\}\,\tilde{S} \otimes P\,\{G_2\}}$$

$$\text{(cmd)}\ \frac{\begin{array}{c} G_1 \supseteq F\theta \qquad F \in \mathit{pre}(P) \qquad \mathit{eff}(P) \circ G_1 \supseteq G_2 \\ G_1 \text{ is ground and consistent} \qquad P \text{ is ground} \end{array}}{\{G_1\}\,P\,\{G_2\}}$$

The empty command ε trivially transforms any ground and consistent state constraint G into itself, according to rule (eps). The rule (cmd) relates commands to their preconditions and postconditions based on *pre* and *eff*, but also incorporates precondition strengthening and postcondition weakening: a command P transforms G_1 into G_2 if the following holds: there exists a substitution θ such that the free variables in a precondition $F \in \mathit{pre}(P)$ can be resolved with $G_1 \supseteq F\theta$, i.e. $F\theta$ is a weakest precondition for the execution of P and G_1 is strong enough to satisfy this preconditon; furthermore, G_2 has to be contained in $\mathit{eff}(P) \circ G_1$, i.e. G_2 is weaker than the *strongest postcondition* obtained by computing $\mathit{eff}(P) \circ G_1$. Finally, rule (seq) states that a sequence $\tilde{S}$ and a command P can be executed sequentially, if the postcondition of the sequence equals the precondition of the command.

The correctness of the proof system is ensured by the following soundness and completeness results, which relate it back to *SMP* semantics.

Theorem 3.10 (Soundness) *For all state constraints G_1, G_2, command sequences $\tilde{S}$ and authorization states $\mathbf{B}_1$ the following holds. If $\{G_1\}\,\tilde{S}\,\{G_2\}$ and $\mathbf{B}_1$ satisfies G_1, then there exists a $\mathbf{B}_2$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{S}$ and $\mathbf{B}_2$ satisfies G_2.*

Theorem 3.11 (Completeness) *For all authorization states $\mathbf{B}_1, \mathbf{B}_2$, command sequences $\tilde{S}$ and state constrains G_2 the following holds. If $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{S}$ and $\mathbf{B}_2$ satisfies G_2, then there exists G_1 such that $\{G_1\}\,\tilde{S}\,\{G_2\}$ and $\mathbf{B}_1$ satisfies G_1.*

3.4 Tabling Algorithm

The proof system gives rise to an algorithm for computing an abstraction of the set of all sequences which, given an authorization state $\mathbf{B}_0$ to start with, lead to states that are guaranteed to satisfy a target state constraint D. $\mathbf{B}_0$, D, and a well-formed program $\mathcal{P}$ are the only inputs to the algorithm. We present this algorithm for the propositional case, i.e. the policy and the state constraints are variable-free. A prototype implementation of the algorithm has been used to confirm the results from the examples presented in this section.

$$
\begin{array}{lll}
\text{(root)} & (\{\langle D_2\rangle\} \uplus \mathcal{N}, \mathit{Ans}, \mathit{Wait}) \rightarrow_{\mathcal{P}} (\mathcal{N} \cup \mathcal{N}_1 \cup \mathcal{N}_2, \mathit{Ans}, \mathit{Wait}) & \\
& \text{if} \quad \mathcal{N}_1 = \{\mathsf{goal}\langle D_1; Q; D_2\rangle & : \quad Q \text{ is some command in } \mathcal{P}, \text{ and } E \stackrel{\text{def}}{=} \mathit{eff}(Q) \\
& & \quad\; E \cap D_2 \neq \emptyset \\
& & \quad\; F \in \mathit{pre}(Q) \\
& & \quad\; D_1 \stackrel{\text{def}}{=} (D_2 \setminus E) \cup F \text{ is consistent} \\
& & \quad\; {D_2}^+ \cap E^- = \emptyset \text{ and } {D_2}^- \cap E^+ = \emptyset\} \\
& \quad\; \mathcal{N}_2 = \{\mathsf{ans}\langle D_2; \varepsilon; D_2\rangle & : \quad \mathbf{B}_0 \text{ satisfies } D_2\}
\end{array}
$$

$$
\begin{array}{ll}
\text{(ans)} & (\{\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle\} \uplus \mathcal{N}, \mathit{Ans}, \mathit{Wait}) \rightarrow_{\mathcal{P}} (\mathcal{N} \cup \mathcal{N}', \mathit{Ans}', \mathit{Wait}) \\
& \text{if} \quad \nexists\, m \in \mathit{Ans}(D_2) \;:\; \mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \succeq m \\
& \quad\; \mathcal{N}' = \bigcup_{n' \in \mathit{Wait}(D_2)} \mathit{merge}(\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle, n') \\
& \quad\; \mathit{Ans}' = \mathit{Ans}[D_2 \mapsto \mathit{Ans}(D_2) \cup \{\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle\}]
\end{array}
$$

$$
\begin{array}{ll}
\text{(goal}_1\text{)} & (\{\mathsf{goal}\langle D_1; Q; D_2\rangle\} \uplus \mathcal{N}, \mathit{Ans}, \mathit{Wait}) \rightarrow_{\mathcal{P}} (\mathcal{N} \cup \mathcal{N}', \mathit{Ans}, \mathit{Wait}') \\
& \text{if} \quad D_1 \in \mathit{dom}(\mathit{Ans}) \\
& \quad\; \mathcal{N}' = \bigcup_{n' \in \mathit{Ans}(D_1)} \mathit{merge}(n', \mathsf{goal}\langle D_1; Q; D_2\rangle) \\
& \quad\; \mathit{Wait}' = \mathit{Wait}[D_1 \mapsto \mathit{Wait}(D_1) \cup \{\mathsf{goal}\langle D_1; Q; D_2\rangle\}]
\end{array}
$$

$$
\begin{array}{ll}
\text{(goal}_2\text{)} & (\{\mathsf{goal}\langle D_1; Q; D_2\rangle\} \uplus \mathcal{N}, \mathit{Ans}, \mathit{Wait}) \rightarrow_{\mathcal{P}} (\mathcal{N} \cup \{\langle D_1\rangle\}, \mathit{Ans}', \mathit{Wait}') \\
& \text{if} \quad D_1 \notin \mathit{dom}(\mathit{Ans}) \\
& \quad\; \mathit{Ans}' = \mathit{Ans}[D_1 \mapsto \emptyset] \\
& \quad\; \mathit{Wait}' = \mathit{Wait}[D_1 \mapsto \{\mathsf{goal}\langle D_1; Q; D_2\rangle\}]
\end{array}
$$

Table 3: Tabling algorithm

The algorithm uses a goal-oriented search that attempts to construct sequences backwards. To ensure completeness and termination, and to prune the search trees, intermediate results are cached in a table and are reused. The technique of tabling, or memoing [49, 24, 19, 50], has also been considered for policy evaluation [8, 6]. Our algorithm works on *nodes* that represent both answers and unsolved goals which arise during the computation.

Definition 3.12 (Nodes) A *root node* is of the form $\langle D\rangle$ where D is a state constraint. An *answer node* is denoted $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle$, where $\tilde{T}$ is a sequence and D_1 and D_2 are state constraints, called the *pre-* and *postcondition* of $\tilde{T}$, respectively. The same terms apply to a *goal node* $\mathsf{goal}\langle D_1; \tilde{T}; D_2\rangle$.

Intuitively, if $\langle D_2\rangle$ occurs in the node set the algorithm is working on, we are looking for command sequences which have D_2 as final state. Tables *Ans* and *Wait* are defined to map state constraints into sets of answer and goal nodes, respectively. If $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$, then $\tilde{T}$ is an answer to D_2, i.e. $\mathbf{B}_0$ satisfies D_1, and $\{D_1\}\,\tilde{T}\,\{D_2\}$. If $\mathsf{goal}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Wait}(D_1)$, then $\tilde{T}$ is not yet an answer to D_2, but it is waiting for new answers to D_1 which can be resolved with the goal to give a new answer to D_2.

