Operational Semantics for DKAL: Application and Analysis

Yuri Gurevich[1] and Arnab Roy[2,⋆]

[1] Microsoft Research Redmond
gurevich@microsoft.com
[2] Stanford University
arnab@cs.stanford.edu

Abstract. DKAL is a new expressive high-level authorization language. It has been successfully tried at Microsoft which led to further improvements of the language itself. One improvement is the separation of concerns between static core policies and dynamic workflow; important safety properties can be proved from the core policies alone, independently from the workflow. Another improvement is true decentralization; different principals live in different worlds exchanging information by means of communication and filtering assertions. We also present some complexity results.

1 Introduction

Preamble. Distributed Knowledge Authorization Language (DKAL) is a new high-level authorization language for distributed systems [6,7]; it is based on liberal datalog [6,3] and is considerably more expressive than other high-level authorization languages in the literature. The first practical application of DKAL was to the problem of automating source asset management (SAM) at Microsoft. We partook an active role in that endeavor. The automation problem involves more than DKAL and is still work in progress. While the trial has been successful, the application provoked improvements of DKAL that became a part of a larger revision of DKAL [7]. In this paper we present some lessons learned during the first practical application of DKAL and the improvements that we made to DKAL. In addition, we present some related theoretical results. We presume some familiarity with [6]; otherwise this paper is self-contained.

Source asset management (SAM). Large software companies have many partners, contractors and subcontractors who need to access the sources of the various software products produced by the company. The ever-growing number of such requests necessitates clear access control policies regulating who is entitled to use what sources, where, for how long, and so on. The company also needs processes in place to efficiently implement those policies. DKAL enables formulating rich and sophisticated policies that result in simpler audit and feasible automation.

⋆ The work was done in the summer of 2008 when the second author was an intern at Microsoft Research.

S. Fischer-Hübner et al. (Eds.): TrustBus 2009, LNCS 5695, pp. 149–158, 2009.

Lessons. The most important lesson that we learned working on SAM was a separation of concerns: static core policies on one side, and dynamic workflow on the other side. The workflow includes the procedural aspects of access decisions; the idea is to ensure the flexibility of the the way access decisions are made. It may be quite complex, but the policies should be succinct and easy to understand. Furthermore, the policies should guarantee important safety properties of the access decision process for any workflow. Various compliance requirements and regulations typically can be formulated as safety properties. It is desirable that the policies alone guarantee the compliance with those requirements and regulations thus eliminating the need to understand large amounts of procedural code.

The other important lesson was that in the case of automated access control regulated by clear access control policies, it is crucial for auditing purposes to properly log all human judgments. No set of policies and processes makes industrial access control deterministic. The inherent non-determinism is resolved by human judgments in making and delegating access decision. Logging these judgments is necessary for auditing purposes as it allows the auditor to understand who made what decisions and whether they had the right to do so (either originally or by delegation).

Operational semantics. As a result of the lessons learned, we improved the operational semantics of DKAL achieving true decentralization. In SecPAL [2], the precursor of DKAL, principals are distributed but their sayings are collected and processed together. In DKAL 1 [6], principals compute their own knowledge, yet some vestiges of a centralized approach remain, e.g. the global state. Now different principals live in different worlds exchanging information by means of communication and filtering assertions. Information exchange is allowed to be flexible, separating policy and workflow. Additionally, logging the right information is a natural byproduct of the way operational semantics is defined.

Complexity. We explored two communication models for defining workflows and proved that deciding reachability questions is possible in polynomial time in certain cases whereas deciding invariants is coNP-complete.

2 Related Work

Several languages have been developed in recent years to express authorization policies. The late genealogy of DKAL consists primarily of Binder, Delegation Logic and SecPAL. Binder [5] builds directly on Datalog, extending it with an import/export construct `says` that connects Datalog policies of different principals and makes the issuer of an imported assertion explicit. Delegation Logic [9,10] does not have explicit issuers for all assertions but it has constructs specific to authorization including ones for delegation, representation and thresholds.

