DISTRIBUTED-KNOWLEDGE AUTHORIZATION LANGUAGE JANUARY 2008 REVISION

YURI GUREVICH AND ITAY NEEMAN

Abstract. DKAL is an expressive declarative authorization language based on existential fixed-point logic. It is considerably more expressive than existing languages in the literature, and yet feasible. Our query algorithm is within the same bounds of computational complexity as e.g. that of SecPAL. DKAL's distinguishing features include

- explicit handling of knowledge and information,
- targeted communication that is beneficial with respect to confidentiality, security, and liability protection,
- the flexible use and nesting of functions, which in particular allows principals to quote (to other principals) whatever has been said to them,
- flexible built-in rules for expressing and delegating trust,
- information order that contributes to succinctness.

January 2008 revision of Microsoft Research Technical Report MSR-TR-2007-116 of August 2007.

Contents

1. Introduction

As computing systems and networks become larger and more distributed, the problem of authorization turns more and more involved. Rights are not necessarily assigned and maintained by central authorities, but may instead result from credentials issued by many different entities. Authorization policies must handle not only permissions for the end user, but also permissions for entities to issue credentials. These permissions themselves may depend on other credentials. Policies must handle all this in a manner which is clearly understood, amenable to analysis, modular so as to afford the greatest stability in a changing environment, and secure.

Logic based declarative trust management languages have been advanced as a solution to this problem. Policies written in these languages can serve as a base, a "legal manifesto" of sorts, from which specific permissions are derived using imperative applications. The languages allow writing high-level policy rules in human readable form. They generally have built-in vocabulary for expressing trust and delegation, so that a policy can be modular and distributed over many different entities. Their declarative form and firm semantics for deriving permissions from the policy rules allow for analyses of policies, see for example [19].

Several languages have been developed in recent years, and those related to this paper are surveyed in Section 8. Our late genealogy consists primarily of Delegation Logic, Binder, and SecPAL. The language Binder [11] builds very directly on Datalog, extending it with an import/export construct `says` that connects Datalog programs maintained by different principals, and makes the issuer of an imported assertion explicit. A principal A may for example condition a Datalog rule on B `says` `employee`$(C, \mathit{Employer})$. The rule will fire when principal B exports `employee`$(C, \mathit{Employer})$. A may express trust in B on `employee`(x, y) using a Datalog rule `employee`(x, y) `:-` B `says` `employee`(x, y). Delegation Logic [15, 16] does not have explicit issuers for all assertions, but on the other hand it has additional constructs, specific to the demands of authorization languages, including ones for delegations, representations, and thresholds. Delegation Logic assertions are reduced to Datalog rules, for example rules similar to the one expressing trust above, for the purpose of execution. SecPAL [4, 12] has both explicit issuers and specific constructs designed with distributed systems authorization policy in mind. The number of constructs is deliberately kept low, but the language is expressive and captures many standard authorization scenarios, including discretionary and mandatory access control, role hierarchies, separation of duties, threshold-constrained trust, attribute-based delegation, and delegation controlled by constraints, depth, and width, see [4, Section 5]. The semantics of the language is defined directly, using a few very condensed deduction rules. SecPAL policies go beyond Datalog in two ways. They allow constraints, and they allow facts obtained by nesting `can say` (a.k.a. `can say`$_\infty$) and `can say`$_0$. SecPAL policies are reduced to safe Constraint Datalog programs by converting nested `can say`$_d$ facts to relations of arity dependent on the nesting depth, which is finite in any given policy and can only decrease in deductions. Nested `can say`$_0$ facts are used for bounded depth delegation, and the SecPAL deduction laws give rise to semantics that prevents any circumventing of the delegation bound.

If an authorization policy is to be amenable to analysis, then certainly the question of which permissions it leads to should be decidable, preferably in polynomial time. Binder, Delegation Logic, and SecPAL all have polynomial time decision algorithms, reached through

the reductions to Datalog and Constraint Datalog, and relaying on syntactic safety conditions that are similar to those typically imposed in Datalog programs to ensure termination.

This paper takes its departure in SecPAL. We present a language that addresses a problem of information leakage in Binder, Delegation Logic, and SecPAL, strengthens the delegation rule, introduces an information order that contributes to strong succinct semantics, and enriches the structure of authorization facts allowing the nesting of arbitrary constructor functions, and in particular the nesting of quotations. All this is done while maintaining two important properties of previous logic based trust management languages: human readability and polynomial time decidability. The new and expressive language is called Distributed-Knowledge Authorization Language, in short DKAL. An overview of the language is given in the next section, with specific subsections dedicated specifically to each of the matters mentioned above: preventing information leakage, delegation, quotations, the information order, and termination and indeed polynomial decidability in the presence of constructor functions. Here let us only comment that prevention of information leakage is achieved using targeting of communication. Similar targeting was recently added to Binder in [2]. Section 2 is written at the level of an introduction to the language and to the problems it addresses.

Later sections present the language in more detail (Sections 3 and 4), give examples, partly to illustrate usage and partly to justify certain choices we made in designing the information order rules (Section 5), present the proof of polynomial time decidability (Section 6), present a translation of SecPAL into DKAL (Section 7), and conclude with a discussion including related work (Section 8). DKAL rests on the firm foundation of Existential Fixed Point Logic, which we recall in Appendix A. We tried to make this paper self-contained. In particular, we do not presume that the reader is familiar with SecPAL, but familiarity with SecPAL is beneficial in a few places where we discuss SecPAL.

Acknowledgements. We gratefully acknowledge collaboration with M. Paramasivam on earlier stages of the project, conversations with Slava Kavsan, and comments of Moritz Becker, Nikolaj Bjorner and Andreas Blass.

2. DKAL, a bird's eye view

We begin with a bird's eye view of DKAL using simple scenarios to illustrate its features. Precise definitions will be given in later sections. The aspects we discuss are: existential fixed point logic; distributed knowledge and targeted communication; delegation and trust; protection against information leakage; bounded depth delegation and connection with SecPAL; quotations and their use in allowing greater policy modularity; the information order; termination; and computational complexity.

2.1. **Existential fixed-point logic (EFPL).**

EFPL was introduced in [7] and has attractive model theory. It is obtained from first-order logic as follows: first restrict first-order logic to its existential fragment and then extend the existential fragment by means of the least fixed-point operator.

The least fixed-point operator enables induction. For example, given the infinite binary tree where unary functions `left` and `right` are free constructors, a logic program

$$\begin{aligned} &T(\texttt{left}(x), \texttt{right}(x)) \\ &T(\texttt{right}(x), \texttt{left}(y)) \leftarrow T(x, y) \\ &\qquad T(x, z) \leftarrow (T(x, y) \wedge T(y, z)) \end{aligned}$$

computes a partial order T that is the lexicographical order at every level of the tree. The structure at which a given program operates is the *substrate structure* or just a *substrate*, and the relations computed by the program are *superstrate relations*. In the example, the binary tree is the substrate, and T is a superstrate relation.

EFPL reduces to its Prolog-like fragment where the least-fixed operator is not iterated and every new relation that is defined as a superstrate relation is given by a logic program over the substrate structure. EFPL may be also reduced to a form of Datalog with constraints. For example the program above can be written in Datalog with constraints as follows:

$$\begin{aligned} T(u, w) &\leftarrow u = \texttt{left}(x) \wedge w = \texttt{right}(x) \\ T(u, w) &\leftarrow u = \texttt{right}(x) \wedge w = \texttt{left}(y) \wedge T(x, y) \\ T(x, z) &\leftarrow T(x, y) \wedge T(y, z) \end{aligned}$$

There are many different forms of Datalog with constraints, distinguished by the safety restrictions they place on programs to ensure termination. The form obtained from EFPL with the safety conditions imposed by DKAL is new. We'll say more on how DKAL guarantees termination in Subsection 2.8 below.

2.2. **A user centric example.** Since SecPAL is naturally translated into DKAL, all the varied scenarios of [3, Section 5] are expressible in DKAL. So we start here with an example of a different kind, a user centric example. The example demonstrates the basics of DKAL, in particular how trust and delegation are expressed.

Alice would like to download Article from Repository in course of her work for Fabricam. Repository lets Fabricam employees download content with no constraints. Fabricam in turn requires that its employees respect intellectual property. Figure 1 shows how Alice verified her right to download Article.

Alice bought the right at an online store Chux (an allusion to Chuck's). Chux told her that she can download Article; this is represented by assertion A1 in Figure 1. In the formal model we compute a superstrate relation `knows`, and the assertion A1 leads to the instance K1 of that relation. The expression `Alice canDownload Article` denotes an *infon*, a piece of information, and so does `Chux said Alice canDownload Article`. The relation `knows` is of type Principal × Info. Note that from assertion A1 Alice learns only that `Chux said Alice canDownload Article`, not that `Alice canDownload Article`.

Alice noticed that the copyright for Article belongs to Best Publishing House; hence the assertion A2 where `tdOn` stands for *is trusted on*. The expression `Best tdOn Alice canDownload Article` denotes yet another infon, and assertion A2 leads to instance K2 of `knows` in the formal model.

The intended meaning of p `tdOn` x is given by two rules. One is C1 which states that a principal a knows x if she knows that some principal p said x and that p is trusted on x. We get to the other rule shortly.

(A1) Chux: (Alice canDownload Article) to Alice

(A2) Alice: Best tdOn Alice canDownload Article

(A3) Best: (Chux tdOn p canDownload Article) to p

(C1) a knows $x \leftarrow a$ knows p said x, a knows p tdOn x

(C2) a knows p tdOn $(q$ tdOn $x) \leftarrow a$ knows p tdOn x, a knows q exists
a knows p tdOn $(q$ $\texttt{tdOn}_0$ $x) \leftarrow a$ knows p tdOn x, a knows q exists

(K1) Alice knows Chux said Alice canDownload Article

(K2) Alice knows Best tdOn Alice canDownload Article

(K3) Alice knows Best said (Chux tdOn Alice canDownload Article)

(K4) Alice knows Chux exists

(K5) Alice knows Best tdOn (Chux tdOn Alice canDownload Article)

(K6) Alice knows Chux tdOn Alice canDownload Article

(K7) Alice knows Alice canDownload Article

FIGURE 1. User centric delegation example

Unfortunately Alice does not know whether Chux can be trusted on Alice canDownload Article, and Best, who is trusted, did not say that Alice canDownload Article. So Alice cannot yet conclude that she is allowed to download Article. Alice contacts Best who authorized Chux to sell download rights to Article and who has in its policy the assertion A3 (with a free, unconstrained variable p). As a result Alice learns K3.

The infon p tdOn x expresses not only trust in p on x, but also a permission for p to delegate the trust. (There is a way to express non-delegatable trust, using $\texttt{tdOn}_0$ instead of tdOn. The distinction between tdOn and $\texttt{tdOn}_0$ is inherited from SecPAL and will be addressed later.) The right to delegate is captured by the double rule C2; only the first line is relevant to the current example. If a knows that p tdOn x and that q exists then a knows that p is also trusted on q tdOn x, and this allows p to delegate the trust to q. The restriction that a knows (the existence) of q is a safety condition that prevents the knowledge of a from exploding with irrelevant details. We'll say a little more about the rule that leads to knowledge of infons of the form q exists in Subsection 2.8; here it suffices to say that the rule applies to K1 and results in K4.

Applying rules C1 and C2 to K1–K4, Alice obtains K5–K7. Having deduced K7, Alice approaches Repository, and downloads Article.

2.3. **Examples related to info leakage.** A naive dramatization in Figure 2 illustrates a potential information leakage (unless plugged by the implementation) in SecPAL. (A similar risk exists in other languages, including Binder [11], Delegation Logic [16], and the RT framework [18].) The department of Special Operations of some intelligence agency designates John Doe to be a secret Agent; hence assertion S1. Bob, who is just a receptionist, wants to find out who secret agents are. He does not dare to pose that query (and suspects that the system would not allow him to); instead he asserts S2 and S3 where spot 97 is one of the parking spots over which he has the authority, e.g. a visitor spot. It follows from S1

(S1) `SpecialOperations says John Doe isSecrAgent`

(S2) `Bob says SpecialOperations can say` p `isSecrAgent`

(S3) `Bob says` p `canParkInSpot 97 if` p `isSecrAgent`

FIGURE 2. Secret agent information leakage

and S2 that `Bob says John Doe isSecrAgent`. By posing an "innocent" query about who can park in spot 97 Bob discovers in particular that John Doe is a secret agent.

The problem can be addressed on the level of implementation, for example by attempting to separate confidential and non-confidential information (which is easier said than done; both may be necessary to derive certain permissions), but the right way to address the problem is at the authorization-language level. DKAL solves the problem by means of targeted communication and the distinction between knowing and saying. The analog of the naive dramatization would not work in DKAL as assertion S1 would be targeted to an audience that excludes Bob.

Let's consider a slightly less naive example. Modify the scenario of §2.2 by replacing assertion A1 with the assertions in Figure 3. Chux compiles payment statistics of customers and rates them. Customers rated "perfect" get the download authorization immediately upon authorizing a proper payment to Chux, even before the funds are received. The rating is managed by accounts.Chux. It is intended that customers know nothing about the rating system or their ratings or other customers' ratings. Chux makes assertion A4 with three conditions. A conditions is an expression of type Info or a substrate constraint. In this case the first two conditions are infon expressions, and the third is a constraint using a substrate function `price`. Implicitly the assertion has also safety conditions, described in Subsection 2.8 and defined precisely in Subsection 4.1, that restrict the ranges of variables; the safety constraints apply also to assertions A6 and A7. According to assertion A5, accounts.Chux rated Alice "perfect"; note that the assertion is targeted only to Chux. According to assertions A6 and A7, Chux trusts accounts.Chux on payment ratings and trusts the customers on payment authorization. The price of Article is $40. When Alice decides to purchase Article, she makes the assertion A8. Assertions A4–A8 lead Chux to communicate the infon `Alice canDownload Article` to Alice; as above Alice can proceed to verify her right to download Article.

No infon of the form p `hasPayRate R` is communicated to Alice, and a probing attack such as the one in Figure 2 does not work. The DKAL parallel of S2 here is assertion `Alice: accounts.Chux tdOn` p `hasPayRate Perfect`. The assertion is harmless, since A5 makes the infon `accounts.Chux said Alice hasPayRate Perfect` known only to Chux, not to Alice. The confidentiality of pay ratings of other principals is similarly protected.

The targeting of communication is beneficial also with respect to liability. Suppose that an agency A of state S_1 issues David a document, addressed to S_1 wine shops, that allows them to sell wine to David. If David buys alcohol from a wine shop in state S_2 and if this violates the law of S_2, agency A is not liable because it addressed the documents to wine shops in S_1, not in S_2.

Audience restrictions can be communicated by means of SAML [23], see specifically [10, §2.5.1.4]. (The issue is addressed in the SecPAL implementation as well.) While the audience

(A4) Chux: (a canDownload s) to a ←
a authorized \$$k$ to Chux for s,
a hasPayRate Perfect, price(s)=k.

(A5) accounts.Chux: (Alice hasPayRate Perfect) to Chux

(A6) Chux: accounts.Chux tdOn a hasPayRate e

(A7) Chux: a tdOn_0 a authorized \$$k$ to Chux for s

(A8) Alice: (Alice authorized $40 to Chux for Article) to Chux

FIGURE 3. Confidentiality example

restriction may be helpful with respect to liability, the SAML audience field does not solve the problem on Figure 2. Indeed, if the fact John Doe isSecrAgent is modified with an audience restriction then all that Bob has to do is to use the modified fact in S2.