We present the algorithm as a state transition system in Table 3. A state is given by the current node set, the answer table, and the waiting table together.

Definition 3.13 (State) A *state* is a triple $(\mathcal{N}, Ans, Wait)$ where $\mathcal{N}$ is a set of nodes, Ans is an answer table, and $Wait$ is a wait table.

For every consistent state constraint D, a state $(\{D\}, [D \mapsto \emptyset], [D \mapsto \emptyset])$ is an *initial state*. A state $\mathcal{S}$ is a *final state* iff there is no state $\mathcal{S}'$ such that $\mathcal{S} \rightarrow_{\mathcal{P}} \mathcal{S}'$.

When a new root $\langle D_2 \rangle$ is spawned, it is processed by transition (root). If $\mathbf{B}_0$ satisfies D_2, then a single answer node $\mathsf{ans}\langle D_2; \varepsilon; D_2 \rangle$ is produced (set $\mathcal{N}_2$), corresponding to rule (eps) of the proof system. Set $\mathcal{N}_1$ contains goal nodes, which are produced according to the following scheme which corresponds to rule (cmd): for all commands Q, it is tested whether they can contribute at least one effect which occurs in D_2, by checking $E \cap D_2 \neq \emptyset$. Then for all preconditions $F \in pre(Q)$, a constraint D_1 is computed, which is obtained by removing all effects of Q from D_2 and adding the precondition F, thus reasoning *backwards* to a state, starting from which Q can execute and yield D_2. D_1 has to be consistent, and so must be the union $D_2 \cup E$, which is ensured by two further side conditions.

Before explaining the remaining rules, we need two further definitions. The first one deals with deriving new answers from given answers and corresponding goals.

Definition 3.14 (merge) An answer node $n_1 \equiv \mathsf{ans}\langle D_1; \tilde{T}; D_2 \rangle$ and a goal node $n_2 \equiv \mathsf{goal}\langle E_1; Q; E_2 \rangle$ can be *merged* iff $D_2 = E_1$ and D_1, D_2, E_1, E_2 are all consistent. The resulting node is $n \equiv \mathsf{ans}\langle D_1; \tilde{T} \otimes Q; E_2 \rangle$, and we write $n = merge(n_1, n_2)$.

In general, given an initial state $\mathbf{B}_0$ and a target state constraint D, there are infinitely many correct command sequences that lead from $\mathbf{B}_0$ to D. However, most of these answers are redundant in the sense that there is a shorter one consisting of the same atoms, or there is one with the same length but involves a smaller number of different commands. It turns out that, even if the complete set of answers is infinite, the set of "minimal" answers is finite. The presented algorithm always terminates and is complete with respect to this finite set of answers. All other answers are then *subsumed* by one of the computed answers.

The following definition defines the subsumption relation between answers. In this definition, let *atoms* be a function such that $atoms(Q_1 \otimes \cdots \otimes Q_k) = \{Q_1, ..., Q_k\}$ and $atoms(\varepsilon) = \emptyset$.

Definition 3.15 (Subsumption for Answers) Let $n \equiv \mathsf{ans}\langle D_1; \tilde{S}; D_2 \rangle$ and $m \equiv \mathsf{ans}\langle E_1; \tilde{T}; E_2 \rangle$. Then, n is said to be *subsumed by* m, written $n \succeq m$, iff

$$D_1 \supseteq E_1 \ \wedge \ D_2 = E_2 \ \wedge \ |\tilde{S}| \geq |\tilde{T}| \ \wedge \ atoms(\tilde{S}) \supseteq atoms(\tilde{T})$$

The idea is that node n is subsumed by m if m is a "better" answer: that is, n's precondition is at least as strong as m's, its postcondition is equal to m's, and its command sequence is at least as long as m's and involves the same (or more) atoms.

Answer nodes are processed by rule (ans). The rule executes if the answer node in question is not subsumed by another one already in $Ans(D_2)$. Then the node is added to the answer table, and it is merged with all waiting goals in $Wait(D_2)$. Goal nodes are processed by either (goal_1) or (goal_2), according to whether or not their precondition D_1 has been spawned before. If not, (goal_2)

ensures that $\langle D_1 \rangle$ is spawned and that the answer and waiting tables are properly initialized for this new goal. Otherwise, (goal_1) prescribes that the goal node is merged with all answer nodes that might be already available for D_1, and that the waiting table is updated.

Example 3.16 We modify Example 2.3 in the following way: customers may extend the subscription for a certain movie after having played it already twice. For the sake of demonstrating the algorithm, we first replace all atoms with purely propositional ones (i.e. reasoning about a single movie and customer). The idea for the modified example then is that the tokens bought, played1, and played2 are removed when they are no longer used, enabling the customer to restart the transaction by buying the same movie again.

$$\begin{array}{ll} \mathsf{buy} & \leftarrow +\mathsf{bought} \otimes -\mathsf{played1} \\ \mathsf{play1} & \leftarrow \mathsf{bought}, \neg\mathsf{played1} \otimes +\mathsf{played1} \otimes -\mathsf{played2} \\ \mathsf{play2} & \leftarrow \mathsf{played1}, \neg\mathsf{played2} \otimes +\mathsf{played2} \otimes -\mathsf{bought} \end{array}$$

Assume now we start with an empty state $\mathbf{B}_0 = \emptyset$ and want to find sequences leading to a state $D_2 = \{\mathsf{played1}\}$, i.e. the customer has played the movie once. Starting from the initial state with $\langle D_2 \rangle$ in the node set, rule (root) will select only command play1, because it is the only one where $\mathit{eff}(\mathsf{play1}) \cap D_2 = \{\mathsf{played1}\} \neq \emptyset$. The computation of D_1 then yields

$$D_1 = (D_2 \setminus \{\mathsf{played1}, \neg\mathsf{played2}\}) \cup \{\mathsf{bought}, \neg\mathsf{played1}\} = \{\mathsf{bought}, \neg\mathsf{played1}\}$$

which is consistent, and the remaining checks work out as well. Thus a goal node is produced and inserted into the waiting table in step (goal_2)

$$\mathsf{goal}\langle\{\mathsf{bought}, \neg\mathsf{played1}\}; \mathsf{play1}; \{\mathsf{played1}\}\rangle \in \mathit{Wait}(D_1)$$

where also $\langle D_1 \rangle$ is spawned. The only applicable command for this rule is buy and after some steps we get $\mathsf{ans}\langle\{\}; \mathsf{buy}; \{\mathsf{bought}, \neg\mathsf{played1}\}\rangle \in \mathit{Ans}(D_1)$ Upon insertion of the answer into $\mathit{Ans}(D_1)$, the same answer is merged with the goal node above, yielding $\mathsf{ans}\langle\{\}; \mathsf{buy} \otimes \mathsf{play1}; \{\mathsf{played1}\}\rangle \in \mathit{Ans}(D_2)$,which is a new and, in this case, the only answer for D_2.

The correctness of the algorithm is stated in the following soundness and completeness theorems:

Theorem 3.17 (Soundness) *If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is reachable from some initial state, then the following implication holds for all $D_2 \in \mathit{dom}(\mathit{Ans})$: if $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$ and D_1 is consistent, then $\{D_1\}\,\tilde{T}\,\{D_2\}$ and $\mathbf{B}_0$ satisfies D_1.*

Intuitively, the soundness theorem states that every answer produced by the algorithm is indeed a correct answer with respect to the proof system. By soundness of the proof system, the algorithm is then also sound with respect to the *SMP* semantics.

Theorem 3.18 (Completeness) *If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is a final state reachable from some initial state, the following implication holds for all $D_2 \in \mathit{dom}(\mathit{Ans})$: if $\{G_1\}\,\tilde{S}\,\{G_2\}$, $G_2 \supseteq D_2$, and $\mathbf{B}_0$ satisfies G_1 then there exists $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$ such that $\mathbf{B}_0$ satisfies D_1, $|\tilde{S}| \geq |\tilde{T}|$, and $\mathit{atoms}(\tilde{S}) \supseteq \mathit{atoms}(\tilde{T})$.*

The completeness theorem states that if a sequence is a correct answer with respect to the proof system, then the algorithm produces an answer that is at least as good, in the sense that it may be shorter or involve a smaller number of different commands.