SecPAL [2] has both explicit issuers and specific authorization constructs designed with distributed systems in mind. The number of constructs is deliberately kept low, but the language is expressive and captures numerous standard authorization scenarios. One important construct is *can say*: if A says B can say foo and if B says foo then A says foo. The can-say construct can be nested. The semantics is defined by means of a few deduction rules but, for execution purposes, SecPAL reduces to safe Constraint Datalog. Nesting of the can-say construct gives rise to relations whose arity depends on the nesting depth.

DKAL [6] extends SecPAL in many directions as far as expressivity is concerned. But it is quite different and builds on Liberal Datalog [6,3]. In particular DKAL allows one to freely use functions; as a result, principals can quote other principals. Contrary to SecPAL, communication in DKAL is targeted. Targeted communication plugs an information leak in SecPAL [6, §4] and improves the liability protection. The additional expressivity is achieved within the same bounds of computational complexity as that of SecPAL.

The main novelty of this paper is that we take workflow into account and deal explicitly with the runtime evolution of policies. In this connection we extend DKAL with the *from* construct. In [6], the sender directs communication to particular receivers but the receivers are passive. The from construct allows the receivers to filter incoming information. A limited policy evolution is possible in SecPAL: policy statements can be revoked but revocation is governed by static rules.

There has been work on the analysis of dynamic evolution of authorization policies in the Trust Management framework [11]. The approach there has been to specify statically what kind of policy rules can be added or removed. They analyzed reachability and invariant checking. We do that too in our framework. The richness of DKAL compared to the simpler TM language makes these questions even more interesting.

Modeling access control policies has been a scientifically very fruitful area with a considerable amount of literature, [1,4] to cite two. Since we focus on operational semantics in this paper, we skip a detailed discussion of the literature on policy modeling itself. Interested readers are referred to [6].

3 Operational Semantics

We refine the computation mechanism of DKAL, making explicit the transaction of information that was only partially explicit in [6]. Also we model situations where principals may listen only to information that they want to listen to by introducing a `from` construct.

We use the following form of DKAL's TrustApplication rule:

$$x \;\leq\; (p \;\texttt{said}\; x + p \;\texttt{tdon}\; x)$$

Here `tdon` alludes to "trusted on." (For experts on DKAL: we deprecate predicate $\texttt{knows}_0$ and function $\texttt{said}_0$, and we rename $\texttt{tdon}_0$ and `tdon` to `tdon` and $\texttt{tdon}^*$ respectively.)

3.1 Terminology

Every principal has an immutable, or at least stable, core policy and a dynamic policy where it can assert communication statements. We use the following notation:

Π_p	Core policy of p.
Δ_p	Dynamic policy of p.
$\mathcal{K}_p$	$\{x : \Pi_p \cup \Delta_p \vdash x,$ no free variables in $x\}$, $\mathcal{K}_p$ is the *knowledge* of p.

3.2 Communication Rules

The main rule is COM. In simple form it says that if a principal A says an infon x to a principal B and if B is willing to accept x then x gets transmitted and B learns that A `said` x. In general, the rule is more complicated and involves a substitution η such that ηx is ground.

$$\frac{\begin{cases}(A\ \texttt{to}\ \ q : x \leftarrow x_1, x_2, \cdots, x_m, \delta) \in \Pi_A \cup \Delta_A \\ (\mathrm{B}\ \texttt{from}\ \ p : y) \in \Pi_B \cup \Delta_B \\ \{\eta x_1, \eta x_2, \cdots, \eta x_m\} \subseteq \mathcal{K}_A, \mathrm{Substrate}_A \models \eta\delta \\ \eta p = A, \eta q = B, \eta x = \eta y,\ \text{no free variables in } \eta x\end{cases}}{\Delta_B \mathrel{+}= \ A\ \texttt{said}\ \eta x} \quad (\texttt{COM})$$

The last line means that Δ_B is augmented with infon A `said` ηx. In practical implementations, any such communication resulting from this rule will be logged.