In probing attacks of the kind illustrated in Figure 2, principals that are allowed to authorize some permissions leverage the authority to learn information they are not meant to know. One way to thwart such probing attacks is to disallow conditional assertions by "outsiders" (like Bob), as in Cassandra [5, 6], but this is too restrictive. One may filter out some conditional assertions on a case by case basis at the implementation level, but this ad-hoc approach makes it hard to reason about security. Yet another way is to compartmentalize facts to the extent possible and handle requests using primarily the relevant compartment policy. But there are limits to compartmentalization (unless you really have a union of essentially disjoint policies), principals still can probe facts in their compartments, and the approach does not make reasoning about security easy.

By targeting communication and separating knowing from saying, DKAL solves the problem at the level of the authorization language, so that information does not have to be compartmentalized a priori, and conditional assertions do not have to be filtered out. Of course DKAL does not prevent information from leaking as a result of negligence. For example, in the pay rate scenario, Chux may accidentally target Bertha's pay rating to Alice. But that is a very different story.

2.4. **Use of functions.** In Datalog, with or without constraints, the symbols of recursively defined relations are applied only to (tuples of) variables and individual constants, and the (Constraint) Datalog based authorization languages inherit the restriction on the use of functions. In contrast, existential fixed-point logic allows free use of function symbols, and EFPL based DKAL makes intensive use of functions, both user-specific and built-in. Function symbols routinely appear in the heads of rules, and typically our functions are free constructors. The flexible use of function comes for a price. The proof of program termination, let alone complexity proofs, become much harder.

The built-in free-constructor functions include said and tdOn. Function said enables (possibly nested) quotations in authorization policies, which leads to greater flexibility in designing more modular policies. Suppose for example that Chux (that appeared in Figures 1 and 2) has several discount plans, and that employees of Fabricam participate in discount plan 5X4302. To obtain the discount, they must present a signed certificate from Fabricam stating that they are employees. Chux relies on a cryptographic server Crypto to verify that

(A9) Chux: Crypto tdOn r said q is an employee of r

(A10) Chux: Fabricam tdOn q is an employee of Fabricam

(A11) Chux: q can take discount 5X4302 $\leftarrow$ q is an employee of Fabricam

(A12) Chux: a tdOn augm(a authorized \$$k$ to Chux for s, c)

(A13) Chux: a authorized \$$k$ to Chux for s $\leftarrow$
augm(a authorized \$$k$ to Chux for s, c),
authentic(a, a authorized \$$k$ to Chux for s, c)

(A14) Alice: augm(Alice authorized \$40 to Chux for Article, C) to Chux

(A15) Chux: Crypto tdOn_0 r said q is an employee of r

(A16) Crypto:_0 (Fabricam said Chris is an employee of Fabricam) to Chux

(A17) Crypto:_0 (r said q is an employee of r) to Chux $\leftarrow$
Crypto2 said r said q is an employee of r

(C3) a knows x $\leftarrow$ a knows p said_0 x, a knows p tdOn_0 x

(K8) Chux knows Crypto said_0 Fabricam said Chris is an employee of Fabricam

(K9) Chux knows Fabricam said Chris is an employee of Fabricam

FIGURE 4. Use of functions, and restricted delegation

the signed statements are authentic. The system should be designed so that Crypto just verifies authenticity. Crypto's actions should not depend on Chux's policy on discounts, so that Chux's policy could be changed without requiring a change in Crypto's behavior.

Using quotations, Chux can make assertion A9 in Figure 4. Crypto acts as a "dumb" server, merely decrypting the statements it receives, and passing them on to Chux. Policy, for example assertions A10 and A11, is the prerogative of Chux. In this case, the end effect (of authorizing discount to Fabricam employees) could be achieved without quotations. Chux could trust Crypto on q is an employee of Fabricam, and Crypto in its own policy could trust Fabricam on this. The issue here is not just achieving the end effect, but the flexibility to concentrate the policy at one place.

DKAL's vocabulary may be extended by user-introduced functions and relations. We already saw function price in §2.3. Other typical user-introduced functions and relations relate to time, various directory structures, basic arithmetical operations, etc. DKAL also permits user-introduced functions that take attribute or infon values. To demonstrate this, modify the confidentiality example above by replacing assertions A7 and A8 with assertions A12–A14 in Figure 4. Chux does not simply accept infon a authorized \$$k$ to Chux for s from customer a, but requires that the infon comes with a certificate, signed using a's private key. (Chux will need the certificate to obtain the funds from a bank.)

We assume here a given (that is substrate) relation authentic(a, x, c) meaning that c is a certificate of infon x signed with the private key of a. Given an infon x and string c, function augm(x, c) (an allusion to "augment") produces a new infon. When Alice wishes to purchase Article, she makes assertion A14, where C is a certificate of the infon Alice authorized \$40 to Chux for Article, which Alice produced and signed using her private

key. Then, due to assertions A12 and A13, `Chux knows Alice authorized $40 to Chux for Article`, and then, as in §2.3, Alice receives an authorization to download Article.

2.5. **Restricted delegation.** One of the major advances of SecPAL [3] is the mechanism of restricted delegation. We adapted that mechanism to DKAL. DKAL has two kinds of infons expressing trust, p `tdOn` x, and p $\texttt{tdOn}_0$ x. The trust given by the former is delegatable; the trust given by the latter is not. To illustrate the use of non-delegatable trust, replace assertion A9 in Figure 4 with assertion A15 in Figure 4. The new assertion expresses non-delegatable trust in Crypto on r `said` q `is an employee of` r. Suppose that Crypto is given a signed certificate from Fabricam attesting that Chris is a Fabricam employee. After authenticating the certificate, Crypto produces assertion A16. The subscript 0 in A16 signifies restricted communication; more on this in the next paragraph. Assertion A16 leads to knowledge K8, with the subscript 0 on the first said. K8 and assertion A15 give K9 by means of rule C3.

The delegation rule C2 has delegatable trust assumed in its body and cannot be applied to A16, so Crypto cannot directly delegate the trust to others. He may attempt to circumvent the prohibition, for example by placing assertion A17. It seems that by saying the appropriate thing, Crypto2 enables A16. But the attempt fails because assertion A17 is restricted. The precise meaning of restricted assertion involves relation $\texttt{knows}_0$, read knows internally. p knows x internally if this follows from assertions placed by p himself, with no dependence on assertions placed by other principals. Restricted assertions can be conditioned only upon internal knowledge (see Subsection 4.1) while A17 is based on communication from Crypto2 to Crypto.

We sometimes write $\texttt{knows}_\infty$, $\texttt{said}_\infty$, and $\texttt{tdOn}_\infty$ for `knows`, `said`, and `tdOn`. The distinction between $\texttt{knows}_\infty$ and $\texttt{said}_\infty$ on one side and $\texttt{knows}_0$ and $\texttt{said}_0$ on the other side is similar to SecPAL distinction between $\mathcal{AC}, \infty \models A$ `says` x and $\mathcal{AC}, 0 \models A$ `says` x, and is used here to the same effect, namely preventing principals from circumventing non-delegatability. Delegations of arbitrary bounded depth can be obtained by nesting $\texttt{tdOn}_0$ in the head of the assertion delegating the right. SecPAL examples on bounded depth delegation, see e.g. [4, §5] become DKAL examples via the embedding of SecPAL into DKAL.

2.6. **SecPAL-to-DKAL translation.** In Section 7 we give a natural translation of SecPAL into a "crippled" version of DKAL, called Open DKAL, where in particular the targeting of communication is removed. Under this translation, every query with a positive answer in SecPAL gets a positive answer in Open DKAL. In addition, due to DKAL's powerful delegation rules, some additional SecPAL queries get positive answers in Open DKAL. Thus more justifiable requests expressible in SecPAL get positive answers in DKAL. If one were to remove DKAL's delegation rules, the translation from SecPAL to Open DKAL would become exact.

2.7. **Information order.** Rules C1–C3 have a common aspect: a principal a knows some infon x because a knows some other infons $y_1, \ldots, y_k$. The information order x `ensues` y (symbolically $x \leq y$) on infons extracts the common aspect. (We resurrect the obsolete transitive meaning of ensue [24].) Ideally, the meaning of $x \leq y$ would be that all information of x is present in y but this leads to undecidability. The actual order is a constructive approximation of the ideal one. The mediating rules KMon and KSum on Figure 5 express the common aspect of C1–C3 and their counterparts for $\texttt{knows}_0$. Rule KMon states that

(KMon)	$a\ \texttt{knows}_d\ x$	$\leftarrow$	$a\ \texttt{knows}_d\ y,\ x \leq y$
(KSum)	$a\ \texttt{knows}_d\ x_1 + x_2$	$\leftarrow$	$a\ \texttt{knows}_d\ x_1,\ a\ \texttt{knows}_d\ x_2$
(TrustApp)	$x \leq p\ \texttt{said}_d\ x\ \texttt{+}\ p\ \texttt{tdOn}_d\ x$		
(Del)	$p\ \texttt{tdOn}\ (q\ \texttt{tdOn}_d\ x) \leq p\ \texttt{tdOn}\ x\ \texttt{+}\ q\ \texttt{exists}$		
(Trust0∞)	$p\ \texttt{tdOn}_0\ x \leq p\ \texttt{tdOn}\ x$		
(SaidMon)	$p\ \texttt{said}_d\ x \leq p\ \texttt{said}_d\ y$	$\leftarrow$	$x \leq y$
(Exists)	$r\ \texttt{exists} \leq x$	$\leftarrow$	$r\ \texttt{regcomp}\ x$

FIGURE 5. Select House Rules

knowledge of x is a consequence of knowledge of y if x ensues y. Rule KSum introduces infon addition operation of type Info×Info → Info, and the rule states that knowledge of $x_1 + x_2$ is a consequence of knowledge of both x_1 and x_2. Each of KMon and KSum is a *double rule*, with $d \in \{0, \infty\}$. We often use this double rule notation.

The content of rules C1 and C3 is now expressed succinctly by ensue double rule TrustApp. Similarly the content of rule C2 is expressed by ensue double rule Del. Rules KMon–Del are *house rules* of DKAL. Rules C1–C3 are not house rules; they are consequences of house rules.

The inclusion of the information order allows creating a rich structure of information with easily understood rules. For example rule Trust0∞ expresses the fact that non-delegatable trust is a consequence of delegatable trust. The inclusion of the information order also allows for easily expressing strong quotation semantics. The deceptively simple rule SaidMon incorporates consequences of speeches into the calculation of knowledge, so that, for example $p\ \texttt{said}\ q\ \texttt{tdOn}_0\ x$ ensues $p\ \texttt{said}\ q\ \texttt{tdOn}\ x$. DKAL thus has very strong semantics for quotations, computing not only principals' speeches, but also their implied consequences. The rule could not be expressed as a single rule without the information order .

2.8. **Termination.** The flexible use of functions makes DKAL closer to Prolog than to Datalog. It is of course only too easy to write a non-terminating program in Prolog. But DKAL is carefully calibrated to ensure the termination of an algorithm that computes answers to queries.

We split the universe of the state into regular and synthetic elements. Regular elements may be principals, time moments, time intervals, files, directories, domain names, etc. Synthetic elements are generated from regular ones by means of free-constructor functions whether built in, such as $\texttt{said}_d$ and $\texttt{tdOn}_d$, or user-defined. In policy assertions, variables range over regular elements only.

The space over which variables of any particular assertion range is limited further using infons of the form $p\ \texttt{exists}$. An infon $p\ \texttt{exists}$ gives only the information that there is a regular element p; it does not give any information about p other than this. This intuition is expressed precisely by house rule Exists in Figure 5. The intended meaning of $r\ \texttt{regcomp}\ x$ is that r appears in x in an essential way; the precise definition is given in 3.1.2.

A DKAL assertion

$$\texttt{A:}_d \quad x \leftarrow x_1, \ldots, x_n, con,$$

where $x, x_1, \ldots, x_n$ are infon expressions and *con* is a substrate constraint, leads to the EFPL rule

$$\begin{aligned} \texttt{A knows}_d\ x \leftarrow\ & \texttt{A knows}_d\ x_1, \cdots, \texttt{A knows}_d\ x_n, \\ & \texttt{A knows}_d\ t_1\ \texttt{exists}, \cdots, \texttt{A knows}_d\ t_k\ \texttt{exists}, \\ & \mathit{con} \end{aligned}$$

where the list $t_1, \ldots, t_k$ consists of all the variable of the assertion and all the non-ground regular components of the head x. We say that $t_1, \ldots, t_k$ are A-*bounded*, that is bounded to vary only over objects whose existence is known to A. The inclusion of the infons t_i exists in the body of the rule is a semantic safety condition that prevents the knowledge of A from exploding. It is similar to the syntactic safety condition in SecPAL, in that it ensures that all variables in *con* are instantiated at the time of evaluation and that regular elements of the head occur in the body. DKAL assertions of the form

$$\texttt{A:}_d \quad x\ \texttt{to}\ p \leftarrow x_1, \ldots, x_n, \mathit{con}$$

are subject to a similar safety condition, but the variable p is not required to be A-bounded.

The semantic safety condition allows us to show that only the regular elements that are explicitly mentioned in the policy are relevant and need to be considered as possible values for variables when evaluating the policy. Since the policy is finite, the number of relevant regular elements is finite.

Things are much more involved with synthetic elements. While assertions have only regular variables, many house rules of DKAL have variables ranging over synthetic elements, in particular over infons. We prove that the number of synthetic elements needed to evaluate a given query under a given policy is finite. The proof is elaborate and uses the nature of the DKAL house rules.

The proofs of finiteness mentioned to above are presented in Section 6, and are used there to show that every DKAL query can be answered by an algorithm in finite time.

2.9. **Worst-case complexity.** A basic query has the form p knows $t(v_1, \ldots, v_k)$ or p $\texttt{knows}_0$ $t(v_1, \ldots, v_k)$ where p is a ground (that is with no variables) expression of type Principal, t is an expression of type Info, and the variables v_i range over regular elements. In Section 6, we construct an algorithm that works as follows over any fixed substrate: given an authorization policy $\mathcal{A}$ and a basic query Q, the algorithm computes the complete answer to Q under $\mathcal{A}$. Here the complete answer is the set of tuples $(b_1, \ldots, b_k)$ of regular elements known to p and of appropriate types such that the principal p knows the infon $t(b_1, \ldots, b_k)$ under the authorization policy $\mathcal{A}$. The finiteness results mentioned in the previous subsection are crucial for the termination of the algorithm. They are not by themselves sufficient for proving the time complexity bounds mentioned below; for this it is important also to have a polynomial bound on the number of synthetic elements relevant to the computation.

The runtime of our algorithm is bounded by a polynomial in $\ell^{\,\delta+1+w}$ where

- ℓ is the length of the input, that is the length of $\mathcal{A}$ plus the length of Q,
- δ is the maximal depth to which said and $\texttt{said}_0$ (possibly mixed) are nested in the policy and query expression, and
- w is the maximal number of variables (the "width") in any assertion or in the query.

In the all-important case when δ and w are bounded, the runtime is polynomial in ℓ. The complexity proof is rather involved but one does not have to master the proof in order to use the algorithm.

In §4.4, we introduce more general queries: first-order queries, in particular allowing negations, about the knowledge of a single principal. We call them single-principal-centric queries. The basic-query algorithm is generalized to work with arbitrary single-principal-centric queries. The time complexity results remains valid.

The availability of negations in queries can be used for conflict resolution at the decision point. For example, in a deny-override system, with read guard RG, read access to File 13 would be given to the users in the answer to the query: `RG knows` p `hasReadAccessTo File 13` $\wedge \neg$ (`RG knows` p `deniedAccessTo File 13`).