Completeness and termination of the algorithm crucially depend on a general compactness property of the subsumption relation:

Lemma 3.19 (Answer compactness) *Every set of answer nodes $\mathcal{N}$ in which only finitely many predicate symbols occur has a finite subset $\mathcal{N}_{fin} \subseteq \mathcal{N}$ such that:*

$$\forall n \in \mathcal{N}\ \exists m \in \mathcal{N}_{fin} : n \succeq m$$

Finally, the following theorem proves that the algorithm always terminates.

Theorem 3.20 (Termination) *All transition paths starting from an initial state are of finite length.*

4 Case Study

This section presents a larger, non-trivial example policy from the area of electronic health care. Community-wide Electronic Health Record (EHR) systems are being developed in Europe, the United States, Canada and Australia to provide "cradle-to-grave" summaries of patients' records, linking clinical information across the entire health system [21]. While the potential benefits of such systems are huge, there is also growing concern that patient confidentiality may be put at risk. The regulations governing access to patient-identifiable data are complex and change frequently. EHRs are thus an application area that require flexible authorization specification and enforcement. An authorization policy comprising over 350 rules was published in [4, 5], based on specifications of the EHR project in the United Kingdom [43]. Many of these rules require updates of the authorization state. In the following, we show in a number of examples how these updates can be made explicit with *SMP*. The policies shown here are simplified for the purpose of illustration. Moreover, *SMP* has been designed to be a minimal language focusing solely on state manipulations; in particular, *SMP* does not deal with constraints, parametrized roles and actions, trust management and delegation features, and other advanced language features such as negation, aggregation or quantification.

The policy is based on roles such as patient, clinician, or administrator. Users may be members of several roles, and may activate such roles within a session, in order to utilize the privileges associated with the roles. In the following example, a user may activate the patient role with the command activate. The command succeeds if the user is a member of that role (expressed by the predicate member), and as a result, a corresponding hasActivated fact is inserted into the authorization state. Patients can deactivate their own role if this fact is in the state. The deactivation entails the removal of the fact from the state.

$$\mathsf{activate}(X, \mathtt{Patient}) \leftarrow \mathsf{member}(X, \mathtt{Patient}) \otimes +\mathsf{hasActivated}(X, \mathtt{Patient})$$

$$\mathsf{deactivate}(X, \mathtt{Patient}) \leftarrow \mathsf{hasActivated}(X, \mathtt{Patient}) \otimes -\mathsf{hasActivated}(X, \mathtt{Patient})$$

For some groups of roles, the policy specifies a separation-of-duties constraint: users may be active in at most one of the roles at the same time. In the rule

below, the clinician role may only be activated if the user is not already active in the administrator role. We also have a symmetric rule for administrators where "`Clinician`" and "`Admin`" are permuted. The deactivation rules for clinicians and administrators are similar to the one for patients above.

$$\mathsf{activate}(X, \texttt{Clinician}) \leftarrow \mathsf{member}(X, \texttt{Clinician}) \wedge \neg\mathsf{hasActivated}(X, \texttt{Admin}) \otimes +\mathsf{hasActivated}(X, \texttt{Clinician})$$

In the examples above, role membership is a prerequisite for role activation. The following two clauses let administrators update the role membership assignment, using the commands register and unregister.

$$\begin{aligned} \mathsf{register}(X, U, R) \leftarrow &\quad \mathsf{hasActivated}(X, \texttt{Admin}) \otimes +\mathsf{member}(U, R) \\ \mathsf{unregister}(X, U, R) \leftarrow &\quad \mathsf{hasActivated}(X, \texttt{Admin}) \wedge \mathsf{registered}(U, R) \otimes \\ &\quad -\mathsf{member}(U, R) \otimes -\mathsf{hasActivated}(U, R) \end{aligned}$$

Permission assignments typically do not manipulate the authorization state, but often specify conditions that depend on the state. The following clause is an example of a deny-override policy: it allows a clinician X to read data from a patient P's health record if X has a so-called legitimate relationship with the patient, and if the patient has not explicitly concealed the record from X.

$$\mathsf{permitted}(X, \texttt{Read}, P) \leftarrow \mathsf{hasActivated}(X, \texttt{Clinician}) \wedge \mathsf{legitRelationship}(X, P) \wedge \neg\mathsf{denied}(P, X)$$

The command readEHR is executed when a user X attempts to read a patient P's EHR. The read access is stored in the authorization state for auditing purposes.

$$\mathsf{readEHR}(X, P) \leftarrow \mathsf{permitted}(X, \texttt{Read}, P) \otimes +\mathsf{hasReadEHR}(X, P)$$

Patients can conceal data from a clinician using the command denyAccess, and remove the concealment with removeDenyAccess. The two corresponding clauses insert (or remove) a denied fact from the authorization state.

$$\begin{aligned} \mathsf{denyAccess}(P, X) \leftarrow &\quad \mathsf{hasActivated}(P, \texttt{Patient}) \otimes +\mathsf{denied}(P, X) \\ \mathsf{removeDenyAccess}(P, X) \leftarrow &\quad \mathsf{hasActivated}(P, \texttt{Patient}) \wedge \mathsf{denied}(P, X) \otimes \\ &\quad -\mathsf{denied}(P, X) \end{aligned}$$

Access to patient data is conditioned on a *legitimate relationship* between the requester and the patient. There are several ways in which a legitimate relationship can be established, hence the corresponding predicate is an intensional one. For example, a patient has a legitimate relationship with herself. A patient can also appoint agents who can make decisions on her behalf: such agents have a legitimate relationship with the patient, too. The following clause specifies that a legitimate relationship exists between a clinician X and a patient P if the patient has explicitly consented to treatment:

$$\mathsf{legitRelationship}(X, P) \leftarrow \mathsf{hasConsented}(P, X, \texttt{Treatment})$$

The following clauses manage the updates for hasConsented facts. (The third predicate parameter specifies the kind of consent, as there are several different

kinds that are relevant within the policy.) Consent is here modelled as a two-step process: a clinician can request consent to treatment, and only then can the patient give consent.

$$\begin{aligned}
\mathsf{giveConsent}(P, X, \mathtt{Treatment}) \leftarrow\ & \mathsf{hasActivated}(P, \mathtt{Patient}) \wedge \\
& \mathsf{hasRequestedConsent}(X, P) \otimes \\
& +\mathsf{hasConsented}(P, X, \mathtt{Treatment}) \\
\mathsf{requestConsent}(X, P, \mathtt{Treatment}) \leftarrow\ & \mathsf{hasActivated}(X, \mathtt{Clinician}) \otimes \\
& +\mathsf{hasRequestedConsent}(X, P, \mathtt{Treatment})
\end{aligned}$$

Finally, patients can withdraw their consent to treatment. Similarly, clinicians can cancel the treatment. The command cancelTreatment removes both the consent request and the (possibly non-existing) patient consent fact from the authorization state.

$$\begin{aligned}
\mathsf{withdrawConsent}(P, X, \mathtt{Treatment}) \leftarrow\ & \mathsf{hasActivated}(P, \mathtt{Patient}) \wedge \\
& \mathsf{hasConsented}(P, X, \mathtt{Treatment}) \otimes \\
& -\mathsf{hasConsented}(P, X, \mathtt{Treatment}) \\
\mathsf{cancelTreatment}(X, P) \leftarrow\ & \mathsf{hasActivated}(X, \mathtt{Clinician}) \otimes \\
& -\mathsf{hasRequestedConsent}(X, P, \mathtt{Treatment}) \otimes \\
& -\mathsf{hasConsented}(P, X, \mathtt{Treatment})
\end{aligned}$$

Suppose Alice (A) attempts to read Bob's (B) records. Under which condition is she permitted to perform this action? The following triple states that the request is authorized if Alice has activated her clinician role, Bob has given consent to treatment and has not explicitly denied access.