COM can be generalized to express common scenarios more succinctly. One such scenario is that if a principal B accepts an infon x, then it accepts infons of the form

$$z_1\ \texttt{tdon}\ z_2\ \texttt{tdon}\ \ldots\ z_k\ \texttt{tdon}\ x.$$

$$\frac{\begin{cases}(A\ \texttt{to}\ \ q : z_1\ \texttt{tdon}\ z_2\ \texttt{tdon}\ \ldots z_k\ \texttt{tdon}\ x \\ \qquad\qquad \leftarrow x_1, x_2, \cdots, x_m, \delta) \in \Pi_A \cup \Delta_A \\ (B\ \texttt{from}\ \ p : y) \in \Pi_B \cup \Delta_B \\ \{\eta x_1, \eta x_2, \cdots, \eta x_m\} \subseteq \mathcal{K}_A, \mathrm{Substrate}_A \models \eta\delta \\ \eta p = A, \eta q = B, \eta x = \eta y,\ \text{no free variables in } \eta x \text{ and } \eta z_i\text{'s}\end{cases}}{\Delta_B \mathrel{+}= \ A\ \texttt{said}\ \eta z_1\ \texttt{tdon}\ \eta z_2\ \texttt{tdon}\ \cdots \eta z_k\ \texttt{tdon}\ \eta x} \quad (\texttt{TCOM})$$

4 Example

Alice wants to download an article from the Chux (an allusion to Chuck's) store. Publishing house Best owns the copyright and delegates downloading permission to Chux. Chux has the policy that anybody meeting certain approval criteria can download the article. All this gives rise to the following core policies.

$$\begin{aligned}
\mathit{\Pi}_{best}: &\quad \text{best } \texttt{to } p: \text{ chux } \texttt{tdon } p \textit{ canDownload} \text{ article}\\
\mathit{\Pi}_{chux}: &\quad \text{chux } \texttt{to } q: q \textit{ canDownload} \text{ article} \leftarrow \mathit{approve}(q, \text{article})\\
\mathit{\Pi}_{alice}: &\quad \text{alice}: \text{ best } \texttt{tdon}^{*} \text{ alice } \textit{canDownload} \text{ article}
\end{aligned}$$

Alice starts the process by issuing a dynamic `from` assertion.

$$\text{alice asserts} \quad \texttt{from } r: \text{alice } \textit{canDownload} \text{ article.} \tag{1}$$

Assuming the relation $\mathit{approve}(\mathit{alice}, \text{article})$ is true, we have this evolution:

$$\begin{aligned}
\text{By } (1), &\quad \mathit{\Delta}_{alice} \mathrel{+}= \text{ alice } \texttt{from } r: \text{alice } \textit{canDownload} \text{ article.} &(2)\\
\text{By } (2), \mathit{\Pi}_{best}, \text{TCOM} &\quad \mathit{\Delta}_{alice} \mathrel{+}= \text{ best } \texttt{said} \text{ chux } \texttt{tdon}\\
&\qquad\qquad \text{alice } \textit{canDownload} \text{ article.} &(3)\\
\text{By } (2), \text{COM} &\quad \mathit{\Delta}_{alice} \mathrel{+}= \text{ chux } \texttt{said} \text{ alice } \textit{canDownload} \text{ article.} &(4)\\
\text{By } (3,4), \mathit{\Pi}_{alice} &\quad \mathit{\Pi}_{alice} \cup \mathit{\Delta}_{alice} \vdash \text{alice } \texttt{knows}\\
&\qquad\qquad \text{alice } \textit{canDownload} \text{ article.} &(5)
\end{aligned}$$

We now add more details to the previous example. Chux lets a customer download an article if she authorizes the right amount of money for payment and has a perfect payrate. Chux trusts his accountant ac.Chux on customer's payrate.

$$\begin{aligned}
\mathit{\Pi}_{chux}: &\quad \text{chux } \texttt{to } a: \ a \textit{ canDownload } s \leftarrow a \textit{ authorized } \$k \text{ to chux for } s,\\
&\qquad a \textit{ hasPayRate} \text{ Perfect}, \mathit{price}(s) = k.\\
&\quad \text{chux}: \text{ accounts.Chux } \texttt{tdon } a \textit{ hasPayRate } e.\\
&\quad \text{chux}: a \texttt{ tdon } a \textit{ authorized } \$k \text{ to chux for } s.\\
&\quad \text{chux } \texttt{from } a: a \textit{ authorized } \$k \text{ to chux for } s.\\
\mathit{\Pi}_{ac.Chux}: &\quad \text{ac.Chux } \texttt{to } \text{chux}: \ a \textit{ hasPayRate } e \leftarrow \ a \textit{ hasPayRate } e.
\end{aligned}$$

The last assertion is not a tautology. If accounts.chux knows that a has PayRate e then it says that to Chux.