3. State of Knowledge

We use a little bit of mathematical logic. All logic that we need in this paper is summarized in the appendix.

A *state of knowledge* is a multi-sort first-order structure. It includes five relations

$$\texttt{knows},\ \texttt{knows}_0,\ \texttt{saysto},\ \texttt{saysto}_0,\ \texttt{ensues}$$

that are given to us implicitly, by means of a logic program composed of *house rules* and *assertions*. The assertions form an *authorization policy*. The logic program operates on the rest of the state of knowledge that we call the *substrate*. Formally, the substrate is the reduct of the state of knowledge obtained by forgetting the five relations. The five relations are *superstrate* relations.

We presume that the substrate is given to us in the sense that there is a feasible (and certainly polynomial time) algorithm Eval for evaluating basic functions and relations of the substrate. (We presume furthermore that substrate elements are given as strings. Eval takes inputs of the form $(F, b_1, \ldots, b_j)$ where F is the name of a basic function or relation of the substrate of arity j and where each a_i is a substrate element. The arity j can be zero.) In the rest of this section, we describe the substrate and then comment on the superstrate relations.

3.1. **Substrate.** The vocabulary of the substrate contains only a finite number of sort symbols, relation symbols, and function symbols of positive arity. However, it includes the vocabulary of any authorization policy over the substrate and thus contains an infinite number of constants. For future reference, we define substrate constraints. A *substrate constraint* is a quantifier-free formula in the substrate vocabulary.

The substrate may depend on application. Different applications may differ in what basic functions and relations they need. For example, some applications may require a Time sort. But there is an obligatory part of substrate, and we describe it here.

3.1.1. **Regular and synthetic layers of the substrate.** The substrate elements split into two layers: *regular* and *synthetic*. Every substrate sort is a part of one of the layers; accordingly we have *regular sorts* and *synthetic sorts*. There are exactly three synthetic sorts: Attribute, Info, and Speech. The regular sorts include at least the sort Principal.

Element of regular sorts *regular elements*, or elements of *Regular* type, and elements of synthetic sorts are *synthetic elements*. Every substrate function is either a *synthetic function*, meaning that it takes synthetic values, or a *regular function*, meaning that it takes regular

values. All basic substrate relations are *regular relations*. Variables of regular sorts are *regular variables*, and variables of synthetic sorts are *synthetic variables*. Compound expressions $f(t_1, \ldots, t_j)$, where j may be zero, are *regular expressions* if f is regular, and are *synthetic expressions* if f is synthetic.

Proviso 1. Every synthetic function is a free constructor and every synthetic element is constructed, in a unique way, from regular elements by means of synthetic functions. □

The proviso allows us to assign to each substrate element b a unique ordered finite tree, the *semantic tree* of b, satisfying the following conditions:

- If b is regular than semtree(b) consists of a single node that is b itself.
- Suppose that b is synthetic. Then $b = F(b_1, \ldots, b_n)$ for a some synthetic function F and some elements $b_1, \ldots, b_n$ where F and the tuple $(b_1, \ldots, b_n)$ are unique. The element b is the root of semtree(b). Under the root there are n subtrees in this order: semtree(b_1), ..., semtree(b_n).

3.1.2. **Regular components.** The substrate contains a binary relation a `regcomp` b that holds if and only if element a is regular, element b is synthetic, and a is a leaf of semtree(b). For example, consider an infon f = (`manager(Bob) can sing`). The manager of Bob is a regular component of f (but Bob is not). The relationship a `regcomp` b is semantic and holds or fails independently of the syntactic presentation of a and b. If the manager of Bob happens to be the husband of Alice then the husband of Alice is a regular component of f.

We will need a syntactic counterpart of the regular component relation. For each expression t define a *syntactic tree* of t as follows:

- If expression t is regular, then syntree(t) consists of a single node that is t itself.
- For a synthetic expression $t = F(t_1, \ldots, t_n)$, the expression t is the root of syntree(t). Under the root there are n subtrees in this order: syntree(t_1), ..., syntree(t_n).

A regular subexpression s of t is a *regular component* of t if s is a leaf of syntree(t). For example, consider an infon expression t = (`manager(`v`) can sing`). Then the subexpression `manager(`v`)` is a regular component of t (but v is not).

3.1.3. **House constructors.** Our framework requires the presence of the following synthetic functions.

- Functions `said` and $\texttt{said}_0$ of type Info → Speech.
- Functions `tdOn` and $\texttt{tdOn}_0$ of type Info → Attribute.
- A function $\mathcal{I}$ of type (Principal × Speech) ∪ (Regular × Attribute) → Info.
- A function + of type Info × Info → Info.
- Functions `canActAs` and `canSpeakAs` of type Principal → Attribute.
- A constant `exists` of type Attribute.

Convention 3.1. Function symbols `said` and `tdOn` can be written as $\texttt{said}_\infty$ and $\texttt{tdOn}_\infty$ respectively. Thus $\texttt{said}_d$ denotes `said` when $d = \infty$ and denotes $\texttt{said}_0$ when $d = 0$, and similarly for $\texttt{tdOn}_d$. In the case of functions $\texttt{said}_d$, $\texttt{tdOn}_d$, `canActAs` and `canSpeakAs`, we write the function name of the house constructor followed by the argument, with no parentheses. For example, `canActAs Bob` is the attribute obtained by applying the function `canActAs` to the constant `Bob`. In the case of the function $\mathcal{I}$ we generally omit the function name altogether writing just `Bob is a user` rather than $\mathcal{I}$(`Bob, is a user`).

Proviso 2. Every additional (to the house constructors) synthetic function or synthetic constant takes Attribute or Info values. In other words, the only synthetic functions which produce speeches are `said` and $\texttt{said}_0$.

3.1.4. **Discussion on house constructors.** We give a few remarks on the intended meaning of the house constructors. Formally, the meaning will be given by *house rules* below.

Spoken and attribute infons. Function `infon` has two subfunctions, one of the type Principal × Speech → Info, and the other of the type Regular × Attribute → Info. The first generates *spoken infons*, and the second *attribute infons*.

Internal knowledge and undelegatable authorization. The difference between p `said foo` and p $\texttt{said}_0$ `foo` is that the latter reflects the internal (initial, prior) knowledge of the principal p. The difference between p `tdOn foo` and p $\texttt{tdOn}_0$ `foo` is that the first allows p to delegate the trust and the second does not.

Can internal knowledge be acquired? One may argue that an infon of the form p $\texttt{said}_0$ q `said foo` makes no sense as it reflects learned rather than internal knowledge of the principal p. But we do not exclude such infons. The q-to-p communication might have happened outside the official authorization policy so that the resulting knowledge of p is internal as far as the system is concerned. See Example 5.5 in this connection.

Sum of infons. It may seem odd that + is a free constructor. Why distinguish between $f + g$ and $g + f$ for example? The reason is to simplify the exposition. The operation + will be commutative, associative and idempotent but only modulo an appropriate equivalence relation. Formally infons $f + g$ and $g + f$ are different.

Example 3.1. Here are examples of attributes, speeches and infons.

- `canActAs Director` is an attribute where `Director` is a principal. Syntactically it is an attribute expression where `Director` is a principal constant.
- `Alice canActAs Director` is an infon obtained by applying the function $\mathcal{I}$ to the principal `Alice` and the attribute `canActAs Director`. Syntactically it is an attribute infon expression with principal constants `Alice` and `Director`.
- `canRead` *file* is an attribute expression where `canRead` is a function (more pedantically a function name) of type File → Attribute and *file* is a variable of sort File. Here File is a regular sort.
- `is a trusted merchant` is an attribute. Syntactically it is an attribute constant.
- p `is a trusted merchant` is an infon expression obtained by applying the function $\mathcal{I}$ to a principal variable p and the attribute constant `is a trusted merchant`.
- $\texttt{said}_0$ p `is a trusted merchant` is a speech expression obtained by applying the function $\texttt{said}_0$ to the infon expression p `is a trusted merchant`.
- `Alice` $\texttt{said}_0$ `Bob is a trusted merchant` is a spoken infon obtained by applying the function $\mathcal{I}$ to the principal `Alice` and the speech $\texttt{said}_0$ `Bob is a trusted merchant`. Syntactically it is a spoken-infon expression.
- `STS said Alice` $\texttt{said}_0$ `Bob is a trusted merchant` is a spoken infon obtained by applying the function $\mathcal{I}$ to the principal `STS` and the speech `said Alice` $\texttt{said}_0$ `Bob is a trusted merchant`. Syntactically it is a spoken-infon expression.

3.2. **Superstrate relations.** In our approach, infons are state elements rather than propositions that can be true or false. One of the superstrate relations is `knows` of type Principal × Info. Intuitively it consists of pairs (p, x) so that infon x is known to principal p. We write p `knows` x rather than `knows`(p, x).

Principals may communicate parts of their knowledge to other principals. (We will shortly introduce format for rules which the principals may write to manage their knowledge and their communications.) For this we have another superstrate relation `saysto` of type Principal × Info × Principal. Formally `saysto` consists of triples (p, x, q) such that principal p says infon x to principal q. We write p `says` x `to` q rather than `saysto`(p, x, q).

We wish to keep track of whether a principal's knowledge of an infon does or does not rely on communications from other principals. For this we have a third superstrate relation $\texttt{knows}_0$ of type Principal × Info. Formally it consists of pairs (p, x) such that infon x is known to principal p internally, independently of assertions made by other principals. We write p $\texttt{knows}_0$ x rather than $\texttt{knows}_0(p, x)$.

A principal may inform other principals not only that he knows an infon, but that he knows it on the basis of his internal knowledge. For this purpose we have a fourth superstrate relation $\texttt{saysto}_0$ of type Principal × Info × Principal. Formally it consists of triples (p, x, q) such that principal p says to principal q that he knows infon x internally. We will put restrictions below that prevent p from basing such assertions on anything other than (the given substrate and) his internal knowledge. We write p $\texttt{says}_0$ x `to` q rather than $\texttt{saysto}_0(p, x, q)$.

Formulas

$$p \texttt{ knows } x,\ p \texttt{ says } x \texttt{ to } q$$

can be written in the form

$$p \texttt{ knows}_\infty x,\ p \texttt{ says}_\infty x \texttt{ to } q,$$

respectively. Thus p $\texttt{knows}_d$ x denotes p `knows` x when $d = \infty$ and denotes p $\texttt{knows}_0$ x when $d = 0$, and similarly for p $\texttt{says}_d$ x `to` q.

Our final superstrate relation `ensues` is of type Info × Info. Intuitively it consists of pairs (f, g) such that f is less informative than g or precisely as informative as g, and we will say what this means later on. We write $x \leq y$ instead of `ensues`$(\texttt{x}, \texttt{y})$.

The superstrate relations satisfy various policy rules of the form:

$$P(t_1, \ldots, t_r) \leftarrow \varphi$$

where P is one of the superstrate relations and φ is an existential first-order formula in the expanded vocabulary where negations are applied only to atomic formulas involving relations of the substrate. Think of such a rule as a constraint on the superstrate relations over the substrate. For any legitimate values of the free variables of the rule, if the *body* φ of the rule is true then the *head* $P(t_1, \ldots, t_r)$ should be true as well.

For example, according to rule

$$q \texttt{ knows } p \texttt{ said } \textit{infon} \leftarrow p \texttt{ says } \textit{infon} \texttt{ to } q$$

the following holds for any p, q and any *infon*: if the triple $(p, \textit{infon}, q)$ satisfies relation `saysto` then the pair $(p,\ q$ `said` *infon*$)$ satisfies `knows`.

The rules for our superstrate relations are taken up in the next section. It turns out that, for every superstrate relation P, there is a unique set of tuples of elements of the substrate

such that the constraints imposed by the rules force these tuples to belong to P. That set is the intended value of P. The intended values can be computed. We explain the details in the appendix.

Remark 3.2. The restriction that the body is a formula of the existential first-order logic can be relaxed but this issue will be taken up elsewhere.

Remark 3.3. One can develop DKAL without the relations `knows` and $\texttt{knows}_0$, adopting the convention that knowing an infon is represented by saying it to oneself. In practice this means replacing $p\ \texttt{knows}_d\ x$ by $p\ \texttt{says}_d\ x\ \texttt{to}\ p$ throughout. We choose to present DKAL *with* the relations $\texttt{knows}_d$ so as to keep the exposition closest to the conceptual understanding of the language.

4. House Rules and Authorization Policy

Recall logic programs of the logic appendix A. A logic program is a collection of logic rules. Here we are interested in logic programs of a very particular kind. The rules split into two categories: assertions and house rules. Assertions are placed by individual principals, and the set of assertions is the current *authorization policy*.

4.1. **Assertions.** An assertion placed by a principal A has one of two forms. The first form is

$$\begin{aligned} A\ \texttt{knows}_d\ x\ \leftarrow\ & A\ \texttt{knows}_d\ x_1 \wedge \cdots \wedge A\ \texttt{knows}_d\ x_n \wedge con \wedge \\ & A\ \texttt{knows}_d\ t_1\ \texttt{exists}\ \ \wedge \cdots \wedge A\ \texttt{knows}_d\ t_k\ \texttt{exists} \end{aligned} \tag{As1}$$

in short

$$A :_d\ x\ \leftarrow\ x_1, \ldots, x_n, con.$$

The second form is

$$\begin{aligned} A\ \texttt{says}_d\ x\ \texttt{to}\ p\ \leftarrow\ & A\ \texttt{knows}_d\ x_1 \wedge \cdots \wedge A\ \texttt{knows}_d\ x_n \wedge con \wedge \\ & A\ \texttt{knows}_d\ t_1\ \texttt{exists}\ \ \wedge \cdots \wedge A\ \texttt{knows}_d\ t_k\ \texttt{exists} \end{aligned} \tag{As2}$$

in short

$$A :_d\ x\ \texttt{to}\ p\ \leftarrow\ x_1, \ldots, x_n, con.$$

Here

- A, the *owner* of the assertion, is a ground principal expression, d is zero or infinity (and the infinity subscript is usually skipped);
- $x, x_1, \ldots, x_n$ are infon expressions, and *con* is a substrate constraint, that is a quantifier-free substrate formula;
- all variables are regular, and p, the *target* variable, is a variable of sort Principal;
- in (As1), the list $t_1, \ldots, t_k$ consists of (i) the variables in the assertion and (ii) the non-ground regular components of x. (The sets (i) and (ii) may intersect but the list has no repetitions.)
- in (As2), the list $t_1, \ldots, t_k$ is as above except that the target variable p may not be on the list (even though it occurs in the assertion and even if it is a regular component of x).

The part of an assertion to the left of $\leftarrow$ is the *head* of an assertion, and the part to the right of $\leftarrow$ is the *body* (or the premise). The first form (As1) is a *knowledge assertion*, and the second form (As2) is a *speech assertion*.

Caution. It is convenient to write assertion in the short form. We do that even if $n = 0$ and there is no substrate constraint (and so we omit $\leftarrow$ as well). In such a case the short form shows no body but the full form body may have some conjuncts A $\texttt{knows}_d$ t_i $\texttt{exists}$.

This completes the description of the two assertion forms. We end the subsection with several comments on these forms. Let R (an allusion to "rule") be an assertion of the form (As1) or (As2).

4.1.1. **Assertion placement.** R does not have to be literally placed by principal A. For example, if A is an employee of a large organization, the assertion may be placed by his manager or by the HR department. When we say that R is placed by A, or that R is owned by A, we mean only that R starts with A. For the purpose of exposition, it is convenient to pretend that the assertions that start with a principal A are placed by A.