$$\begin{aligned}
&\{\mathsf{hasActivated}(\mathtt{A}, \mathtt{Clinician}), \\
&\ \mathsf{hasConsented}(\mathtt{B}, \mathtt{A}, \mathtt{Treatment}), \neg\mathsf{denied}(\mathtt{B}, \mathtt{A})\} \\
&\quad\ \mathsf{readEHR}(\mathtt{A}, \mathtt{B}) \\
&\{\mathsf{hasReadEHR}(\mathtt{A}, \mathtt{B})\}
\end{aligned}$$

Now suppose we start in an authorization state in which we know that Charlie (C) is a member of the administrator role and has not activated the clinician role. Furthermore, Bob has not explicitly concealed his data from Charlie. We can then infer a command sequence that terminates in a state where Charlie has read Bob's EHR:

$$\begin{aligned}
&\{\mathsf{member}(\mathtt{C}, \mathtt{Admin}), \neg\mathsf{hasActivated}(\mathtt{C}, \mathtt{Clinician}), \\
&\ \neg\mathsf{denied}(\mathtt{B}, \mathtt{C})\} \\
&\quad \mathsf{activate}(\mathtt{C}, \mathtt{Admin}) \otimes \mathsf{register}(\mathtt{C}, \mathtt{C}, \mathtt{Clinician}) \otimes \\
&\quad \mathsf{register}(\mathtt{C}, \mathtt{B}, \mathtt{Patient}) \otimes \mathsf{activate}(\mathtt{B}, \mathtt{Patient}) \otimes \\
&\quad \mathsf{deactivate}(\mathtt{C}, \mathtt{Admin}) \otimes \mathsf{activate}(\mathtt{C}, \mathtt{Clinician}) \otimes \\
&\quad \mathsf{requestConsent}(\mathtt{C}, \mathtt{B}, \mathtt{Treatment}) \otimes \\
&\quad \mathsf{giveConsent}(\mathtt{B}, \mathtt{C}, \mathtt{Treatment}) \otimes \\
&\quad \mathsf{readEHR}(\mathtt{C}, \mathtt{B}) \\
&\{\mathsf{hasReadEHR}(\mathtt{C}, \mathtt{B})\}
\end{aligned}$$

5 Discussion

Related work *SMP* was designed to be a minimal language with the ability of querying and updating the authorization state. Its functionality can be added to

a wide range of existing authorization languages, but most easily to logic-based languages, e.g. [34, 1, 9, 35, 31, 23, 37, 36, 27, 51, 8, 6].

Cassandra [8, 7] is an authorization language that defines the actions of activating a role and deactivating a role, along with a transition system that updates the authorization state by inserting and removing corresponding "hasActivated" facts. Users can thus write state-dependent and implicitly state-manipulating policies, but this rather ad-hoc approach is inflexible and not very user-friendly. In a similar spirit, dynFAF [18] keeps track of the history of user requests by dynamically adding facts (with a time-stamp parameter) to the logic program. In dynFAF, facts are never removed; instead, permissions are signed, and permission revocation is modelled by adding a fact with a negative permission. In [29], a sub-language of Timed Default Concurrent Constraint Programming is used to specify dynamic policies. Their language, being almost a full-fledged procedural programming language, can express state changes triggered by both user requests as well as environmental changes. This high expressiveness comes at a price: policies are generally harder to analyze, and evaluation may not terminate.

Some languages such as Ponder [22] or XACML [44] support *obligation policies*. An obligation is a task to be executed after evaluating and enforcing an access request. Obligations are typically used for post-processing jobs such as auditing or for sending out notifications, but in principle an obligation could also be a call to an external function that updates the state. While this approach would move the effects from the hard-coded resource guard into the policy, it does not provide a precise semantics for the state changes.

The semantics of *SMP* is defined via a mapping to a fragment of Transaction Logic [15]. An inference system for reasoning about Transaction Logic is presented in [16]. There are a number of general-purpose logics supporting database updates, such as Insert-Delete-Modify Transactions (IDM) [3, 33], Dynamic Logic Programming (DLP) [41], Logical Data Language (LDL) [42], Transaction Datalog [14], U-Datalog [11, 12] or DatalogU [40]. The expressiveness of these general languages by far exceeds our needs, but they could all have been used as an alternative semantic basis for *SMP*.

Harrison et al. [28] present a model of an access control system where the entries in a (purely propositional) access matrix can be manipulated by a given set of commands. They show that the general problem of determining safety (i.e., that an unreliable principal cannot delegate a right to someone who does not already have it) is undecidable in their model, and identify decidable subcases.

Some work has been done on analyzing security properties in dynamic role-based systems, in the context of the role-based authorization language RT [39, 38] and Administrative RBAC (ARBAC) [47], where members of administrative roles can modify the role membership and privilege assignments [46]. Security properties in the context of SPKI/SDSI certificates are analyzed in [30] by model checking pushdown automata. [10] presents a Datalog-based logical framework for representing and reasoning about access control policies. Neither of the two papers deals with policies updating the authorization state.

Changes to the policy itself can obviously affect the set of actions that are permitted or denied. Margrave [26] is a tool that can compute the consequences of changes to an XACML policy. Pucella and Weissman [45] consider systems in which policies (not just the facts) can change between state transitions. They introduce a modal logic that can capture the dynamic nature of such systems

and prove its decidability in the propositional case. In [25], policies written in Datalog can refer to facts in the authorization state, as in our model. Events (such as access requests) can change the authorization state, and the changes are specified as a state machine whose transition labels are guarded by the policy. Security properties can then be analyzed by model checking formulas in first-order temporal logic.

Future Work We are currently working on extending the sequence algorithm to the non-propositional case. Allowing variables leads to a combinatorial explosion in search space and seems to require expensive operations such as set unification. Moreover, it is not possible to assume a finite domain, as the parameters of the effects are supplied by (external) access requests. Syntactic safety conditions have been used in authorization logics to deal with the complexities introduced by variables [27, 6]. We are investigating if this approach can be applied here as well.

Future work also includes investigating ways to further enhance the expressiveness of our logic. For example, it may be necessary to specify effects in the event of an access denial – the current logic only supports effects if a request is successful. The benchmark policy in [4] shows examples where the deactivation of a role is triggered by preceding deactivations (cascading revocation). Our current logic cannot express this and similar active policies. We plan to adopt an active declarative database language such as [11] as the semantic basis for an extended language.

Conclusion In this paper, we have introduced a logic that not only expresses authorization conditions but can also specify effects of access requests on the authorization state. The effects are specified explicitly in the language (as opposed to e.g. a state machine). Existing authorization languages can thus be easily extended to support effects. The logic is a non-monotonic extension of Datalog and has a formal semantics based on Transaction Logic. Examples of its applicability are shown in a case study on a policy for electronic health records. We have also presented an inference system for reasoning about sequences of user actions, and a sound and complete goal-oriented algorithm for computing minimal sequences (or proving their non-existence) in the propositional case.

References

[1] M. Abadi, M. Burrows, B. Lampson, and G. Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, 1993.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] S. Abiteboul and V. Vianu. Procedural and declarative database update languages. In *PODS '88: Proceedings of the seventh ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 240–250, New York, NY, USA, 1988. ACM Press.

[4] M. Y. Becker. Cassandra: Flexible trust management and its application to electronic health records. Technical Report UCAM-CL-TR-648, University of Cambridge, Computer Laboratory, 2005.

[5] M. Y. Becker. Information governance in NHS's NPfIT: A case for policy specification. *International Journal of Medical Informatics*, 2007. Article In Press.

[6] M. Y. Becker, C. Fournet, and A. D. Gordon. SecPAL: Design and semantics of a decentralized authorization language. In *20th IEEE Computer Security Foundations Symposium (CSF)*, 2007.

[7] M. Y. Becker and P. Sewell. Cassandra: distributed access control policies with tunable expressiveness. In *IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY)*, pages 159–168, 2004.

[8] M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *17th IEEE Computer Security Foundations Workshop (CSFW)*, pages 139–154, 2004.