Alice starts by authorizing payment to Chux for the article and then asks for the article. Chux then asks whether Alice has perfect payrate. The dynamic assertions are as follows:

$$\begin{aligned}
&\text{alice asserts} \quad \texttt{to } \text{chux}: \text{ alice } \textit{authorized} \text{ \$40 to chux for article.}\\
&\text{alice asserts} \quad \texttt{from } r: \text{alice } \textit{canDownload} \text{ article.}\\
&\text{chux asserts} \quad \texttt{from } a: \text{ alice } \textit{hasPayRate } e.
\end{aligned}$$

If the price of the article is \$40 and Alice has a perfect payrate, we can again infer using the operational semantics that Alice knows she can download the article.

5 Modeling Source Asset Management

In this section we explore the application of DKAL to a practical problem in large software companies, namely Source Asset Management (SAM). This encompasses formulating a policy and processes for making code access decisions.

A Project Manager in a large software company A wants a codeset to be accessed by staff at another company. In A, the codeset is owned by an Information Asset Owner (IAO) who usually delegates such access decisions to a Source Rep. The access decision takes into account the codeset, the company to access the codeset, the type of permission — read/write or read-only.

In our DKAL model, authorizations are managed by Authorization Managers (AMs) of the respective companies. AMs are automated tools modeled as principals. As far as code authorization is concerned, the AM of the company is (or represents) the company itself. The Project Manager, IAO and Source Rep are also modeled as principals. The policy of the authorization manager A-AM of A may be as follows:

$$
\begin{aligned}
\Pi_{A\text{-AM}}: \quad & p \texttt{ tdon } q \; \mathit{canGet}(\mathit{codeset}, \mathit{parameters}) \leftarrow \mathit{assetOwner}(p, \mathit{codeset}).\\
& p \texttt{ tdon } q \texttt{ tdon } r \; \mathit{canGet}(\mathit{codeset}, \mathit{parameters}) \leftarrow \\
& \quad p \texttt{ tdon } r \; \mathit{canGet}(\mathit{codeset}, \mathit{parameters}),\ p \ \mathit{is\ the\ manager\ of}\ q.\\
& p \texttt{ tdon } q \; \mathit{canAccess}(\mathit{code}, \mathit{parameters}) \leftarrow \\
& \quad p \ \mathit{is\ a\ Project\ Manager}, r \; \mathit{canGet}(\mathit{codeset}, \mathit{parameters}),\\
& \quad r \ \mathit{is\ the\ AM\ of}\ q, \mathit{code\ is\ in\ codeset}.\\
& p \texttt{ tdon } q \texttt{ tdon } r \; \mathit{canAccess}(\mathit{code}, \mathit{parameters}) \leftarrow \\
& \quad p \texttt{ tdon } q \; \mathit{canAccess}(\mathit{code}, \mathit{parameters}).
\end{aligned}
$$

The *canGet* function specifies access permission to a set of source code files, whereas the *canAccess* function specifies access permission to a particular source code file.