4.1.2. **A-bound variables and regular components.** For any variable v of R different from p, the body of R contains the conjunct A $\texttt{knows}_d$ v $\texttt{exists}$. In that sense, the variables of R are A-bound.

Similarly, if t is a non-ground regular component of x, and t differs from p, then the body of R contains the conjunct A $\texttt{knows}$ t $\texttt{exists}$. In that sense t is A-bound.

In the (As2) case, the body of R is not required to have a conjunct A $\texttt{knows}$ p $\texttt{exists}$. A may issue a proclamation to principals whose existence is not known to A. For example, an assertion

$$A :_d \ x \ \texttt{to} \ all \ \leftarrow \ x_1, \ldots, x_n, con$$

where *all* is a "fresh" principal variable that does not appear in $x, x_1, \ldots, x_k$, *con* addresses all principals. But the set of addressees may be bound in one way or another. For example, the substrate constraint may have a conjunct $p = s$ where s is a expression.

4.1.3. **Regular elements that A knows of.** We keep the number of regular elements that a principal knows of (that is he knows that they exist) finite. That goal is behind our requirements that the variables of R and the non-ground regular components of x be A-bound. The requirements is then used to restrict the search space for true instances of the body of R: only regular elements that A knows of need to be tried as values for the variables of R.

Both requirements are necessary for decidability. Indeed, consider a substrate where the regular layer includes arithmetic: natural numbers, constant $\texttt{zero}$, addition, multiplication and the successor function S. By [21], there is no algorithm that, given an integer polynomial $G(u_1, u_2, u_3, u_4)$, determines whether G takes value 0. Let $\texttt{foo}$ be a ground infon.

If we omit the requirement that the variables be A-bound in assertions, then any polynomial $G(u_1, u_2, u_3, u_4)$ gives rise to a legal assertion

$$A \ \texttt{knows foo} \ \leftarrow \ G(u_1, \ldots, u_4) = \texttt{zero}.$$

Let $\mathcal{A}_G$ be the authorization policy that consists of this one assertion. The decision problem whether A knows $\texttt{foo}$ under $\mathcal{A}_G$ is undecidable.

If we require that the variables be A-bound but omit the requirement that the non-ground regular components of x be A-bound then assertions

$$A \texttt{ knows zero exists}$$
$$A \texttt{ knows } S(u) \texttt{ exists} \leftarrow A \texttt{ knows}_d \ u \texttt{ exists},$$
$$A \texttt{ knows foo} \leftarrow G(u_1, \ldots, u_4) = \texttt{zero} \wedge A \texttt{ knows } u_1 \texttt{ exists} \ \wedge \cdots \wedge A \texttt{ knows } u_4 \texttt{ exists}$$

are legal. Let $\mathcal{A}_G$ be the authorization policy composed of these three assertions. Under $\mathcal{A}_G$, A knows of all natural numbers, and the decision problem whether A knows `foo` is undecidable. The trouble arises because of the second assertion where the role of x is played by the infon expression ($S(u)$ `exists`); the non-ground regular component $S(u)$ of the infon expression is not A-bound. In both counter-examples `knows` could be replaced with $\texttt{knows}_0$.

4.1.4. **Impossible liberalizations.** One may be tempted to liberalize (As2) by replacing $A \texttt{ says}_d \ x \texttt{ to } p$ with $A \texttt{ says}_d \ x \texttt{ to } t(p)$ where $t(p)$ is a expression whose only variable is p (without requiring p to be A-bound). However it may be hard to decide which principles have the form $t(p)$, and the liberalization leads to undecidability.

Indeed, consider again a substrate where the regular layer includes arithmetic, and let `foo` be a ground infon. Order quadruples of natural numbers first by the maximum and then lexicographically. To simplify the exposition, we require this time that arithmetic contain unary functions $F_1(n), F_2(n), F_3(n), F_4(n)$ such that $\langle F_1(n), F_2(n), F_3(n), F_4(n)\rangle$ is the *nth* quadruple. Suppose that every natural number represents a principal. Under the liberalization in question, the assertion

$$A : \texttt{foo to } G((F_1(p), F_2(p), F_3(p), F_4(p))$$

is legal. Let $\mathcal{A}_G$ be the authorization policy that consists of this one assertion. By the first Say2know house rule (see the next subsection), principal `zero` knows that A `said foo` if and only if $G((F_1(p), F_2(p), F_3(p), F_4(p)) = 0$ for some p which happens if and only if $G(u_1, u_2, u_3, u_4)$ takes value 0. Thus the decision problem whether `zero` knows (A `said foo`) under $\mathcal{A}_G$ is undecidable.

4.2. **House rules.** House rules reflect the inherent meaning of the house constructors. We list our house rules together with short comments.

4.2.1. **K0∞ house rule:**

$$p \texttt{ knows } x \leftarrow p \texttt{ knows}_0 \ x.$$

Internal knowledge is knowledge.

4.2.2. **Say2know house double rule:**

$$p \texttt{ knows } q \texttt{ said}_d \ x \leftarrow q \texttt{ says}_d \ x \texttt{ to } p.$$

Principal p knows whatever is said to him; he also knows whether the speech was based on the internal knowledge of the speaker. The two Say2know rules, corresponding to the two values of d, form a *double rule*. Note that p learns only the spoken infon $q \texttt{ said}_d \ x$, not the infon x itself. Learning x itself depends on whether p has placed trust in q on x, see rule TrustApplication below.

4.2.3. **KSum house double rule:**

$$p \ \texttt{knows}_d \ x + y \leftarrow (p \ \texttt{knows}_d \ x) \wedge (p \ \texttt{knows}_d \ y).$$

The converse will follow form the KMon and ESum policy rules.

4.2.4. **KMon house double rule:**

$$p \ \texttt{knows}_d \ x \leftarrow x \leq y \wedge (p \ \texttt{knows}_d \ y).$$

Knowledge is monotone with respect to the ensue relation.

The remaining house rules govern the ensue relation for which we use symbol $\leq$. The intuition behind $x \leq y$ is that x is less informative than y or precisely as informative as y.

4.2.5. **EOrder house rules:**

$$x \leq x,$$
$$x \leq z \leftarrow (x \leq y) \wedge (y \leq z).$$

Thus the ensue relation is a preorder relation on infons. These rules will be used so often that, in many cases, they will be used implicitly.

Let $x \sim y$ abbreviate $(x \leq y) \wedge (y \leq x)$. If $x \sim y$, we say that infons x, y are *equivalent*.

4.2.6. **ESum house rules:**

$$x \leq \ x + y,$$
$$y \leq \ x + y,$$
$$x + y \leq \ z \ \leftarrow \ (x \leq z) \wedge (\ y \leq z).$$

Thus $x + y$ is the least upper bound for x and y in the preorder $\leq$. It follows that the sum is commutative, associative and idempotent with respect to the equivalence relation on infons.

Corollary 4.1. *For any infons x, y, z in the state of knowledge, we have*

$$x + y \sim y + x$$
$$(x + y) + z \sim x + (y + z)$$
$$x + x \sim x.$$

4.2.7. **Exists house rule:**

$$q \ \texttt{exists} \ \leq \ x \ \leftarrow \ q \ \texttt{regcomp} \ x.$$

Here $(q \ \texttt{regcomp} \ x)$ is a substrate constraint. To understand the rule, consider it from the point of view of a principal p, combining it with the house rule KMon. If p knows some infon x with regular component q, then p knows that q exists.

4.2.8. **Said0∞ house rule:**

$$p \ \texttt{said} \ x \ \leq \ p \ \texttt{said}_0 \ x.$$

Saying based on internal knowledge is saying. (See also Lemma 4.5.1 in this connection.)

4.2.9. **SaidMon house double rule:**

$$p \ \texttt{said}_d \ x \leq p \ \texttt{said}_d \ y \ \leftarrow \ x \leq y.$$

The function `said` is monotone with respect to the ensue relation.

4.2.10. **SaidSum house double rule:**

$$p\ \texttt{said}_d\ x + y\ \leq\ (p\ \texttt{said}_d\ x) + (p\ \texttt{said}_d\ y).$$

The converse follows from rules SaidMon and ESum. Thus $\texttt{said}$ and $\texttt{said}_0$ distribute over sums.

Corollary 4.2. $p\ \texttt{said}_d\ x + y\ \sim\ (p\ \texttt{said}_d\ x) + (p\ \texttt{said}_d\ y)$.

4.2.11. **SelfQuote house double rule:**

$$p\ \texttt{said}_d\ x\ \leq\ p\ \texttt{said}_d\ p\ \texttt{said}_d\ x.$$

4.2.12. **Trusted0∞ house rule:**

$$p\ \texttt{tdOn}_0\ x\ \leq\ p\ \texttt{tdOn}\ x.$$

Trust without authority to delegate follows from trust with authority to delegate.

4.2.13. **TrustApplication house double rule:**

$$x\ \leq\ (p\ \texttt{tdOn}_d\ x) + (p\ \texttt{said}_d\ x).$$

A principal p trusted on x exercises this trust by saying x. Note that the subscript of $\texttt{said}$ should match that of $\texttt{tdOn}$. For, consider the stronger rule $x \leq (B\ \texttt{tdOn}_0\ x) + (p\ \texttt{said}\ x)$ and suppose $A\ \texttt{knows}\ B\ \texttt{tdOn}_0\ x$ so that B can cause A to know x by saying x. But B can bypass the delegation restriction and delegate to C the ability to cause A to know x by placing an assertion $B : x\ \texttt{to}\ A \leftarrow B\ \texttt{knows}\ C\ \texttt{said}\ x$. If C places an assertion $C : x\ \texttt{to}\ B$, we have $B\ \texttt{says}\ x\ \texttt{to}\ A$ by B's assertion, hence we have $A\ \texttt{knows}\ B\ \texttt{said}\ x$ by Say2know, and therefore we have $A\ \texttt{knows}\ x$ by TrustApplication. A similar issue arises in SecPAL, and our subscript matching requirement is derived from the solution there.

4.2.14. **Del house double rule:**

$$p\ \texttt{tdOn}\ (q\ \texttt{tdOn}_d\ x) \leq\ (p\ \texttt{tdOn}\ x) + (q\ \texttt{exists}).$$

This is the delegation double rule. Trust with no subscript (or subscript ∞) can be delegated to others. The rule is used in conjunction with KMon: if $A\ \texttt{knows}\ p\ \texttt{tdOn}\ x$, and $p\ \texttt{says}\ (q\ \texttt{tdOn}_d\ x)\ \texttt{to}\ A$, then it follows that $A\ \texttt{knows}\ q\ \texttt{tdOn}_d\ x$; the trust that A placed in p has been delegated to q.

4.2.15. **Del$^-$ house double rule:**

$$p\ \texttt{tdOn}_d\ x\ \leq\ p\ \texttt{tdOn}_d\ p\ \texttt{tdOn}_d\ x.$$

4.2.16. **Role house rules:**

$$p\ \mathit{attribute}\ \leq\ (q\ \mathit{attribute}) + (p\ \texttt{canActAs}\ q),$$
$$q\ \mathit{speech}\ \leq\ (p\ \mathit{speech}) + (p\ \texttt{canSpeakAs}\ q).$$

Here *attribute* and *speech* are variables of types Attribute and Speech respectively. The functions $\texttt{canActAs}$ and $\texttt{canSpeakAs}$ allow assigning roles. Notice that these house rules transfer attributes and speeches in opposite directions. For example, if Bob can act as director and if directors can hire then Bob can hire. If Bob can speak as director and Bob says "Cathy is hired" then the director says "Cathy is hired."

4.3. **Some consequences and discussion.** Let X be a substrate and $\mathcal{A}$ an authorization policy. Further, let Π be the program that consists of the house rules and the assertions in $\mathcal{A}$. The state $\Pi(X)$ is our state of knowledge.

4.3.1. **Subrecursions.**

Proposition 4.3.
- *The interpretation of* ensues *in the state of knowledge depends only on the substrate and does not depend on the authorization policy.*
- *For any principal p, the set of infons f such that*

$$p \ \texttt{knows}_0 \ f \text{ holds in the state of knowledge}$$

and the set of elements b such that

$$p \ \texttt{knows}_0 \ b \ \texttt{exists} \quad \text{holds in the state of knowledge}$$

depend only on the substrate and the assertions of the form (As1) placed by p himself.
- *For any principal p, the set of pairs (q, f) such that $p \ \texttt{says}_0 \ f \ \texttt{to} \ q$ holds in the state of knowledge depends only on the substrate and the assertions placed by p.*

Proof. Just examine the rules and assertions that are used to compute ensues, $\texttt{knows}_0$ and $\texttt{saysto}_0$. □

4.3.2. **Some simple consequences of house rules.**

Lemma 4.4. *The following formulas are universally true in the state of knowledge. The subscript d could be 0 or ∞.*

(1) $(p \ \texttt{knows} \ x) \wedge (q \ \texttt{regcomp} \ x) \to p \ \texttt{knows} \ q \ \texttt{exists}$.
(2) $(p \ \texttt{knows} \ q \ \texttt{tdOn}_d \ x) \wedge (p \ \texttt{knows} \ q \ \texttt{said}_d \ x) \to p \ \texttt{knows} \ x$.
(3) $q_2 \ \texttt{tdOn}_d \ x \ \leq \ (q_1 \ \texttt{tdOn} \ x) + (q_1 \ \texttt{said} \ q_2 \ \texttt{tdOn}_d \ x)$,
$(p \ \texttt{knows} \ q_1 \ \texttt{tdOn} \ x) \wedge (p \ \texttt{knows} \ q_1 \ \texttt{said} \ q_2 \ \texttt{tdOn}_d \ x) \to p \ \texttt{knows} \ q_2 \ \texttt{tdOn}_d \ x$.
(4) $p \ \texttt{canActAs} \ r \ \leq \ (p \ \texttt{canActAs} \ q) + (q \ \texttt{canActAs} \ r)$.

Proof. 1. By the Exists house rule, we have $q \ \texttt{exists} \leq x$. Now apply the appropriate KMon house rule.

2. Suppose that p knows infons $q \ \texttt{tdOn}_d \ x$ and $q \ \texttt{said}_d \ x$. By the appropriate KSum house rule, p knows the infon $f = (q \ \texttt{tdOn}_d \ x) + (q \ \texttt{said}_d \ x)$. By the TrustApplication house rule, $x \leq f$. By the KMon house rule, $p \ \texttt{knows} \ x$.

3. It suffices to prove only the first formula because the second formula follows from the first by the appropriate KMon house rule. Let $f = (q_1 \ \texttt{tdOn} \ x) + (q_1 \ \texttt{said} \ q_2 \ \texttt{tdOn}_d \ x)$. By 1 and the appropriate Del house rule, we have $q_1 \ \texttt{tdOn} \ q_2 \ \texttt{tdOn}_d \ x \ \leq \ f$. Now apply the appropriate TrustApplication house rule.

4. Apply the first Role house rule with *attribute* $= \texttt{canActAs} \ r$. □

4.3.3. **Redundant rules.** Consider extending the set of house rules with an additional rule R. The rule R is *redundant* if the addition of R does not change the interpretation of knows in any state. The reason for that definition is that our queries are all about knowledge.

Proposition 4.5. *The following rules, where $d \in \{0, \infty\}$, are redundant.*

(1) $p \ \texttt{says} \ x \ \texttt{to} \ q \leftarrow p \ \texttt{says}_0 \ x \ \texttt{to} \ q$.
(2) $p \ \texttt{says}_d \ x + y \ \texttt{to} \ q \leftarrow (p \ \texttt{says}_d \ x \ \texttt{to} \ q) \wedge (p \ \texttt{says}_d \ y \ \texttt{to} \ q)$.
(3) $p \ \texttt{says}_d \ x \ \texttt{to} \ q \leftarrow x \leq y \wedge (p \ \texttt{says}_d \ y \ \texttt{to} \ q)$.