[9] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Trans. Database Syst.*, 23(3), 1998.

[10] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6(1):71–127, 2003.

[11] E. Bertino, B. Catania, V. Gervasi, and A. Raffaet. Active-U-Datalog: Integrating active rules in a logical update language. In *ILPS '97: International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases*, pages 107–133, London, UK, 1998. Springer-Verlag.

[12] E. Bertino, B. Catania, and R. Gori. Enhancing the expressive power of the U-Datalog language. *Theory Pract. Log. Program.*, 1(1):105–122, 2001.

[13] M. Blaze, J. Feigenbaum, and A. D. Keromytis. The role of trust management in distributed systems security. In *Secure Internet Programming*, pages 185–210, 1999.

[14] A. J. Bonner. Workflow, transactions and datalog. In *PODS '99: Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 294–305, New York, NY, USA, 1999. ACM Press.

[15] A. J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133(2):205–265, 1994.

[16] A. J. Bonner and M. Kifer. Results on reasoning about updates in transaction logic. In *ILPS '97: International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases*, pages 166–196, London, UK, 1998. Springer-Verlag.

[17] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about Datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1):146–166, 1989.

[18] S. Chen, D. Wijesekera, and S. Jajodia. Incorporating dynamic constraints in the flexible authorization framework. In *9th European Symposium on Research Computer Security (ESORICS)*, pages 1–16, 2004.

[19] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, 1996.

[20] ContentGuard. *eXtensible rights Markup Language (XrML) 2.0 specification part II: core schema*, 2001. At www.xrml.org.

[21] A. Cornwall. Electronic health records: an international perspective. *Health Issues*, 73, 2002.

[22] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder policy specification language. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 18–38, London, UK, 2001. Springer-Verlag.

[23] J. DeTreville. Binder, a logic-based security language. In *IEEE Symposium on Security and Privacy*, pages 105–113, 2002.

[24] S. W. Dietrich. Extension tables: Memo relations in logic programming. In *Symposium on Logic Programming*, pages 264–272, 1987.

[25] D. J. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *Automated Reasoning, Third International Joint Conference, IJCAR 2006*, pages 632–646, 2006.

[26] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, 2005.

[27] J. Y. Halpern and V. Weissman. Using first-order logic to reason about policies. In *IEEE Computer Security Foundations Workshop*, pages 187–201, 2003.

[28] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[29] R. Jagadeesan, W. Marrero, C. Pitcher, and V. Saraswat. Timed constraint programming: a declarative approach to usage control. In *PPDP '05: 7th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*, pages 164–175, 2005.

[30] S. Jha and T. Reps. Analysis of SPKI/SDSI certificates using model checking. In *CSFW '02: Proceedings of the 15th IEEE Computer Security Foundations Workshop (CSFW'02)*, page 129, Washington, DC, USA, 2002. IEEE Computer Society.

[31] T. Jim. SD3: A trust management system with certified evaluation. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 106–115, 2001.

[32] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[33] D. Karabeg and V. Vianu. Simplification rules and complete axiomatization for relational update transactions. *ACM Transactions on Database Systems (TODS)*, 16(3):439–475, 1991.

[34] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.

[35] N. Li, B. Grosof, and J. Feigenbaum. A practically implementable and tractable delegation logic. In *IEEE Symposium on Security and Privacy*, pages 27–42, 2000.

[36] N. Li and J. C. Mitchell. Datalog with constraints: A foundation for trust management languages. In *Proc. PADL*, pages 58–73, 2003.

[37] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 114–130, 2002.

[38] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM*, 52(3):474–514, 2005.

[39] N. Li and M. V. Tripunitara. Security analysis in role-based access control. In *Proceedings of the ninth ACM symposium on access control models and technologies*, pages 126–135, New York, NY, USA, 2004. ACM Press.

[40] M. Liu. Extending datalog with declarative updates. *J. Intell. Inf. Syst.*, 20(2):107–129, 2003.

[41] S. Manchanda and D. S. Warren. A logic-based language for database updates. *Foundations of deductive databases and logic programming*, pages 363–394, 1988.

[42] S. Naqvi and S. Tsur. *A language for data and knowledge bases*. Computer Science Press, Rockville, MD, USA, 1989.

[43] National Health Service, UK. Integrated Care Records Service: Output based specification version 2. 2003.

[44] OASIS. *eXtensible Access Control Markup Language (XACML) Version 2.0 core specification*, 2005.

[45] R. Pucella and V. Weissman. Reasoning about dynamic policies. In *Foundations of Software Science and Computation Structures*, pages 453–467, 2004.

[46] R. Sandhu, V. Bhamidipati, E. Coyne, S. Canta, and C. Youman. The ARBAC97 model for role-based administration of roles: Preliminary description and outline. In *Proceedings of 2nd ACM Workshop on Role-Based Access Control*, pages 41–54, 1997.

[47] A. Sasturkar, P. Yang, S. D. Stoller, and C. R. Ramakrishnan. Policy analysis for administrative role based access control. In *CSFW '06: Proceedings of the 19th IEEE Workshop on Computer Security Foundations*, pages 124–138, Washington, DC, USA, 2006. IEEE Computer Society.

[48] M. Shanahan. Prediction is deduction but explanation is abduction. In *International Joint Conference on Artificial Intelligence*, pages 1055–1060, 1989.

[49] H. Tamaki and T. Sato. OLD resolution with tabulation. In *Proceedings on Third international conference on logic programming*, pages 84–98, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[50] D. Toman. Memoing evaluation for constraint extensions of Datalog. *Constraints*, 2(3/4):337–359, 1997.

[51] L. Wang, D. Wijesekera, and S. Jajodia. A logic-based framework for attribute based access control. In *FMSE '04: Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55, 2004.

A Proofs

A.1 Correctness of the Proof System

The following two theorems hold for well-formed programs $\mathcal{P}$ and ε-free sequences $\tilde{S}$.

Theorem 3.10 If $\{G_1\}\,\tilde{S}\,\{G_2\}$ and $\mathbf{B}_1$ satisfies G_1, then there exists a $\mathbf{B}_2$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{S}$ and $\mathbf{B}_2$ satisfies G_2.

Proof. By induction on the inference of $\{G_1\}\,\tilde{S}\,\{G_2\}$.

Case (cmd)**.** Then the judgment is of the form $\{G_1\}\,Q\,\{G_2\}$ and we have the following by (cmd) and assumption:

(1) $G_1 \supseteq F\theta$

(2) $F \in \mathit{pre}(Q)$

(3) $\mathit{eff}(Q) \circ G_1 \supseteq G_2$

(4) $\mathbf{B}_1$ satisfies G_1

From (4) and (1) we have that $\mathbf{B}_1$ satisfies $F\theta$. Together with (2) we thus have $\mathbf{B}_1 \vDash_{\mathcal{P}^\star} Q$ by Definition 3.8. Thus by rule (impl) we have that there exists a ground instantiation $Q \leftarrow \vec{L}$ of some rule in $\mathcal{P}^\star$, and $\mathbf{B}_1 \vDash_{\mathcal{P}^\star} \vec{L}$. Since none of the literals in $\vec{L}$ can be command literals (see Table 1), we have also $\mathbf{B}_1 \vDash_{\mathcal{P}} \vec{L}$. There exists a unique effect $\tilde{K}$ for Q and $(Q \leftarrow \vec{L} \otimes \tilde{K}) \in \mathcal{P}$. Suppose $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{K}$, then we can use (seq) to have $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \vec{L} \otimes \tilde{K}$, and have $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} Q$ by the rule (impl). It thus remains to show that there exists a $\mathbf{B}_2$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{K}$ and that $\mathbf{B}_2$ satisfies G_2.