Consider a scenario where A wants to allow a vendor B an access to a codeset, and where B-AM has the policy

$$
\begin{aligned}
\Pi_{B\text{-AM}}: \quad & A\text{-AM} \texttt{ tdon } B\text{-AM} \; \mathit{canGet}(\mathit{codeset}, \mathit{parameters}).\\
& A\text{-AM} \texttt{ tdon } q \; \mathit{canAccess}(\mathit{code}, \mathit{parameters}).
\end{aligned}
$$

Suppose that Alan, Alfred, Andrew, Anthony and Alice work for A, and let Alan $\triangleright$ Alfred $\triangleright$ Andrew $\triangleright$ Anthony where p $\triangleright$ q means p is the manager of q. The role of Alice in the hierarchy will not matter. Suppose further that Alan is the IAO of *drivercodes*, a codeset that is to be given access to by B. A simplified workflow for the access decision may be as follows:

Access decision starts; Anthony goes to A-AM to request that B be given access to drivercodes.

$$A\text{-AM} \ \mathit{asserts} \texttt{ from } p: \ B\text{-AM} \ \mathit{canGet}(\text{drivercodes}, \text{params1}).$$

Decision is escalated to Andrew, and then to Alfred.

Alfred *asserts* `to` A-AM : B-AM *canGet*(drivercodes, params1).

Alan *asserts* `to` A-AM : Alfred `tdon` B-AM *canGet*(drivercodes, params1).

A-AM communicates decision to B-AM.

A-AM *asserts* `to` B-AM : B-AM *canGet*(drivercodes, params1).

A developer Bruce in B needs a portion of drivercodes called gfx. Policy of Bruce is as follows:

$$\Pi_{bruce}: \quad B\text{-AM } \texttt{tdon} \text{ Bruce } \mathit{canAccess}(\mathit{code}, \mathit{parameters}).$$

A typical workflow to acquire this code would be as follows:

Bruce asks for access to the code.

Bruce *asserts* `from` p : Bruce *canAccess*(gfx, params1).

In the workflow, this goes to B-AM. B-AM forwards the query along.

B-AM *asserts* `from` p : Bruce *canAccess*(gfx, params1).

In the workflow, query goes to A-AM, which again forwards the query.

A-AM *asserts* `from` p : Bruce *canAccess*(gfx, params1).

The workflow "chooses" who to ask about the query. It happens to be Alice. She decides to grant the code access to Bruce.

Alice *asserts* `to` A-AM : Bruce *canAccess*(gfx, params1).

Anthony confirms that Alice has the authority to grant access.

Anthony *asserts* `to` A-AM : Alice `tdon` Bruce *canAccess*(gfx, params1).

A-AM infers that Bruce can access the code and informs B-AM accordingly.

A-AM *asserts* `to` B-AM : Bruce *canAccess*(gfx, params1).

B-AM trusts A-AM on such statements and concludes that Bruce can access the code. B-AM communicates the decision to Bruce.

B-AM *asserts* `to` Bruce : Bruce *canAccess*(gfx, params1).

Bruce concludes that he can access the code.

As you can imagine, the workflow can proceed in myriad possible ways. Yet, the core policies of A-AM, B-AM and Bruce lets us state the following theorem, which holds independent of the workflow.

Theorem 1 (Safety of SAM Policy)

1. *If B-AM* `knows` *B-AM canGet*(*codeset*, *parameters*) *then B-AM* `knows` *A-AM* `said` *B-AM canGet*(*codeset*, *parameters*)
2. *If Bruce* `knows` *Bruce canAccess*(*code*, *parameters*) *then Bruce* `knows` *B-AM* `said` *Bruce canAccess*(*code*, *parameters*)

Proving invariant relations of this nature is hard in general, as we will find out in the next section.

6 Complexity Analysis of Safety Properties

6.1 Unrestricted Communication in Restricted DKAL

In this subsection we consider a restricted, yet useful fragment $DKAL^-$ of DKAL which is DKAL without variables, without $+$, `canActAs, canSpeakAs, exists`, `tdon`* and without substrate constraints in the rules. Note that attributes allow us to deal with roles even in the absence of `canActAs`, `canSpeakAs`. As opposed to `canActAs` and `canSpeakAs`, attributes take parameters. In the restricted fragment, we have three forms of infons. Firstly, there are *primitive infons* of the form Attribute$(a_1, \ldots, a_r)$ where elements a_i are regular. Secondly, we have *said infons* of the form p `said` x. Thirdly, we have *tdon infons* of the form p `tdon` x.