Proof. We start with an obvious claim: For each d, there are no assertions with $\texttt{saysto}_d$ in the premise, and there is only one house rule where relation $\texttt{saysto}_d$ occurs on the right, namely the appropriate instance of the Say2know double rule.

1. By the claim above, with $d = \infty$, there is only one way that p `says` z `to` q can be used: to derive an infon q `knows` p `said` z. So it suffices to prove an implication

$$p \texttt{ says}_0 \ z \texttt{ to } q \to q \texttt{ knows } p \texttt{ said } z.$$

Suppose $p \texttt{ says}_0 \ x \texttt{ to } q$. By Say2know, $q \texttt{ knows } p \texttt{ said}_0 \ x$. By KMon and Said0$\infty$, $q \texttt{ knows } p \texttt{ said } x$.

2. By the claim above, it suffices to prove an implication

$$(p \texttt{ says}_d \ x \texttt{ to } q) \wedge (p \texttt{ says}_d \ y \texttt{ to } q) \to q \texttt{ knows } p \texttt{ said}_d \ x + y.$$

Suppose $(p \texttt{ says}_d \ x \texttt{ to } q)$ and $(p \texttt{ says}_d \ y \texttt{ to } q)$. By Say2know, $q \texttt{ knows } p \texttt{ said}_d \ x$ and $q \texttt{ knows } p \texttt{ said}_d \ y$. By KSum, $q \texttt{ knows } \big((p \texttt{ said}_d \ x) + (p \texttt{ said}_d \ y)\big)$. By KMon and SaidSum, $q \texttt{ knows } p \texttt{ said}_d \ x + y$.

3. By the claim above, it suffices to prove an implication

$$x \leq y \wedge (p \texttt{ says}_d \ y \texttt{ to } q) \to q \texttt{ knows } p \texttt{ said}_d \ x.$$

Suppose $x \leq y \wedge (p \texttt{ says}_d \ y \texttt{ to } q)$. By Says2know, $q \texttt{ knows } p \texttt{ said}_d \ y$. By KMon and SaidMon, $q \texttt{ knows } p \texttt{ said}_d \ x$. □

4.3.4. **Discussion about extending the set of house rules.** One may want to require that there is a least informative infon:

`Vacuous` $\leq$ `x.`

Any principal that knows anything would know the Vacuous infon.

By Lemma 4.4.4, the `canActAs` relation happens to be transitive. We also have

$$r \texttt{ says foo} \leq (p \texttt{ says foo}) + (p \texttt{ canSpeakAs } q) + (q \texttt{ canSpeakAs } r)$$

but the transitivity of `canSpeakAs` itself is not derivable from our house rules; also the reflexivity of either relation (`canActAs` or `canSpeakAs`) is not derivable. One may want to impose the transitivity of `canSpeakAs` and the reflexivity of `canActAs` and `canSpeakAs`. The utility of that is debatable.

One may reasonably argue in favor of adding a Cartesian rule p `knows` p `exists`. But the effect of the Cartesian rule is achieved by a single assertion

A: `(`p `exists) to` p

where A could be e.g. the system itself. By Say2know, it follows that p `knows` p `exists` for every principal p.

We considered rules

(1) $p \texttt{ tdOn}_d \ x \leq p \texttt{ tdOn}_d \ x + y$,
(2) $p \texttt{ tdOn}_d \ x + y \leq (p \texttt{ tdOn}_d \ x) + (p \texttt{ tdOn}_d \ y)$

as candidates for house rules but did not endorse them. Rule 1 asserts that $\texttt{tdOn}_d$ is monotone. But this is not necessarily so. For example a low-level administrator may be trusted on a package of rights to new hires but may not be allowed to pick and choose which rights to give. Rule 2 looks more plausible but we have a counterexample for it as well; see Example 5.6.

4.4. **Queries and computability.** Fix a substrate X, and let Υ be the vocabulary of X extended with the superstrate relation names. $\mathcal{A}$ will denote an authorization policy over X. For each $\mathcal{A}$, let $\Pi_{\mathcal{A}}$ be the program that consists of the house rules and the assertions in $\mathcal{A}$.

First we define basic queries in the vocabulary Υ. Then we formulate two theorems; one asserts the existence of an algorithm answering basic queries, and the other bounds the time complexity of the answering algorithm; the answering algorithm itself will be presented in Section 6 where the two theorems are also proved. Then we generalize the notion of basic queries and prove that the two theorems remain valid.

4.4.1. **Basic queries.** A *basic query* is a formula $p\ \texttt{knows}_d\ t(v_1, \ldots, v_k)$ where:

- p is a ground principal expression in the substrate vocabulary,
- $v_1, \ldots, v_k$ are regular variables,
- t is an infon expression in the vocabulary Υ, with all its variables among $v_1, \ldots, v_k$, and
- d is 0 or ∞.

Basic queries are evaluated over the state of knowledge $\Pi_{\mathcal{A}}(X)$ given by the fixed substrate X with respect to an authorization policy $\mathcal{A}$. The *answer* to a basic query $Q = (p\ \texttt{knows}_d\ t(v_1, \ldots, v_k))$ under authorization policy $\mathcal{A}$ is denoted $\mathrm{ans}_{\mathcal{A}}(Q)$. It is the set of tuples $\langle b_1, \ldots, b_k\rangle$ of substrate elements such that the type of b_i is that of v_i and

$$\Pi_{\mathcal{A}}(X) \models \big(p\ \texttt{knows}_d\ t(b_1, \ldots, b_k) \wedge p\ \texttt{knows}_d\ b_1\texttt{exists} \wedge \cdots \wedge p\ \texttt{knows}_d\ b_k\ \texttt{exists}\big).$$

The precise meaning of the displayed statement is this: the conjunction of the $k+1$ atomic formulas holds in the state of knowledge $\Pi_{\mathcal{A}}(X)$ under the assignment of elements $b_1, \ldots, b_k$ to the variables $v_1, \ldots, v_k$ respectively.

If v_i is a regular component of $t(v_1, \ldots, v_k)$ then the requirement $p\ \texttt{knows}_d\ b_i\ \texttt{exists}$ is superfluous as it follows from $p\ \texttt{knows}_d\ t(b_1, \ldots, b_k)$ by the Exist house rule. But in general the requirement is necessary. For example suppose that the substrate has a regular function `master` that one way or another assigns principals to files and consider a expression $t(\mathit{file}) =$ (`master`(file) `tdOn foo`) where *file* is a regular variable and `foo` is a ground infon expression. expression `master`(file) is a regular component of $t(\mathit{file})$ but the variable *file* is not. And the answer to a query p `knows` $t(\mathit{file})$ may be infeasibly large, even infinite, if we require that it contains all files whose master is trusted on `foo`, including those files whose existence is not known to p.

If $k = 0$, so that Q is ground, the answer set is either empty or contains one element, namely the empty tuple. This is unnatural. The intended meaning of the answer is `false` in the first case and `true` in the second. It would be reasonable but tedious to consider ground queries separately. Instead we stipulate that, in the answers to ground queries, the empty set represents `false` and the set that consists of the empty tuple represents `true`.

4.4.2. **Computing basic queries.** In Section 6, we present an algorithm for answering basic queries. For the purpose of measuring time complexity we assume that there is an algorithm Eval that evaluates the basic substrate functions and relations of X in constant time; the assumption is discussed in §A.2. We also assume that our constants and variables are strings in a fixed finite alphabet, so that all expressions and formulas in the language of our model are strings in the fixed finite alphabet. This allows us to speak about the *length*

of a expression or formula. The *length* of an authorization policy is the sum of the lengths of its assertions.

Theorem 4.6. *The answer to any basic query is finite, and there is an algorithm that, given a basic query Q and an authorization policy $\mathcal{A}$, computes* $\mathrm{ans}_{\mathcal{A}}(Q)$.

A particular algorithm for answering basic queries is constructed in Section 6.

Theorem 4.7. *The time that our algorithm needs to answer a basic query Q under an authorization policy $\mathcal{A}$ is bounded by a polynomial in* $\big(\mathrm{length}(\mathcal{A}) + \mathit{length}(Q)\big)^{\delta+1+w}$, *where δ bounds the depth to which* `said` *and* $\texttt{said}_d$ *(possibly mixed) are nested in the policy and query expression, and w bounds the number of free variables allowed in assertions and in the query expression. In particular, assuming a fixed bound on the nesting depth of* $\texttt{said}_d$ *and on the assertion and query width, the computation time of the answering algorithm is polynomial in* $\mathrm{length}(\mathcal{A}) + \mathit{length}(Q)$.

The theorems are proved in Section 6.

4.4.3. **Toward more general queries.** There is an easy way to generalize basic queries in a semantically sound way: view any first-order formula $\varphi(v_1, \ldots, v_k)$ of vocabulary Υ with free variables $v_1, \ldots, v_k$ of regular types as a potential query with the answer

$$\{\langle b_1, \ldots, b_k\rangle : \ \mathrm{type}(b_i) = \mathrm{type}(v_i) \text{ for all } i, \text{ and } \Pi(X) \models \varphi(b_1, \ldots, b_k)\}.$$

But this approach is flawed as it leads to infeasible queries.

Consider for example a query v `knows` t where v is a variable and t has no variables and thus evaluates to an infon in X. The answer to the query would be the list of all principals that know the infon. This set is unfeasibly large and possibly infinite. This explains why p is forbidden to have variables in the definition of basic queries.

For another example, consider the negation $\neg Q$ of a basic query $Q = (p$ `knows` $t(v))$. The answer to $\neg Q$ would contain all elements of X of the type of v except for the finite set $\mathrm{ans}(Q)$. Again the answer may be infinite. Notice that both Q and its negation are about the knowledge (or the lack thereof) of p. In that sense they are p-centered. As far as p-centered queries are concerned, it is natural to restrict attention to elements known to p.

4.4.4. **Single-principal-centric queries.** Let's fix an arbitrary principal expression p without variables and restrict attention to p-centered queries, that is queries about the knowledge — or the lack thereof — of p. A basic query q $\texttt{knows}_d$ $t(v_1, \ldots, v_k)$ is *p-centric* if the expression q is literally the expression p. A more general definition of p-centric queries is needed.

Remark 4.8. The requirement that expression q is literally expression p can be weakened to require only that q evaluates to the same value as p in X. The current definition has the advantage of being independent of the choice of substrate. □

For every regular type T, let

$$T_{\mathcal{A}}(p) = \mathrm{ans}_{\mathcal{A}}\big(p \ \texttt{knows} \ (v \ \texttt{exists})\big) \text{ where } v \text{ is a variable of type } T.$$

Any first-order formula $\varphi(v_1, \ldots, v_k)$ of vocabulary Υ with free variables $v_1, \ldots, v_k$ of regular types can be treated as a p-centric query, with the p-bounded answer

$$\{\langle b_1, \ldots, b_k\rangle : \ \text{every } b_i \in T^i_{\mathcal{A}}(p), \text{ and } \Pi(X) \models \varphi(b_1, \ldots, b_k)\}.$$

But this more refined approach is still flawed. There are two problems with it. One is that the formula φ may have superstrate atomic subformulas, like q `knows` t where q is different from p in the substrate X; in such cases φ is only artificially p-centric. The other problem is that the quantified variables of the formula φ may range over their whole sorts; as a result it may be infeasible to evaluate statements $\varphi(b_1, \ldots, b_k)$ even though every $b_i \in T^i_{\mathcal{A}}(p)$. This little analysis brings us to a definition of p-centric queries.

A *p-centric query* is a first-order formula φ in the vocabulary Υ such that every atomic superstrate subformula of φ is of the form p `knows`$_d$ t, every variable is of regular type, and every quantification is of the form $\exists v \in T_{\mathcal{A}}(p)$ or $\forall v \in T_{\mathcal{A}}(p)$ where T is the type of v. Alternatively, p-centric queries can be defined inductively.

- every substrate constraint is a p-centric query, and every p-centric basic query is a p-centric query,
- if Q_1, Q_2 are p-centric queries then $\neg Q_1$, $Q_1 \wedge Q_2$ and $Q_1 \vee Q_2$ are p-centric queries,
- if $Q(v)$ is a p-centric query with a free variable v of type T (and possibly other free variables) then $\big(\exists v \in T_{\mathcal{A}}(p)\big)Q(v)$ and $\big(\forall v \in T_{\mathcal{A}}(p)\big)Q(v)$ are p-centric queries.

The *answer* to a p-centric query $\varphi(v_1, \ldots, v_k)$ under an authorization policy $\mathcal{A}$ is the p-bounded answer mentioned above:

$$\mathrm{ans}_{\mathcal{A}}(\varphi(v_1, \ldots, v_k)) = \{\langle b_1, \ldots, b_k\rangle : \text{ every } b_i \in T^i_{\mathcal{A}}(p), \text{ and } \Pi(X) \models \varphi(b_1, \ldots, b_k)\}.$$

In particular, a Boolean combination of substrate constraints and p-centric basic queries is a p-centric query.

Lemma 4.9. *For any p-centric queries Q and R, we have*

$$\mathrm{ans}(Q \vee R) = \mathrm{ans}\big(\neg\big((\neg Q) \wedge (\neg R)\big)\big)$$
$$\mathrm{ans}\big(\forall v \in T_{\mathcal{A}}(p)\big)Q(v)) = \mathrm{ans}\big(\neg(\big(\exists v \in T_{\mathcal{A}}(p)\big)\neg Q(v)).$$

The proof is obvious.

4.4.5. **Answering single-principal-centered queries.** We start by defining an (answer) *envelope* of a p-centric query under an authorization policy $\mathcal{A}$. If $Q(v_1, \ldots, v_k)$ is a p-centric query where v_i is a variable of regular type T^i then

$$\mathrm{env}(Q(v_1, \ldots, v_k)) = T^1_{\mathcal{A}}(p) \times \cdots \times T^k_{\mathcal{A}}(p).$$

We have

$$\mathrm{ans}_{\mathcal{A}}(Q(v_1, \ldots, v_k)) \subseteq \mathrm{env}(Q(v_1, \ldots, v_k)),$$
$$\mathrm{ans}_{\mathcal{A}}(\neg Q(v_1, \ldots, v_k)) = \mathrm{env}(Q(v_1, \ldots, v_k)) - \mathrm{ans}_{\mathcal{A}}(Q(v_1, \ldots, v_k)).$$

Lemma 4.10. *There is a polynomial time algorithm that,*

- *given a p-centric query Q and*
- *given the sets $\mathrm{ans}_{\mathcal{A}}(\varphi)$ and $\mathrm{env}_{\mathcal{A}}(\varphi)$ for every atomic subformula of Q,*

computes $\mathrm{ans}_{\mathcal{A}}(Q)$ and $\mathrm{env}_{\mathcal{A}}(Q)$.

Proof. We explain how to compute $\mathrm{ans}_{\mathcal{A}}(Q)$ by induction on Q. It will be obvious that the algorithm is polynomial time. Note that, every subquery R of Q is p-centric, and we are given the sets $\mathrm{ans}_{\mathcal{A}}(\varphi)$ and $\mathrm{env}_{\mathcal{A}}(\varphi)$ for every atomic subformula of R. Due to the previous lemma, we may assume for brevity that Q does not use disjunction and does not use the universal quantifier. And, also for brevity, we omit the subscript $\mathcal{A}$ in the rest of the proof.

The case when Q is atomic is trivial. The case when $Q = \neg R$ is obvious: $\mathrm{ans}(Q) = \mathrm{env}(R) - \mathrm{ans}(R)$, and $\mathrm{env}(Q) = \mathrm{env}(R)$.