By induction on the length of the sequence $\tilde{K}$, and rules (plus) and (min), we have $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \psi$ iff $\mathbf{B}_2 = (\mathbf{B}_1 \cup K^+) \setminus K^-$. We have ${G'_2}^+ := (\mathit{eff}(Q) \circ G_1)^+ = ({G_1}^+ \cup \mathit{eff}(Q)^+) \setminus \mathit{eff}(Q)^-$. By definition, $\mathit{eff}(Q)^+ = K^+$ and $\mathit{eff}(Q)^- = K^-$, and from (4) we have ${G_1}^+ \subseteq \mathbf{B}_1$. Hence ${G'_2}^+ \subseteq \mathbf{B}_2$. Since G'_2 is consistent by Lemma 3.4, we have that $\mathbf{B}_2$ satisfies G'_2, and therefore also its subset G_2.

Case (seq)**.** Then the judgment is of the form $\{G_1\}\,\tilde{S} \otimes Q\,\{G_2\}$. We have by (seq) and assumption:

(1) $\{G_1\}\,\tilde{S}\,\{H\}$

(2) $\{H\}\,Q\,\{G_2\}$

(3) $\mathbf{B}_1$ satisfies G_1

Because of (3) we can apply the induction hypothesis to (1), and thus there exists a $\mathbf{B}_i$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_i \vDash_{\mathcal{P}} \tilde{S}$ and $\mathbf{B}_i$ satisfies H. Therefore we can apply the induction hypothesis on (1), there exists a $\mathbf{B}_2$ such that $\mathbf{B}_i, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} Q$ and $\mathbf{B}_2$ satisfies G_2. Hence $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{S} \otimes Q$ with (seq). □

Theorem 3.11 If $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{S}$ and $\mathbf{B}_2$ satisfies G_2, then there exist G_1 such that $\{G_1\}\,\tilde{S}\,\{G_2\}$ and $\mathbf{B}_1$ satisfies G_1.

Proof. By induction on the structure of $\tilde{S}$.

Case $\tilde{S} = Q$. Then we have a judgment of the form $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} Q$, and by (impl) we have that $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \vec{L} \otimes \tilde{K}$ for some ground instantiation $Q \leftarrow \vec{L} \otimes \tilde{K}$ of a rule in $\mathcal{P}$. Thus $\mathbf{B}_1 \vDash_{\mathcal{P}} \vec{L}$ and $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{K}$ by (seq). We also have $\mathbf{B}_1 \vDash_{\mathcal{P}^\star} \vec{L}$, because $\vec{L}$ does not contain command literals, and hence by (impl) that $\mathbf{B}_1 \vDash_{\mathcal{P}^\star} Q$. Furthermore, by Definition 3.8, we have that there exists $F \in \mathit{pre}(Q)$ and a substitution θ such that $\mathbf{B}_1$ satisfies $F\theta$.

Consider $\mathbf{B}_1, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} \tilde{K}$. By induction on the length of the sequence $\tilde{K}$, and rules (plus) and (min), we have $\mathbf{B}_2 = (\mathbf{B}_1 \cup K^+) \setminus K^-$. Now assume $\mathbf{B}_2$ satisfies G_2, i.e. (1) ${G_2}^+ \subseteq (\mathbf{B}_1 \cup K^+) \setminus K^-$ and (2) $((\mathbf{B}_1 \cup K^+) \setminus K^-) \cap {G_2}^- = \emptyset$.

Define ${G_1}^+ := \mathbf{B}_1$ and ${G_1}^- := ({G_2}^- \setminus \mathbf{B}_1) \cup F\theta^-$. $\mathbf{B}_1$ satisfies G_1 because ${G_1}^+ \subseteq \mathbf{B}_1$ and ${G_1}^- \cap \mathbf{B}_1 = (({G_2}^- \setminus \mathbf{B}_1) \cap \mathbf{B}_1) \cup (F\theta^- \cap \mathbf{B}_1) = \emptyset$ (remember $\mathbf{B}_1$ satisfies $F\theta$). Remains to show that $\mathit{eff}(Q) \circ G_1 \supseteq G_2$. We have $\mathit{eff}(Q)^+ = K^+$ and $\mathit{eff}(Q)^- = K^-$, therefore for the positive part we have to show $({G_1}^+ \cup K^+) \setminus K^- \supseteq {G_2}^+$ which we have from (1) and the fact ${G_1}^+ = \mathbf{B}_1$. For the negative part we have to show that $(({G_2}^- \setminus \mathbf{B}_1) \cup F\theta^- \cup K^-) \setminus K^+ \supseteq {G_2}^-$. Assume thus $P \in {G_2}^-$. If $P \in K^+$ then $P \notin K^-$ because of the well-formedness condition of Definition 2.2. But then $(K^+ \setminus K^-) \cap {G_2}^- \neq \emptyset$, a contradiction to (2). Hence $P \notin K^+$. If $P \in {G_2}^- \setminus \mathbf{B}_1$ or $P \in K^-$ then we are finished. Hence assume $P \in \mathbf{B}_1$ and $P \notin K^-$, which is again a contradiction to (2).

Case $\tilde{S} = \tilde{S}' \otimes Q$. By (seq) there exists $\mathbf{B}_i$ such that $\mathbf{B}_1, \ldots, \mathbf{B}_i \vDash_{\mathcal{P}} \tilde{S}'$ and $\mathbf{B}_i, \ldots, \mathbf{B}_2 \vDash_{\mathcal{P}} Q$. We have $\mathbf{B}_2$ satisfies G_2, thus we can apply the induction hypothesis on the second equation to have that there exists a H such that $\{H\}\, \tilde{S}\, \{G_2\}$ and $\mathbf{B}_i$ satisfies H. Hence we can apply the induction hypothesis on the first equation $\{G_1\}\, \tilde{S}'\, \{H\}$ and $\mathbf{B}_1$ satisfies G_1. Finally rule (seq) gives $\{G_1\}\, \tilde{S}' \otimes Q\, \{G_2\}$. □

A.2 Correctness of the Tabling Algorithm

Lemma A.1 *If $\{G_1\}\, \tilde{T}\, \{G_2\}$, then both G_1 and G_2 are ground and consistent.*

Proof. By a simple induction on the inference of $\{G_1\}\, \tilde{T}\, \{G_2\}$, where Lemma 3.4 and the well-formedness condition for effects of Definition 2.2 are used in case (cmd). □

Lemma A.2 *If ${D_2}^+ \cap E^- = \emptyset$, ${D_2}^- \cap E^+ = \emptyset$, and $D_1 \supseteq D_2 \setminus E$, then the following inequality holds:*

$$E \circ D_1 \supseteq D_2$$

Proof. **Case "+".** Suppose $Q \in {D_2}^+$, but $Q \notin ({D_1}^+ \cup E^+) \setminus E^-$. Because ${D_2}^+ \cap E^- = \emptyset$ we have $Q \notin E^-$, and thus $Q \notin {D_1}^+ \cup E^+$. We know ${D_1}^+ \supseteq {D_2}^+ \setminus E^+$, and therefore $Q \notin {D_2}^+ \setminus E^+$. Hence, because of $Q \in {D_2}^+$, it must be that $Q \in E^+$, a contradiction. **Case "−".** This case is completely analogous and uses equation ${D_2}^- \cap E^+ = \emptyset$. □

In the following lemma, we use the notation $\star\langle D_1; \tilde{T}; D_2\rangle$ to express that some node may either be an answer or a goal node.

Lemma A.3 *If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is reachable from an initial state, then the following implication holds: if $n \equiv \star\langle D_1; \tilde{T}; D_2\rangle \in \mathcal{N}$ and D_1 is consistent, then $\{D_1\}\, \tilde{T}\, \{D_2\}$. Furthermore, if m is an answer node, then $\mathbf{B}_0$ satisfies D_1.*

Proof. By induction on the transition rules which insert $\star\langle D_1; \tilde{T}; D_2\rangle$ into $\mathcal{N}$.

Case (root)**. Subcase** $n \in \mathcal{N}_1$**.** Then n is of the form $\mathsf{goal}\langle D_1; Q; D_2\rangle$. In order to be able to derive $\{D_1\}\, Q\, \{D_2\}$ with rule (cmd), we have to show

(i) D_1 is ground and consistent, and Q is ground,

(ii) there exists some F' such that $D_1 \supseteq F'$ and $F' \in \mathit{pre}(Q)$, and

(iii) $\mathit{eff}(Q) \circ D_1 \supseteq D_2$.