The communication is unrestricted in the following sense. We are interested in the consequences of the core policy of a given principal, say A, without restricting information that A can learn from other principals.

To simplify exposition, we let Σ_A be the set of said infons derived by the COM rule from Δ_A and communications to A by other principal. For example, if $(B$ `to` $A : x) \in \Delta_A$ and $(A$ `from` $B : x) \in \Delta_B$ then $(B$ `said` $x) \in \Sigma_A$. From the perspective of knowledge, it is sufficient to work with Σ_A, since the `from` and `to` statements by themselves do not add any knowledge for A; it is only when we combine corresponding `from` and `to` statements by the COM rule that additional knowledge is derived.

One interesting property is the reachability relation which is defined as follows: an infon I is reachable from a core policy Π_A if there exists a set Σ_A such that $\Pi_A \cup \Sigma_A \vdash A$ `knows` I.

Theorem 2 (Reachability)
The reachability relation for $DKAL^-$ is polynomial time computable.

Proof. Without loss of generality, we may assume that initially Σ_A is empty. We transform the total policy $\Pi_A \cup \Sigma_A$ of A in ways that preserve the reachability status of the given infon I.

Remove Some Said infons from Core Policy

1. Remove any rule $(p$ `said` $x) \leftarrow x_1, \ldots, x_k$ from Π_A and add the assertion $(p$ `said` $x)$ to Σ_A.
2. If an infon p `said` x occurs in the body of a rule R in Π_A, remove the infon from the body of R and and add the assertion $(p$ `said` $x)$ to Σ_A.

The new policy $\Pi_A^1 \cup \Sigma_A^1$ has the following property. In any core rule $y_0 \leftarrow y_1, \ldots, y_m$, every infon y_i is either primitive or a tdon infon. Obviously, $\Pi_A \cup \Sigma_A^1 \vdash A$ `knows` I if and only if $\Pi_A^1 \cup \Sigma_A^1 \vdash A$ `knows` I.

Augment tdon Rules. For any core rule p_1 `tdon` p_2 `tdon` $\ldots$`tdon` p_j `tdon` x $\leftarrow x_1, \ldots, x_k$ where x is primitive or a said infon, do the following. Add core rules

$$p_2 \texttt{ tdon } p_3 \texttt{ tdon } \ldots p_j \texttt{ tdon } x \leftarrow p_1 \texttt{ tdon } p_2 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$p_3 \texttt{ tdon } \ldots p_j \texttt{ tdon } x \leftarrow p_2 \texttt{ tdon } p_3 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$\ldots$$
$$p_j \texttt{ tdon } x \leftarrow p_{j-1} \texttt{ tdon } p_j \texttt{ tdon } x$$
$$x \leftarrow p_j \texttt{ tdon } x$$

Also add the following assertions to Σ_A:

$$p_1 \texttt{ said } p_2 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$p_2 \texttt{ said } p_3 \texttt{ tdon } \ldots p_j \texttt{ tdon } x$$
$$\ldots$$
$$p_{j-1} \texttt{ said } p_j \texttt{ tdon } x$$
$$p_j \texttt{ said } x$$

This way we obtain $\Pi_A^2 \cup \Sigma_A^2$. Observe that $\Pi_A \cup \Sigma_A^2 \vdash A \texttt{ knows } I$ if and only if $\Pi_A^2 \cup \Sigma_A^2 \vdash A \texttt{ knows } I$.

Any said infon $p \texttt{ said } x$ is reachable from any Π_A with a witness Σ_A asserting $(A \texttt{ from } p : x)$ and Σ_p asserting $(p \texttt{ to } A : x)$. So we may assume that I is primitive or a tdon infon. Recall that the DKAL derivability relation is polynomial time. Thus it suffices to prove that I is reachable from Π_A if and only if $\Pi_A^2 \cup \Sigma_A^2 \vdash A \texttt{ knows } I$. The if part is obvious. It remains to prove the only if part.