Suppose that Q is a conjunction

$$R_1(u_1, \ldots, u_j, v_1, \ldots, v_k) \wedge R_2(v_1, \ldots, v_k, w_1, \ldots, w_\ell)$$

where all $j + k + \ell$ variables are distinct. From the relational-database point of view, $\mathrm{ans}_{\mathcal{A}}(Q_1(u_1, \ldots, u_j, v_1, \ldots, v_k))$ is a table with $j + k$ columns. Similarly, $\mathrm{ans}_{\mathcal{A}}(Q_2(v_1, \ldots, v_k, w_1, \ldots, w_\ell))$ is a table with $k+\ell$ columns. The join of the two tables over the k columns corresponding to the common variables $v_1, \ldots, v_k$ gives $\mathrm{ans}_{\mathcal{A}}(Q)$. Similarly $\mathrm{env}(Q)$ is the join of $\mathrm{env}(R_1)$ and $\mathrm{env}(R_2)$.

Finally suppose that $Q(v_1, \ldots, v_k) = \big(\exists v_0 \in T(p)\big)R(v_0, \ldots, v_k)$ where T is the type of v_0. In this case, $\mathrm{ans}(Q)$ is the projection

$$\{(b_1, \ldots, b_k) : \ (b_0, b_1, \ldots, b_k) \in \mathrm{ans}(R) \text{ for some } b_0.\}$$

Similarly $\mathrm{env}(Q)$ is a projection of $\mathrm{env}(R)$. □

Theorem 4.11. *The answer to any single-principal-centric query is finite, and there is an algorithm that, given a single-principal-centric query Q and an authorization policy Q, computes* $\mathrm{ans}(Q)$.

Proof. The desired algorithm is the algorithm of Lemma 4.10 that uses the substrate evaluation algorithm Eval and the basic query evaluation algorithm of Theorem 4.6 to compute the givens of Lemma 4.10. Let Q be a p-centric query, and $\varphi(v_1, \ldots, v_k)$ be an atomic subformula of Q. Since

$$\mathrm{env}_{\mathcal{A}}(\varphi(v_1, \ldots, v_k)) = \mathrm{ans}_{\mathcal{A}}(p \ \texttt{knows} \ (v_1\texttt{exists} + \cdots \ v_k \ \texttt{exists})),$$

a single call to the basic query evaluation algorithm results in $\mathrm{env}(\varphi(v_1, \ldots, v_k))$. If $\varphi(v_1, \ldots, v_k)$ is a basic query then another call to the basic query evaluation algorithm results in $\mathrm{ans}(\varphi(v_1, \ldots, v_k))$. If $\varphi(v_1, \ldots, v_k)$ is a substrate constraint, we need a number of calls to Eval. Since

$$\mathrm{ans}(\varphi(v_1, \ldots, v_k)) = \{(b_1, \ldots, b_k) \in \mathrm{env}(\varphi(v_1, \ldots, v_k)) : \ X \models \varphi(b_1, \ldots, b_k)\},$$

the number of calls is the cardinality of $\mathrm{env}(\varphi(v_1, \ldots, v_k))$. □

Theorem 4.12. *The time bound of Theorem 4.7 remains valid for our algorithm for answering single-principal-centric queries. In other words, Theorem 4.7 remains valid if "basic query" is replaced with "single-principal-centric query."*

Proof. Let Q be a single-principal-centric query. The number of atomic subformulas of Q is bounded by $\mathrm{length}(Q)$. According to the previous proof, the number of calls to the basic query evaluation algorithm is $O(\mathrm{length}(Q))$, and the number of calls to Eval is bounded by the cardinality of the cumulative output of the calls to the basic query evaluation algorithm. It remains to recall that every call to Eval costs us only 1 time unit. □

5. Examples

We give a few examples, which serve to illustrate DKAL and to explain some choices we made in designing DKAL. We begin with examples which illustrate delegation and trust in scenarios which are not user centric. We say that a principal p knows of q if p knows the infon q `exists`.

Example 5.1 (Delegation)**.** We make some assumptions about the substrate. There is a sort File and a function `owner: File` $\to$ `Principal` that assigns to each file the owner of the file. There is an attribute function `canRead` with one argument of type File, and the rights to read files are controlled by a read-rights manager RR: a principal p is allowed to read file f just in case that RR knows the infon p `canRead` f.

The policy is that each principal is allowed to read his files (that is the files that he owns), is allowed to let others read his files, and is allowed to let them delegate the right:

1. `RR:` p `canRead` *file* $\leftarrow$ `owner(`*file*`) =` p`,`
2. `RR:` p `tdOn` r `canRead` *file* $\leftarrow$ `owner(`*file*`) =` p`.`

A policy of this kind can apply in a community web service consisting of a great many users, who post files and share them with other users in the system. We assume here that the system has a central read-rights manager. Alternatively one could use a user centric approach, and localize the computations so that each user computes his own rights.

Recall that we write assertions in an abbreviated form §4.1. Since this is our first example, let us rewrite the two assertions in full.

1*. `RR knows` p `canRead` *file* $\leftarrow$ `owner(`*file*`) =` p $\wedge$
`RR knows` p `exists` $\wedge$ `RR knows` *file* `exists.`

2*. `RR knows` p `tdOn` r `canRead` *file* $\leftarrow$ `owner(`*file*`) =` p $\wedge$
`RR knows` p `exists` $\wedge$ `RR knows` r `exists` $\wedge$ `RR knows` *file* `exists.`

The ability of p to allow q to pass the reading right is implicit in 2 due to the Del house rule in §4.2. Let us illustrate that on the following scenario. Alice lets Bob read her file Alice/Poem but not to pass the right to others:

3. `Alice: (Bob canRead Alice/Poem) to RR.`

She lets him pass along the right to read Alice/Recipe:

4. `Alice: (Bob tdOn` r `canRead Alice/Recipe) to RR.`

Bob allows Cathy to read Alice/Recipe and notifies Alice about that:

5.1. `Bob: (Cathy canRead Alice/Recipe) to RR,`
5.2. `Bob: (Cathy canRead Alice/Recipe) to Alice.`

As a result, Cathy can read Alice/Recipe. The proof that Cathy can read Alice/Recipe is routine and obvious but it may be instructive and so we provide it.

By 5.1 (and the appropriate Say2know house rule), we have

5.1′. `RR knows Bob said Cathy canRead Alice/Recipe.`

Similarly `Alice knows Bob said Cathy canRead Alice/Recipe`, which, by the Exist and KMon house rules, gives us

5.2′. `Alice knows Cathy exists.`

By Lemma 4.4.1, RR knows of Bob, Cathy and the file Alice/Recipe. By 4 and 5.2′, with r = Cathy, we have

4′. `RR knows Alice said Bob tdOn Cathy canRead Alice/Recipe,`

which allows to conclude that RR knows of Alice. By 2, with p = Alice, r = Cathy and file = Alice/Recipe (and taking into account that Alice is the owner of Alice/Recipe), we have

2′. `RR knows Alice tdOn Cathy canRead Alice/Recipe.`

Apply the Del house rule (with p = Alice and q = Bob) to 2′ to infer

6. `RR knows Alice tdOn Bob tdOn Cathy canRead Alice/Recipe.`

From 6 and 4′, by Lemma 4.4.2, we have

7. `RR knows Bob tdOn Cathy canRead Alice/Recipe.`

Similarly, from 7 and 5′, we have

8. `RR knows Cathy canRead Alice/Recipe,`

which means that Cathy indeed can read the file Alice/Recipe.

Example 5.2 (Post-authentication)**.** The policy of the previous example is employed for access control in a particular company. When user p connects from a remote system, he shows up as a remote user q, for example "User number 24 on remote machine Zelda." There is an authentication server A1S, whose job it is to authenticate q as being, in fact, user p. The read-rights manager RR trusts A1S on authentication but does not allow A1S to delegate authentication. If q is authenticated as p, then the read-rights manager RR gives q all the rights of p. All this is expressed by assertions

`RR: A1S tdOn`$_0$ q `authenticatedAs` p`,`
`RR:` q `canActAs` $p \leftarrow$ q `authenticatedAs` p`,`
`RR:` q `canSpeakAs` $p \leftarrow$ q `authenticatedAs` p`.`

Example 5.3 (Targeted communication)**.** There is a write-rights manager WR, with policy similar to that of RR. WR too trusts A1S to do the authentications. But the system requires only the password authentication for reading, while it requires the password and the smartcard authentication for writing. Accordingly, A1S places an assertion

`A1S (`q `authenticatedAs` p`) to RR`

when q authenticates as p using just a password, and A1S places assertion

`A1S (`q `authenticatedAs` p`) to RR`
`A1S (`q `authenticatedAs` p`) to WR,`

when q is authenticated as p using both a password and a smartcard.

One could model this without targeted communication, instead using two different infons `authenticatedByPasswordAs` and `authenticatedBySmartCardAs`, or using an infon `authenticatedAs` p `by` z with an extra variable z which may take value `password` or `smartCard`. But a system with targeted communication is more easily upgraded. Suppose the time comes to increase security requirements, and demand biometric authentication for writing. With targeted communication the system upgrade requires only a change in the authentication server A1S; the other components of the system may be left as they are.

Example 5.4 (Targeted communication)**.** A company has regular review of employees. The employee records are maintained by HR. A special Auditor sets up a review committee for every employee p; the members of the committee can read the record of p. The employee p should not know who is on his review committee. Accordingly HR places the assertion

`HR: (`q `canRead record(`p`)) to RR` $\leftarrow$
`Auditor said` q `is-on-review-committee-of` p`.`

We presume that HR owns the records of the employees, and so, in accordance with the

assertion 2 in the Delegation Example above,

RR knows HR tdOn q canRead record(p).

It is important that Auditor's assertions about the composition of the review committees are targeted to HR and not broadcast to all. Otherwise the employee p would know who is on his review committee.

Example 5.5 (Internal knowledge acquired a priori)**.** We return briefly to one of the examples in Section 2, to justify the fact that DKAL allows nesting of $\texttt{said}_d$ with $\texttt{said}_0$ preceding said. Recall that Chux is an online store. Chux has agreements on discounts with several companies. In particular employees of Fabricam participate in discount plan 5X4302. Chux trusts Fabricam on identifying its employees, and trusts Crypto to quote Fabricam on this matter. This is expressed precisely in assertions A10, A11, and A15 in Figure 4. The trust in Crypto in non-delegatable. In the example, Crypto exercises the trust, and this leads to knowledge K8 in Firgure 4 which we repeat below:

Chux knows Crypto $\texttt{said}_0$ Fabricam said Chris is an employee of Fabricam

Note that the first $\texttt{said}_d$ has subscript zero. The validation procedure performed by Crypto is not visible to DKAL. As far as the system is concerned, the server's speech is based on its internal knowledge. (In fact in this case it is based on no knowledge at all; the assertion leading to it, A15 in Figure 4, has an empty body.)

Example 5.6 (p tdOn $x + y \not\leq (p$ tdOn $x) + (p$ tdOn $y)$)**.** This example illustrates the difference between p tdOn $(x + y)$ and $(p$ tdOn $x) + (p$ tdOn $y)$. The latter expresses trust in p on each of x and y, while the former expresses trust on x and y bundled together. In DKAL neither one follows from the other.

Parcom is a national provider of both internet and cable service. It does not sign customers up directly, but has affiliated stores who do this. Parcom will provide internet service to customer q if an affiliated store says q signed up for the service, and similarly with cable. ITNat is a national chain of stores with which Parcom has an affiliation agreement:

1. Parcom: ITNat tdOn q signedFor cable.
2. Parcom: ITNat tdOn q signedFor internet.

Note that Parcom lets the national chain ITNat delegate the trust, for example to its own local affiliates.

Parcom is not allowed to bundle the internet and cable services into a single package. But ITNat is not aware of this, and makes the assertion:

3. ITNat: Store1 tdOn (q signedFor cable + q signedFor internet).

If one takes the rule p tdOn $x + y \leq (p$ tdOn $x) + (p$ tdOn $y)$, it follows from 1, 2, and 3 that:

Parcom knows Store1 tdOn (q signedFor cable + q signedFor internet).

Store1 can then sign a customer q for cable and internet together, but cannot sign q for just cable or just internet, so that the two services are bundled together.

DKAL therefore does not have the rule p tdOn $x + y \leq (p$ tdOn $x) + (p$ tdOn $y)$. Nor does DKAL have the converse rule, $(p$ tdOn $x) + (p$ tdOn $y) \leq p$ tdOn $x + y$; there are scenarios in which a principal is trusted on a package of infons, for example using a service and paying for it, but is not trusted on the infons separately.

6. Answering basic queries

This section is dedicated to the proof of Theorems 4.6 and 4.7, on the time complexity of answering basic queries.

Recall that the *width* of an assertion (respectively query) is the number of variables in the assertion (query). The *quotation depth* of an infon is defined by the conditions:

- If x is not a spoken infon or a sum, then $\texttt{quoteDepth}(x) = 0$.
- $\texttt{quoteDepth}(p\ \texttt{said}_d\ x) = \texttt{quoteDepth}(x) + 1$.
- $\texttt{quoteDepth}(x + y) = \max\{\texttt{quoteDepth}(x), \texttt{quoteDepth}(y)\}$.

The *quotation depth* of an assertion $A :_d x \leftarrow x_1, \ldots, x_n, con$ is the quotation depth of the infon x. The quotation depth of $A :_d x\ \texttt{to}\ p \leftarrow x_1, \ldots, x_n, con$ is one plus the quotation depth of x.

Let X be a substrate structure, let $\mathcal{A}$ be an authorization policy, and let $Q = (a\ \texttt{knows}_d\ t(v_1, \ldots, v_k))$ be a basic query. We intend to show that the answer to Q under $\mathcal{A}$ can be computed, and the computation time complexity is polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$, where δ is the maximal quotation depth of assertions in $\mathcal{A}$, and w is the maximal width of assertions in $\mathcal{A}$ and of the query Q.

Let Π be the logic program consisting of the assertions in $\mathcal{A}$ and of the house rules. We cannot directly compute $\Pi(X)$, since X is infinite, and the superstrate relations may be infinite. But there are a couple of initial reductions we can in general perform to concentrate on a part of X which is closer to being finite.

Let V consist of all the principals who own assertions in $\mathcal{A}$ and the principal a whose knowledge is to be queried about. We compute the knowledge of each of the principals in V, as they may direct speeches at one another, and so each may affect the knowledge of the others. But there is no need to compute the knowledge of principals outside V, as they have no assertions and cannot affect the knowledge of principals in V. Thus

Reduction 1: It is enough to compute the knowledge of w for principals $w \in V$.

Let B consist of the principals in V plus (the values of) all ground regular components of heads of assertions in $\mathcal{A}$. If $b \in B$, then it is possible that the existence of b will become known to some principal in V. For example suppose b is a ground regular component in the head of an assertion A: $x \leftarrow x_1, \ldots, x_n, con$ of form (As1). If at any stage in the computation of the fixed point, the body of the assertion evaluates to true, then $A\ \texttt{knows}\ x$ enters the fixed point, and since b is a ground regular component of x it follows that $A\ \texttt{knows}\ b\ \texttt{exists}$. The same is true in the case of an assertion A: $x\ \texttt{to}\ p \leftarrow x_1, \ldots, x_n, con$ of form (As2), except that here both the ground regular components of x and A itslef may become known to p.