Ad (i)**.** The groundness is immediate since the algorithm is for the propositional case, and the consistency of D_1 holds by assumption.
Ad (ii)**.** We know $F \in \mathit{pre}(Q)$ by construction of the goal node, and from $D_1 \stackrel{\text{def}}{=} (D_2 \setminus E) \cup F$ we have $D_1 \supseteq F$.
Ad (iii)**.** We have ${D_2}^+ \cap E^- = \emptyset$ and ${D_2}^- \cap E^+ = \emptyset$ by construction of the goal node. Because $D_1 \stackrel{\text{def}}{=} (D_2 \setminus E) \cup F$ we have $D_1 \supseteq D_2 \setminus E$. Thus we can apply Lemma A.2 to have $E \circ D_1 \supseteq D_2$, and we know that $E \stackrel{\text{def}}{=} \mathit{eff}(Q)$.

Subcase $n \in \mathcal{N}_2$**.** Then n is of the form $\mathsf{ans}\langle D_2; \varepsilon; D_2\rangle$. By assumption D_2 is ground and consistent, therefore $\{D_2\}\, \varepsilon\, \{D_2\}$ holds by rule (eps). Furthermore, we have that $\mathbf{B}_0$ satisfies D_2 by construction of the answer node, and the second part of the lemma holds as well.

Case (ans)**.** The induction hypothesis holds for $n_1 \equiv \mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle$ and $n_2 \equiv \mathsf{goal}\langle E_1; Q; E_2\rangle \in \mathit{Wait}(D_2)$, since all elements in the wait table must have been in the node set of a previous state. Node n is of the form $\mathsf{ans}\langle D_1; \tilde{T} \otimes Q; E_2\rangle$ where we have by Definition 3.14 that $D_2 = E_1$ and all of D_1, D_2, E_1, and E_2 are consistent. We want to derive $\{D_1\}\, \tilde{T} \otimes Q\, \{E_2\}$ by rule (seq) for the case where D_1 is consistent, and ${D_1}^- \cap \mathbf{B}_0 = \emptyset$. Therefore we are left to show that there exists an H such that both $\{D_1\}\, \tilde{T}\, \{H\}$ and $\{H\}\, Q\, \{E_2\}$ hold.

Since D_1 is consistent, we can apply the induction hypothesis for n, and thus $\{D_1\}\, \tilde{T}\, \{D_2\}$. With Lemma A.1 applied to $\{D_1\}\, \tilde{T}\, \{D_2\}$ we have that D_2 is ground and consistent, and because of $D_2 = E_1$ this is true for $E_1 = D_2$ as well. Hence, as above for n_1 we have $\{D_2\}\, Q\, \{E_2\}$ in the case of n_2.

Case (goal_1)**.** Analogous to case (ans). □

Theorem 3.17 If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is reachable from some initial state, then the following implication holds for all $D_2 \in \mathit{dom}(\mathit{Ans})$: if $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$ and D_1 is consistent, then $\{D_1\}\, \tilde{T}\, \{D_2\}$ and $\mathbf{B}_0$ satisfies D_1.

Proof. A direct consequence of Lemma A.3. □

Lemma A.4 *If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is a final state reachable from some initial state, the following implication holds for all $D_2 \in \mathit{dom}(\mathit{Ans})$: if $\{G_1\}\, \tilde{S}\, \{G_2\}$ and $G_2 \supseteq D_2$, then*

(i) *if $\mathbf{B}_0$ satisfies G_1, then there exists $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$*

(ii) *if $|\tilde{S}| = 1$ and $G_1 \not\supseteq D_2$, then there exists $\mathsf{goal}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Wait}(D_1)$*

such that in both cases $G_1 \supseteq D_1$, $|\tilde{S}| \geq |\tilde{T}|$, and $\mathit{atoms}(\tilde{S}) \supseteq \mathit{atoms}(\tilde{T})$.

Proof. By induction on the inference of $\{G_1\}\, \tilde{T}\, \{G_2\}$.

Case (eps)**.** By assumption we have $\{G\}\, \varepsilon\, \{G\}$ and $G \supseteq D_2$. Since $|\varepsilon| < 1$ we are proving case (i) only and thus assume that $\mathbf{B}_0$ satisfies G. $\mathbf{B}_0$ satisfies D_2

as well, because $G \supseteq D_2$. Since $D_2 \in dom(Ans)$, the node $\langle D_2 \rangle$ must have been evaluated via rule (root) at some point. Therefore $n \equiv \mathsf{ans}\langle D_2; \varepsilon; D_2 \rangle \in \mathcal{N}_2$ and all conditions of the lemma are fulfilled. Thus n will be processed by rule (ans) at some point, which ensures that $n \in Ans(D_2)$, provided that there is no node m already in $Ans(D_2)$ which subsumes n (in which case Definition 3.15 ensures that m fulfills the goals of the lemma).

Case (cmd)**.** By assumption we have $\{G_1\}\, P\, \{G_2\}$ and $G_2 \supseteq D_2$, where P is some command. Together with the preconditions of (cmd) we have the following facts:

(1) $\{G_1\}\, P\, \{G_2\}$

(2) $G_2 \supseteq D_2$

(3) G_1 is ground and consistent

(4) P is ground

(5) $G_1 \supseteq F$

(6) $F \in pre(P)$

(7) $\mathit{eff}(P) \circ G_1 \supseteq D_2$

Since $D_2 \in dom(Ans)$, the node $\langle D_2 \rangle$ must have been evaluated via rule (root) at some point. We distinguish two cases:
Subcase "$G_1 \supseteq D_2$"**.** Then we are only concerned with case (i), for which we can assume that $\mathbf{B}_0$ satisfies G_1. We can thus argue just as in case (eps) that there is some node $\mathsf{ans}\langle D_2; \varepsilon; D_2 \rangle \in \mathcal{N}_2$. Note in particular that $|P| \geq |\varepsilon|$, and $atoms(P) \supseteq atoms(\varepsilon) = \emptyset$.
Subcase "$G_1 \not\supseteq D_2$"**.** Consider case (ii) first: we are trying to prove that $n \equiv \mathsf{goal}\langle D_1; Q; D_2 \rangle$ is an element of the set $\mathcal{N}_1$ of (root) during a run of the algorithm and that the remaining consequences of the theorem hold. The following abbreviations are used:

(def-1) $E \stackrel{\text{def}}{=} \mathit{eff}(Q)$

(def-2) $D_1 \stackrel{\text{def}}{=} (D_2 \setminus E) \cup F$

We have to show that $G_1 \supseteq D_1$. Since $G_1 \supseteq F$, it remains to show that $G_1 \supseteq D_2 \setminus E$. Thus assume $Q \in {D_2}^+$ and $Q \notin E^+$. The positive part of (7) is $({G_1}^+ \cup E^+) \setminus E^- \supseteq {D_2}^+$, hence $Q \in {G_1}^+$. The reasoning for the negative case is completely analogous.

In order to ensure that the goal node n is an element of the node set during some intermediate stage of the algorithm, all the conditions for the construction of set $\mathcal{N}_2$ have to be fulfilled.

We are thus left to show that $E \cap D_2 \neq \emptyset$, D_1 is consistent, and ${D_2}^+ \cap E^- = \emptyset$ and ${D_2}^- \cap E^+ = \emptyset$. From the assumption of the subcase we know that $D_2 \setminus G_1$ is nonempty. Therefore assume $Q \in D_2 \setminus G_1$, and thus with (7) we have $Q \in {D_2}^+ \subseteq ({G_1}^+ \cup E^+) \setminus E^-$. Since $Q \notin G_1$ it must be that $Q \in E^+$. We can argue with the negative part of (7) in the same way and have thus shown $E \cap D_2 \neq \emptyset$. It is clear that D_1 is consistent from (3) and $G_1 \supseteq D_1$. For the last goal, note that we have $E \circ G_1 \supseteq D_2$ from (7). Equivalently, $({G_1}^+ \cup E^+) \setminus E^- \supseteq {D_2}^+$ and

$(G_1{}^- \cup E^-) \setminus E^+ \supseteq D_2{}^-$. Thus follows that $Q \in D_2{}^+$ implies $Q \notin E^-$, and $Q \in D_2{}^-$ implies $Q \notin E^+$.