We claim that A cannot derive a new primitive or tdon infon as a result of learning any additional information, that is as a result of extending Σ_A^2 with any additional assertion R of said infon $p \texttt{ said } x$ (after using the COM rule with corresponding `from` and `to` assertions). Indeed, there are no said infons in the bodies of the rules in $\Pi_A^2 \cup \Sigma_A^2$. So adding R triggers no rule. Further, if $\Pi_A^2 \cup \Sigma_A^2 \vdash p \texttt{ tdon } x$ (x can be an infon of any type) then, by the second transformation above, $\Pi_A^2 \cup \Sigma_A^2 \vdash x$. So adding R does not help either. But these are the only ways that a said assertion can be helpful.

We say that $(A \texttt{ knows } I_1) \longrightarrow (A \texttt{ knows } I_2)$ is an invariant for a core policy Π_A if $\Pi_A \cup \Sigma_A \vdash A \texttt{ knows } I_1$ implies $\Pi_A \cup \Sigma_A \vdash A \texttt{ knows } I_2$ for any set Σ_A. This gives rise to a binary invariant relation $\mathrm{Inv}(I_1, I_2)$ for $DKAL^-$.

Theorem 3 (Invariant Checking)
Computing the invariant relation for $DKAL^-$ is `coNP`-complete.

The proof is in [8].

6.2 Restricted Communication in Unrestricted DKAL

In this subsection we analyze the complexity of deciding classes of safety properties for unrestricted DKAL. The core policy does not have any `from` rules.

The workflow interacts with the DKAL inference machine through `from` assertions only. The content of `from` assertions is unrestricted within the bounds of syntactic well-formedness.

Since the `to` assertions can only be derived from the core policy, we need to look at the union of the policies of all the principals $\bigcup_\alpha \Pi_\alpha$ in order to analyze reachability and invariant checking. We assume that the number of variables in each rule in each policy is fixed a-priori and that α ranges over all principals in the system. We have the following results which are proved in [8].

Theorem 4 (Reachability Checking)
The reachability relation for DKAL is polynomial time computable.

Theorem 5 (Invariant Checking)
Computing the invariant relation for DKAL is `coNP`*-complete.*

References

1. Abadi, M., Burrows, M., Lampson, B., Plotkin, G.: A calculus for access control in distributed systems. ACM Transactions on Programming Languages and Systems 15(4), 706–734 (1993)
2. Becker, M.Y., Fournet, C., Gordon, A.D.: SecPAL: Design and Semantics of a Decetralized Authorization Language. In: 20th IEEE Computer Security Foundations Symposium (CSF), pp. 3–15 (2007)
3. Blass, A., Gurevich, Y.: Two Forms of One Useful Logic: Existential Fixed Point Logic and Liberal Datalog. Bulletin of the European Association for Theoretical Computer Science 95, 164–182 (2008)
4. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: Proc. 1996 IEEE Symposium on Security and Privacy, pp. 164–173 (1996)
5. DeTreville, J.: Binder, a Logic-Based Security Language. In: IEEE Symposium on Security and Privacy, pp. 105–113 (2002)
6. Gurevich, Y., Neeman, I.: DKAL: Distributed-Knowledge Authorization Language. In: 21st IEEE Computer Security Foundations Symposium (CSF 2008), pp. 149–162 (2008)
7. Gurevich, Y., Neeman, I.: DKAL 2 — A Simplified and Improved Authorization Language. Microsoft Research Tech Report MSR-TR-2009-11 (February 2009)
8. Gurevich, Y., Roy, A.: Operational Semantics for DKAL: Application and Analysis. Microsoft Research Tech Report MSR-TR-2008-184 (December 2008)
9. Li, N.: Delegation Logic: A Logic-Based Approach to Distributed Authorization, Ph.D. thesis, New York University (September 2000)
10. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation Logic: A Logic-Based Approach to Distributed Authorization. ACM Trans. on Information and System Security (TISSEC) 6(1), 128–171 (2003)
11. Li, N., Winsborough, W.H., Mitchell, J.C.: Beyond Proof-of-Compliance: Safety and Availability Analysis in Trust Management. In: Proceedings of 2003 IEEE Symposium on Security and Privacy, May 2003, pp. 123–139 (2003)