In computing the fixed point, we must consider all regular elements that may become known to principals who own assertions in $\mathcal{A}$, as the variables in the assertions may range over these elements. The last paragraph shows that at the very least we better include all elements of B. We will prove that *no other* regular elements need be considered:

Reduction 2: We may without loss of generality restrict attention to the substructure of the substrate generated using synthetic functions and non-ground expressions in $\mathcal{A}$ and Q from the regular elements in B.

The proof of Reduction 2 will use the following safety conditions, which were imposed on assertions in Subsection 4.1:

- Variables range only over regular elements.

- Variables must be "knowledge bound" in the sense that if a variable v, other than the target of the assertion, occurs in the assertion, then the premise of the assertion must include knowledge of the infon v `exists`.
- Non-ground regular expressions in the assertion head must also be knowledge bound.

Reduction 2 is an important step, but it still leaves us with an infinite structure. The regular layer in the structure is the set B, and this set is finite since the authorization policy $\mathcal{A}$ is finite. But the synthetic layer is infinite. We now attempt to restrict further, by limiting depth in the synthetic layer.

Definition 6.1. The *depth* of x is defined by induction as follows:

- If x is regular then its depth is 0.
- If x is synthetic of the form $x_1 + x_2$ then its depth is the maximum of the depths of x_1 and x_2.
- If x is synthetic of the form $F(x_1, \ldots, x_k)$ with F a function other than $+$, then the depth of x is the maximum of the depths of $x_1, \ldots, x_k$, plus 1.

Thus the depth of an infon is the nesting depth of synthetic functions in the infon, except that uses of the function $+$ are not counted.

Let D be the maximal depth of an infon in $\mathcal{A}$. Let T be the set of all infons of depth $\leq D$, with regular components in B. The number of infons in T is infinite, but by eliminating repetitions in sums we may find a finite set of canonical representatives for all infons in T. Let S be this set. If we could restrict attention to only the synthetic elements in S, we would have reduced $\mathrm{ans}(Q)$ to a fixed point of a logic program over a finite structure, and such a fixed point can be computed, see Theorem A.3.

Ultimately we shall prove that this approach works. But there are difficulties which must be tackled. First, we need a way to compute the restriction of the ensue relation to infons of depth $\leq D$. Second, we have to reduce the set of infon we work with further; S, though finite, has size far greater than polynomial in $(\mathrm{length}(\mathcal{A}) + \mathrm{length}(Q))^{\delta+1+w}$.

We deal with both these difficulties in Subsection 6.1. Then in Subsection 6.2 we perform the reductions described above, and show how to compute the answer to a basic query.

6.1. **Downward closure under ensue.** We develop in this section an algorithm for producing, given a set of infons Z and another set I of infons of interest, the set Y of all infons in I which ensue sums of infons in Z. We shall use this algorithm later when computing answers to queries, to ensure that knowledge is closed under the rule KMon. In that context I will include all infons occurring in bodies of assertions placed by a principal A. Z will include all infons known to A in a given stage of the computation. We shall need to compute which infons in I ensue Z (those infons are then also known to A, and may cause some of A's assertions to fire). But we shall not need to compute the full ensue relation beyond these fragments.

Let D bound the depth of infons in Z and in I. It is tempting to simply follow the ensue house rules, restricted to infons of depth $\leq D$, and hope that this would compute the restriction of ensue to infons of this depth. The following example shows that this approach fails.

Consider the infons

(a) p `tdOn` r $\texttt{tdOn}_d$ `foo`,
(b) p `said` q `tdOn foo`,

(c) p `said` r `exists`,
(d) q `tdOn` r $\texttt{tdOn}_d$ `foo`,

and the infons of greater depth

(e) p `tdOn` q `tdOn` r $\texttt{tdOn}_d$ `foo`, and
(f) p `said` q `tdOn` r $\texttt{tdOn}_d$ `foo`

Note that (e) ensues (a)+(b) by Del and Exists (Exists and infon (b) are only needed to show that q `exists`), (f) ensues (b)+(c) by Del and SaidMon, and (d) ensues (e)+(f) by TrustApplication. Hence (d) ensues (a)+(b)+(c). But to derive this relationship from the house rules, we had to go through infons (e) and (f), which have greater depth than infons (a), (b), (c), and (d). In fact the relationship cannot be derived following the house rules without going through infons of greater depth.

Luckily we shall see that this example is in some sense the *only* reason to ever need infons of depths greater than D in computing ensue on infons of depths $\leq D$. We shall prove that the need for infons of greater depth can be eliminated with the addition of one rule handling this particular example:

wDel.1.

$$q \texttt{ tdOn } r \texttt{ tdOn}_d\ x \leq p \texttt{ tdOn } r \texttt{ tdOn}_d\ x + p \texttt{ said } q \texttt{ tdOn } x + p \texttt{ said } r \texttt{ exists}.$$

The rule does not affect the semantics of the ensue relation, since it can be derived using Del, Exists, SaidMon, and TrustApplication, following the example above but with x instead of `foo`. On the other hand the rule allows reaching (d) in the example above from (a)+(b)+(c) *directly*, without going through infons of greater depths.

Once we prove that any ensue relationship on infons of depth $\leq D$ can be derived using the house rules plus wDel.1 without going through infons of depths $> D$, we will have shown that the question of which infons in I ensue Z is decidable. But we wish to do more. We will show that it is decidable with time complexity polynomial in $|\text{length}(Z \cup I)| \cdot (2 \cdot |\texttt{regcomp}(Z)|)^{\delta+1}$, where δ bounds the quotation depth (which may well be smaller than full depth) of infons in Z. (Here $\text{length}(Z \cup I)$ and $\texttt{regcomp}(Z)$ are the sum of lengths of infons in $Z \cup I$, written as strings, and in the set of regular components of infons in Z, respectively.) For this it is not enough that the question of which infons in I ensue Z can be answered by following the ensue rules plus wDel.1 on infons of depth $\leq D$; there are too many infons of such depth. We will have to limit the rules so that the number of consequences of Z that they produce is bounded by a polynomial in $|\text{length}(Z \cup I)| \cdot (2 \cdot |\texttt{regcomp}(Z)|)^{\delta+1}$, but still the limited rules should allow deriving all consequences in I.

The rules most in need of limiting are Del and Sum. The number of consequences of Z that they produce, even when restricted to infons of depths $\leq D$, is exponential in D in the case of Del, and even greater in the case of Sum. It will be a while before we get to the stage of introducing the limited versions of these rules. But it is useful already here to introduce another consequence of Del:

wDel.2. $q \texttt{ tdOn}_d\ x \leq p \texttt{ tdOn } x + p \texttt{ said } q \texttt{ tdOn}_d\ x$.

This rule too does not result in any new ensue relationships. Indeed the rule is simply Lemma 4.4.3. But it allows a direct "Del free" deduction of an infon whose deduction would otherwise require Del, and will later make it easier to keep tab on uses of Del.

Recall that we have sets Z and I of infons, and wish to compute which infons in I ensue Z. We do not wish to compute any more of the ensue relation than is required for this purpose.

An ensue rule is *proper* if the body of the rule is empty, or else it is a substrate constraint. Proper ensue rules can be turned into deduction rules on infons. The following list includes the deduction rules resulting from the proper house ensue rules. In each rule, the infon to the left of $\dashv$ ensues the sum of the infons to the right. The list omits the rule resulting from SaidSum, as we shall obtain it indirectly later. The deduction rule Sum.3 in the list paraphrases the third part of ESum to the context of a deduction.

(Exists)	q `exists` $\dashv x$, for q and x so that q `regcomp` x.
(Said0∞)	p `said` $x \dashv p$ `said`$_0$ x.
(SelfQuote)	p `said`$_d$ $x \dashv p$ `said`$_d$ p `said`$_d$ x.
(Trusted0∞)	p `tdOn`$_0$ $x \dashv p$ `tdOn` x.
(TrustApplication)	$x \dashv p$ `tdOn`$_d$ x, p `said`$_d$ x.
(Del)	p `tdOn` (q `tdOn`$_d$ x) $\dashv p$ `tdOn` x, q `exists`.
(Del$^-$)	p `tdOn`$_d$ $x \dashv p$ `tdOn`$_d$ p `tdOn`$_d$ x.
(Role.1)	p *attribute* $\dashv q$ *attribute*, p `canActAs` q.
(Role.2)	q *speech* $\dashv p$ *speech*, p `canActAs` q.
(Sum.1)	$x \dashv x + y$.
(Sum.2)	$y \dashv x + y$.
(Sum.3)	$x + y \dashv x$, y.

Non-proper house rules pose a problem for us: their bodies refer to the ensue relation, and we do not wish to have to compute whether they evaluate to true. The only non-proper ensue house rules are EOrder, ESum, and SaidMon. We already rephrased ESum in a way that removes its hypothesis, and in particular makes it proper. EOrder will hold automatically in our context. But we must find a way to get around uses of SaidMon.

Let $x \dashv y_1, \ldots, y_k$ be a deduction rule. A *SaidMon extension* of the deduction rule is any deduction rule of the form *pref* $x \dashv$ *pref* $y_1, \ldots,$ *pref* y_k, where *pref* is a *prefix*, by which we mean an expression of the form p_1 `said`$_{d_1}$ $\ldots$ p_i `said`$_{d_i}$.

To give an example, a `said` b `tdOn`$_0$ `foo` $\dashv a$ `said` b `tdOn foo` is a SaidMon extension of b `tdOn`$_0$ `foo` $\dashv b$ `tdOn foo`.

Definition 6.2. A *basic ensue-deduction* from a set X of infons is a sequence of infons $x_1, \ldots, x_r$ so that for each $i \leq r$, either $x_i \in X$, or there are $j_1, \ldots, j_k < i$ so that $x_i \dashv x_{j_1}, \ldots, x_{j_k}$ is a SaidMon extension of one of the deduction rules Exists, Said0∞, SelfQuote, Trusted0∞, TrustApplication, Del, Del$^-$, Role.1–2, and Sum.1–3. A *basic ensue-deduction of* y is a basic ensue-deduction $x_1, \ldots, x_r$ so that $x_r = y$.

In this subsection we only deal with ensue-deductions, and so we will refer to them simply as deductions. The deductions in the definition above are called basic because they use deduction rules resulting very directly from the ensue house rules. (Later we shall have deductions that use slightly different rules.) The inclusion of Sum.1–3 allows us to deduce $x + y$ once we have deduced each of x and y, and conversely deduce each of x and y once we have deduced $x + y$. The inclusion of SaidMon extensions of Sum.3 allows us to deduce

pref $(x + y)$ once we have deduced each of *pref* x and *pref* y. In other words it subsumes the house rule SaidSum.

We write $y \dashv^b Z$ to mean that there is a basic ensue deduction of y from the set Z. We write $Y \dashv^b Z$ for a set Y of infons to mean that each of the infons in Y can be deduced from Z. We write $y \dashv^b z$ for an infon z to mean that $y \dashv^b Z$ where Z is the singleton set $\{z\}$, and similarly with $Y \dashv^b z$.

Lemma 6.3. *$y \leq z$ iff $y \dashv^b z$.*

Proof. The right-to-left direction is straightforward by induction on the length of the given deduction. We prove the left-to-right direction. It suffices to show that $\dashv^b$ is a fixed point for the ensue house rules of Section 4. Indeed suppose that $\dashv^b$ is a fixed point. Since $\leq$ is the least fixed point, it is a subset of $\dashv^b$, and the left-to-right direction of the lemma follows.

$\dashv^b$ is a fixed point for EOrder since deductions can be composed. Because of the inclusion of Sum.1–3, and the fact that SaidMon extensions of Sum.3 subsume SaidSum, $\dashv^b$ is a fixed point for both ESum and SaidSum. Each of the remaining ensue house rules except SaidMon has a corresponding deduction rule, and from this it follows that $\dashv^b$ is a fixed point for these rules. Thus it remains to show that $\dashv^b$ is a fixed point for SaidMon. To this end, we assume that $y \dashv^b z$ and we show that p $\texttt{said}_d$ $y \dashv^b p$ $\texttt{said}_d$ z. Let $x_1, \ldots, x_r$ be a deduction of y from $\{z\}$. Then the sequence p $\texttt{said}_d$ x_1, …, p $\texttt{said}_d$ x_r is still a deduction, just with longer prefixes for the SaidMon extensions of the deduction rules used, and it is a deduction of p $\texttt{said}_d$ y from $\{p$ $\texttt{said}_d$ $z\}$. □

We now have an order, $\dashv^b$, defined by means of deduction rules which avoid the non-proper SaidMon, and still precisely captures the order $\leq$.

Recall that the depth of an infon is the nesting level of synthetic functions other than + in the infon. (The precise definition appears in the preamble to this section.) A deduction is *depth-bounded* by D if all the infons in the deduction have depth D or less. We write $y \dashv^b_D Z$ to mean that there is a basic deduction of y from Z, which is depth-bounded by D. We use $\dashv^b_D$ on sets Y and infons z as we do with $\dashv^b$.

Earlier in the subsection we saw that in general there may be infons y and z so that both y and z have depth $\leq D$, $y \dashv^b z$, and yet $y \not\dashv^b_D z$. This poses a problem in attempting to compute whether $y \dashv^b z$. To overcome this problem, we generalize the notion of deduction, adding the following deduction rules that correspond to the ensue rules wDel.1 and wDel.2 mentioned above.

(wDel.1) q $\texttt{tdOn}$ r $\texttt{tdOn}_d$ $x \dashv p$ $\texttt{tdOn}$ r $\texttt{tdOn}_d$ x, p $\texttt{said}$ q $\texttt{tdOn}$ x, p $\texttt{said}$ r $\texttt{exists}$.

(wDel.2) q $\texttt{tdOn}_d$ $x \dashv p$ $\texttt{tdOn}$ x, p $\texttt{said}$ q $\texttt{tdOn}_d$ x.

Definition 6.4. An *enhanced ensue-deduction* from a set X of canonical infons is a sequence $x_1, \ldots, x_r$ satisfying the conditions in Definition 6.2 with wDel.1 and wDel.2 added to the list of deduction rules whose SaidMon extensions may be used in the deduction.

We use the notation $y \dashv^e Z$ to mean that there is an enhanced deduction of y from Z. We use $y \dashv^e_D Z$ if the deduction is depth-bounded by D. We use $\dashv^e_D$ and $\dashv^b_D$ on sets as we do with $\dashv^b$ and $\dashv^b_D$.

Claim 6.5. *$y \leq z$ iff $y \dashv^b z$ iff $y \dashv^e z$.*

Proof. We already know $y \leq z \Leftrightarrow y \dashv^b z$, and $y \dashv^b z \Rightarrow y \dashv^e z$ is clear. The implication $y \dashv^e z \Rightarrow y \leq z$ holds because all the rules allowed in enhanced deductions, including wDel.1 and wDel.2, are true of the ensue relation. □

We aim to prove that if y and z have depths $\leq D$, then $y \leq z$ iff $y \dashv^e_D z$. Thus the depth bound that does not hold for ensue deductions without wDel.1–2, does hold when these rules are added. This will later help us to compute whether $y \leq z$.

Remark 6.6. In the proofs below we keep track of any uses of the deduction rules Del and Sum.3. The reason for this care will become clear later.

Definition 6.7. A *germ* for an infon y is a set X of infons so that y can be deduced from X using only the SaidMon extensions of deduction rules Del, Trust0∞, and Sum.3.