Node n is thus element of $\mathcal{N}_2$ is thus processed by either (goal_1) or (goal_2) and element of $\mathit{Wait}(D_1)$.

For case (i), assume that $\mathbf{B}_0$ satisfies G_1, thus, because of $G_1 \supseteq D_1$, $\mathbf{B}_0$ also satisfies D_1. Since $n \in \mathit{Wait}(D_1)$, D_1 must have been spawned as a new root $\langle D_1 \rangle$ by (goal_2) during the course of the algorithm. Because of $\mathbf{B}_0$ satisfies D_1, we can argue as in case (eps) that there exists an answer node carrying the empty command ε. This answer node will be merged either in (ans) or (goal_1) with $n \in \mathit{Wait}(D_1)$, to yield $\mathsf{ans}\langle D_1; \varepsilon \otimes Q; D_2 \rangle$. Note especially that $|P| = |\varepsilon \otimes Q|$, and $\mathit{atoms}(P) \supseteq \mathit{atoms}(\varepsilon \otimes Q) = \mathit{atoms}(Q)$. Furthermore, n will eventually be processed by rule (ans) and hence there is some node $m \in \mathit{Ans}(D_2)$ which fulfills the goals of the lemma by the argumentation of rule (eps).

Case (seq)**.** Since $|\tilde{S} \otimes P| > 1$, we are proving case (i) only and thus assume $\mathbf{B}_0$ satisfies G_1. We can summarize the assumptions (partly from rule (seq)) as follows:

(1) $\mathbf{B}_0$ satisfies G_1

(2) $\{G_1\}\, \tilde{S}\, \{H\}$

(3) $\{H\}\, P\, \{G_2\}$

(4) $\{G_1\}\, \tilde{S} \otimes P\, \{G_2\}$

(5) $G_2 \supseteq D_2$

Subcase "$\mathbf{B}_0$ satisfies H or $H \supseteq D_2$"**.** If $\mathbf{B}_0$ satisfies H we can apply the induction hypothesis, case (i), on (3) to have $\mathsf{ans}\langle D_1; Q; D_2 \rangle \in \mathit{Ans}(D_2)$ for which the goals of the lemma are fulfilled. Note especially that $|\tilde{S} \otimes P| \geq |Q|$, and $\mathit{atoms}(\tilde{S} \otimes P) \supseteq \mathit{atoms}(Q)$ because $\mathit{atoms}(P) \supseteq \mathit{atoms}(Q)$.

If $H \supseteq D_2$ we can also apply the induction hypothesis, case (i), but now on (2) to have $\mathsf{ans}\langle D_1; \tilde{T}; D_2 \rangle \in \mathit{Ans}(D_2)$ for which the goals of the lemma are fulfilled. Note especially that $|\tilde{S} \otimes P| \geq |\tilde{T}|$, and $\mathit{atoms}(\tilde{S} \otimes P) \supseteq \mathit{atoms}(\tilde{T})$ because $\mathit{atoms}(\tilde{S}) \supseteq \mathit{atoms}(\tilde{T})$.

Subcase "$\mathbf{B}_0$ does not satisfy H and $H \not\supseteq D_2$"**.** Since $|P| = 1$ and $H \not\supseteq D_2$, we can use the induction hypothesis for (ii) to have that there exists $\mathsf{goal}\langle E; Q; D_2 \rangle \in \mathit{Wait}(E)$ such that:

- $H \supseteq E$
- $|P| \geq |Q|$
- $\mathit{atoms}(P) \supseteq \mathit{atoms}(Q)$

Because of (1) and $H \supseteq E$, we can apply the induction hypothesis for case (i) to (2). Thus we have that there exists $\mathsf{ans}\langle D_1; \tilde{T}; E \rangle \in \mathit{Ans}(E)$ such that:

- $G_1 \supseteq D_1$
- $|\tilde{S}| \geq |\tilde{T}|$
- $\mathit{atoms}(\tilde{S}) \supseteq \mathit{atoms}(\tilde{T})$

Assume the goal node was inserted into the wait table before the answer node was inserted into the answer table. Then rule (ans) will try to merge the two nodes. Otherwise this will be attempted in (goal_1). Therefore $n' \equiv \mathsf{ans}\langle D_1; \tilde{T} \otimes Q; D_2\rangle$ is inserted as a new answer, and it is easy to check with the above itemized assumptions the consequences of the lemma are fulfilled. Furthermore, both (ans) and (goal_1) ensure that $n' \in \mathcal{N}'$, thus n will eventually be processed by rule (ans) and hence there is some node $m \in \mathit{Ans}(D_2)$ which fulfills the goals of the lemma by the argumentation of rule (eps). □

Theorem 3.18 If $(\mathcal{N}, \mathit{Ans}, \mathit{Wait})$ is a final state reachable from some initial state, the following implication holds for all $D_2 \in \mathit{dom}(\mathit{Ans})$: if $\{G_1\}\,\tilde{S}\,\{G_2\}$, $G_2 \supseteq D_2$, and $\mathbf{B}_0$ satisfies G_1 then there exists $\mathsf{ans}\langle D_1; \tilde{T}; D_2\rangle \in \mathit{Ans}(D_2)$ such that $\mathbf{B}_0$ satisfies D_1, $|\tilde{S}| \geq |\tilde{T}|$, and $\mathit{atoms}(\tilde{S}) \supseteq \mathit{atoms}(\tilde{T})$.

Proof. A direct consequence of Lemma A.4. □

Lemma 3.19 Every set of answer nodes $\mathcal{N}$ in which only finitely many predicate symbols occur has a finite subset $\mathcal{N}_{\mathit{fin}} \subseteq \mathcal{N}$ such that:

$$\forall n \in \mathcal{N}\ \exists m \in \mathcal{N}_{\mathit{fin}} : n \succeq m$$

Proof. Since the set of (propositional) predicate symbols occurring in $\mathcal{N}$ is finite, the set of literals constructed from these is also finite. Let $\mathcal{P}$ be the powerset of the former, and $\mathcal{Q}$ be the powerset of the latter. Both powersets are finite. Letting A and B range over $\mathcal{Q}$ and C range over $\mathcal{P}$, define $\mathcal{N}_{A,B,C}$ to be the set of answer nodes in $\mathcal{N}$ of the form $\mathsf{ans}\langle A; \tilde{S}; C\rangle$ such that $\mathit{atoms}(\tilde{S}) = B$. Then $\mathcal{N}$ can be partitioned into finitely many such sets.

Each such set $\mathcal{N}_{A,B,C}$ contains an answer node $n_{A,B,C}$ with a sequence that is shorter than (or equally long as) the sequences occurring in the other answer nodes. By construction, this node subsumes all other nodes in $\mathcal{N}_{A,B,C}$.

Therefore, the finite set $\mathcal{N}_{\mathit{fin}} = \{n_{A,B,C} \ :\ A, B \in \mathcal{Q}, C \in \mathcal{P}\}$ subsumes the entire set $\mathcal{N}$. □

Theorem 3.20 All transition paths starting from an initial state are of finite length.

Proof. Since $\mathcal{P}$ is finite and propositional, (goal_2) can only spawn finitely many root nodes. Therefore, (root) can only be applied finitely often, and each application produces only finitely many goal and answer nodes. Every goal node either gets consumed by (goal_2) or by (goal_1). The latter produces a set of answer nodes. Therefore, after a finite number of steps, only answer nodes are left. By Lemma 3.19, only finitely many answer nodes can satisfy the non-subsumption condition of (ans), hence it can also only be applied a finite number of times. □