We say that X has depth $\leq D$ if all infons in X have depth $\leq D$. We sometimes abuse notation, and write $X + z$ for a set X of infons and an infon z. We mean the set of infons $X \cup \{z\}$. We also write *pref* X for the set $\{\mathit{pref}\ x \mid x \in X\}$. We use H `exist` to abbreviate the set of infons $\{r$ `exists` $\mid r \in H\}$. The typical example of a germ is the following: $X = \mathit{pref}\ p$ `tdOn` $x + \mathit{pref}\ H$ `exist` is a germ for every infon $y = \mathit{pref}\ p$ $\texttt{tdOn}_{d_0}\ q_1\ \texttt{tdOn}_{d_1}\ \dots q_n\ \texttt{tdOn}_{d_n}\ x$, where $q_1, \dots, q_n \in H$. More generally unions of sets such as X are germs for sums of infons such as y.

Remark 6.8. If X is a germ for infon $y = \mathit{pref}\ v$, then X contains a set of the form *pref* U, with U a germ for v. The reason is that infons which do not begin with *pref* can safely be erased from the deduction of y from X. The deduction only uses SaidMon instances of Del, Trust0∞, and Sum.3, and none of these can produce an infon which starts with *pref* from infons which do not.

Claim 6.9. *If X is a germ for y, then $y \dashv^e_D X$ for any D greater than or equal to the depth of y.*

Proof. Since X is a germ for y, there is a deduction of y from X using SaidMon extensions of Del, Trust0∞, and Sum.3. These rules do not decrease depth. So any infons of depth $> D$ are of no consequence in deducing y, and can be removed from the deduction. The resulting deduction witnesses that $y \dashv^e_D X$. □

Definition 6.10. A deduction rule R is *depth-conservative on germs* if for every instance $u \dashv u_1, \dots, u_k$ of the rule, the following holds: Suppose $U_1, \dots, U_k$ are germs for $u_1, \dots, u_k$. Suppose that $U_1, \dots, U_k$ all have depths $\leq D$. Then there is U so that:

(1) U is a germ for u.
(2) $U \dashv^e_D U_1 \cup \dots \cup U_k$. (In particular U has depth $\leq D$.)

We are being very general in condition 2. But typically deduction witnessing it is very simple, using mainly the rule R referenced in the definition. The only exception is in the case of the rule TrustApplication, where the enhanced deduction witnessing condition 2 will use wDel.1–2.

Claim 6.11. *If a rule R is depth-conservative on germs, then so are all its SaidMon extensions.*

Proof. Let $y \dashv y_1, \dots, y_k$ be a SaidMon extension of R. Say $y = \mathit{pref}\ u$ and $y_i = \mathit{pref}\ u_i$. Suppose Y_i are germs for y_i. Using Remark 6.8 we may, by reducing germs Y_i if needed,

assume that each Y_i has the form *pref* U_i, with U_i a germ for u_i. Apply Definition 6.10 to the instance $u \dashv u_1, \ldots, u_k$ of R, to obtain U so that

(1) U is a germ for u, and
(2) $U \dashv^e_D U_1 \cup \cdots \cup U_k$.

Set $Y =$ *pref* U. Prepending *pref* to the deductions witnessing 1 and 2, it follows that Y is a germ for y, and $Y \dashv^e_D Y_1 \cup \cdots \cup Y_k$. □

Claim 6.12. *The rules Sum.1–3, Exists, Said*0∞*, SelfQuote, Trusted*0∞*, Del, Del*$^-$*, and Role.1–2 are all depth-conservative on germs.*

Proof. The case of Sum.3 is obvious. If U_1 is a germ for u_1, and U_2 is a germ for u_2, then $U = U_1 \cup U_2$ is a germ for $u = u_1 + u_2$.

The rules Sum.1 and Sum.2 are slightly more involved. We handle Sum.1. Let U_1 be a germ for $x + y$. If the infon $x + y$ itself is an element of U_1 then $x \dashv^e_D U_1$ by an application of a SaidMon extension of an instance of Sum.1, and we can take $U = \{x\}$. If the infon $x + y$ is not an element of U_1, then it must be that both x and y are deducible from U_1 using SaidMon extensions of Del, Trust0∞, and Sum.3 (for otherwise it would be impossible to deduce $x + y$ using these rules). Thus U_1 is a germ for x, and we may take $U = U_1$.

The case of Exists is obvious, noting that if q `regcomp` x and U_1 is a germ for x, then q is a regular component of one of the infons in U_1, and therefore q `exists` $\dashv^e_D U_1$, so we can take $U = \{q$ `exists`$\}$.

Consider the case of Said0∞. Let $u = (p$ `said`$_0$ $x) \dashv (p$ `said` $x) = u_1$ be an instance of this rule. Suppose U_1 is a germ for u_1. By remark 6.8 we may assume that U_1 has the form p `said` X. Then $U = p$ `said`$_0$ X is a germ for u, and $U \dashv^e_D U_1$ by an application of Said0∞.

A similar argument works for the rule SelfQuote.

Let us now handle Trust0∞. Let $u = (p$ `tdOn`$_0$ $x) \dashv (p$ `tdOn` $x) = u_1$ be an instance of the rule. Let U_1 be a germ for u_1. Then U_1 itself is also a germ for u, so one can take $U = U_1$.

An argument similar to that of the previous paragraph works for Del.

Let $u = (p$ `tdOn`$_d$ $x) \dashv (p$ `tdOn`$_d$ p `tdOn`$_d$ $x) = u_1$ be an instance of Del$^-$. Let U_1 be a germ for u_1. If p `tdOn`$_d$ p `tdOn`$_d$ x belongs to U_1, then $u \dashv^e_D U_1$ by an application of Del$^-$, and we may take $U = \{u\}$. If $d = 0$ and p `tdOn` p `tdOn`$_d$ x belongs to U_1 then $u \dashv^e_D U_1$ by an application of Trust0∞, and again we can take $U = \{u\}$. If neither of these two cases holds, then it must be that p `tdOn` x is deducible from U_1 using SaidMon extensions of Del, Trust0∞, and Sum.3 (for otherwise it would be impossible to deduce p `tdOn`$_d$ p `tdOn`$_d$ x from U_1 using these rules, and U_1 would not be a germ for the infon). So U_1 is a germ for (p `tdOn` $x) = u$, and we may take $U = U_1$.

Consider finally the Role rules. Let us start with Role.2. Let $u = (q$ *speech*$) \dashv p$ *speech*, p `canSpeakAs` q. Let $u_1 = (p$ *speech*$)$ and let $u_2 = (p$ `canSpeakAs` $q)$. Let U_1 and U_2 be germs for u_1 and u_2. u_2 cannot be deduced non-trivially using just Del, Trust0∞, and Sum.3, so it must be that u_2 belongs to U_2. Using Remark 6.8 we may assume that U_1 has the form p `said`$_d$ X. Set $U = (q$ `said`$_d$ $X)$. Then U is a germ for u, and $U \dashv^e_D U_1 \cup U_2$ by an application Role.2.

Passing now to Role.1, suppose $u = (p$ *attribute*$) \dashv q$ *attribute*, p `canActAs` q. Let $u_1 = (q$ *attribute*$)$ and let $u_2 = (p$ `canActAs` $q)$. Let U_1 and U_2 be germs for u_1 and u_2. Again it must be that u_2 belongs to U_2. If u_1 belongs to U_1, then $u \dashv^e_D U_1 + U_2$ using Role.1, and we may take $U = \{u\}$. So suppose u_1 does not belong to U_1. Since u_1 is deducible from U_1 using SaidMon extensions of Del, Trust0∞, and Sum.3, and has the form q *attribute*, it must be

that *attribute* has the form $\texttt{tdOn}_d\ y$, and U_1 has some infons $q\ \texttt{tdOn}\ x$ and $H\ \texttt{exist}$ from which one can deduce $u_1 = q\ \texttt{tdOn}_d\ y$ with applications of Del and Trust0∞. Set $U = p\ \texttt{tdOn}\ x, H\ \texttt{exist}$. Then U is a germ for $u = p\ \texttt{tdOn}_d\ y$, and $U \dashv^e_D U_1 \cup U_2$ by an application of the Role.1. □

Remark 6.13. In each of the cases in the proof of Claim 6.12 we were given an instance $u \dashv u_1, \ldots, u_k$ of one of the rules listed in the claim, and germs U_i of depth $\leq D$ for u_i, and we produced a germ U for u so that $U \dashv^e_D U_1 \cup \cdots \cup U_k$. Note that, in each of the cases in the proof, the deduction witnessing $U \dashv^e_D U_1 \cup \cdots \cup U_k$ made no use of either Del or Sum.3.

Claim 6.14. *The rule TrustApplication too is depth-conservative on germs.*

Proof. Let $u \dashv p\ \texttt{tdOn}_d\ u, p\ \texttt{said}_d\ u$ be an instance of TrustApplication. Let U_1 and U_2 be germs for $p\ \texttt{tdOn}_d\ u$ and $p\ \texttt{said}_d\ u$ respectively, of depths $\leq D$. We shall find a germ U for u so that $U \dashv^e_D U_1 \cup U_2$. The deduction witnessing this last fact will have to use Del and (in some cases) Sum.3, and for future reference we shall mark each use of these rules, and list the uses after the proof, in Remark 6.15.

First Case. We start with the case that, for some $e \geq d$, the infon $p\ \texttt{tdOn}_e\ u$ belongs to U_1 (so that U_1 is a germ for $p\ \texttt{tdOn}_d\ u$ in a very trivial sense; the infon either belongs to the set, or can be deduced from the set using just Trust0∞). In this case D is at least the depth of $p\ \texttt{tdOn}\ u$, which is equal to the depth of $p\ \texttt{said}_d\ u$. We have:

$$\begin{aligned} u &\dashv^e_D \{p\ \texttt{tdOn}_d\ u\,,\ p\ \texttt{said}_d\ u\} \\ &\dashv^e_D \{p\ \texttt{tdOn}_d\ u\} \cup U_2 \qquad \text{(a)} \\ &\dashv^e_D \{p\ \texttt{tdOn}_e\ u\} \cup U_2 \\ &\subseteq\ U_1 \cup U_2. \end{aligned}$$

The first $\dashv^e_D$ holds by a use of TrustApplication, noting that all infons involved have depths $\leq D$. The second $\dashv^e_D$ holds by Claim 6.9, as U_2 is a germ for $p\ \texttt{said}_d\ u$, and the infon has depth $\leq D$. (This second deduction may use Del and Sum.3, and for this reason we tagged it. We will return to the use of Del and Sum.3 later, in Remark 6.15.) The third and final $\dashv^e_D$ holds either trivially (if $e = d$) or by a use of Trust0∞. Let $U = \{u\}$. Then U is certainly a germ for u, and by the deduction above, $U \dashv^e_D U_1 \cup U_2$. This competes the proof of Claim 6.14 in the first case. □(First Case)

Suppose from now on that the condition of the first case fails, i.e., neither $p\ \texttt{tdOn}_0\ u$ nor $p\ \texttt{tdOn}\ u$ belongs to U_1. Since U_1 is a germ for $p\ \texttt{tdOn}_d\ u$, it follows that there must be $l \geq 1$, infon f, and a set $H = \{q_1, \ldots, q_l\}$, so that:

(i) $p\ \texttt{tdOn}_d\ u$ has the form $p\ \texttt{tdOn}_d\ q_1\ \texttt{tdOn}_{d_1}\ \ldots\ q_l\ \texttt{tdOn}_{d_l}\ f$.
(ii) $p\ \texttt{tdOn}\ f + H\ \texttt{exist}\ \subseteq U_1$.

Let x be $q_2\ \texttt{tdOn}_{d_2}\ \ldots\ q_l\ \texttt{tdOn}_{d_l}\ f$. Then u has the form $q_1\ \texttt{tdOn}_{d_1}\ x$, and U_2 is a germ for $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_{d_1}\ x$. By (i) and (ii), U_1 is a germ for $p\ \texttt{tdOn}\ x$.

Second Case. Suppose that $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_e\ x$ belongs to U_2 for some $e \geq d_1$ (so that U_2 is a germ for $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_{d_1}\ x$ in a trivial sense; the infon either belongs to the set or can be deduced from the set using just Trust0∞). Then again D is at least the depth of $p\ \texttt{said}_d$

u, and:

$$\begin{aligned} u \;&=\; q_1\ \texttt{tdOn}_{d_1}\ x \\ &\dashv^e_D\ \{p\ \texttt{tdOn}\ x\,,\ p\ \texttt{said}\ q_1\ \texttt{tdOn}_{d_1}\ x\} \\ &\dashv^e_D\ U_1 \cup \{p\ \texttt{said}\ q_1\ \texttt{tdOn}_{d_1}\ x\} \\ &\dashv^e_D\ U_1 \cup \{p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_e\ x\} \\ &\subseteq\ U_1 \cup U_2. \end{aligned} \tag{b}$$

The first $\dashv^e_D$ uses wDel.2. The second uses Claim 6.9, noting that U_1 is a germ for $p\ \texttt{tdOn}\ x$ and the infon has depth $< D$. The third uses (one of, both, or neither, depending on the values of d and e) Said0∞ and Trust0∞. (Only the second, tagged deduction uses Del and Sum.3.) We can now let $U = \{u\}$ completing the proof of Claim 6.14 in the second case. □(Second Case)

Suppose from now on that neither $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_0\ x$ nor $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ x$ belong to U_2. Since U_2 is a germ for $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}_{d_1}\ x$, it follows that there must be $k \geq 1$ and infon g so that:

(iii) $u = q_1\ \texttt{tdOn}_{d_1}\ x$ has the form $q_1\ \texttt{tdOn}_{d_1}\ q_2\ \texttt{tdOn}_{d_2}\ \ldots\ q_k\ \texttt{tdOn}_{d_k}\ g$.
(iv) $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g + p\ \texttt{said}_d\ I\ \texttt{exist}\ \subseteq U_2$, where $I = \{q_2, \ldots, q_k\}$.

Note that the principals q_i for $i \leq \min(l, k)$ in conditions (i) and (iii) must match. We have the following picture:

$$\begin{aligned} p\ \texttt{tdOn}_d\ u = p\ \texttt{tdOn}_d\ q_1\ \texttt{tdOn}_{d_1}\ \ldots\ldots\ q_l\ \texttt{tdOn}_{d_l}\ f, \text{ and} \\ u = \qquad\qquad q_1\ \texttt{tdOn}_{d_1}\ \ldots\ldots\ q_k\ \texttt{tdOn}_{d_k}\ g. \end{aligned}$$

Third Case. Suppose that $l \geq k$. In this case U_1 is a germ for $p\ \texttt{tdOn}\ g$. D is greater than the depth of $p\ \texttt{tdOn}\ g$, since the infon of greater depth $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g$ belongs to U_2. Thus:

$$\begin{aligned} q_1\ \texttt{tdOn}\ g\ &\dashv^e_D\ \{p\ \texttt{tdOn}\ g\,,\ p\ \texttt{said}\ q_1\ \texttt{tdOn}\ g\} \\ &\dashv^e_D\ U_1 \cup \{p\ \texttt{said}\ q_1\ \texttt{tdOn}\ g\} \\ &\dashv^e_D\ U_1 \cup \{p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g\} \\ &\subseteq\ U_1 \cup U_2. \end{aligned} \tag{c}$$

The first $\dashv^e_D$ uses wDel.2, the second uses Claim 6.9, and the third uses Said0∞ (or nothing, if $d = \infty$). (Only the second, tagged deduction may use Del and Sum.3.) We can now let $U = \{q_1\ \texttt{tdOn}\ g\} \cup \{q_2, \ldots, q_k\}\ \texttt{exist}$. Then $U \dashv^e_D U_1 \cup U_2$ by the above, and U is a germ for $u = q_1\ \texttt{tdOn}_{d_1}\ \ldots\ q_k\ \texttt{tdOn}_{d_k}\ g$. So the proof of Claim 6.14 is complete in this case. □(Third Case)

Fourth Case. Suppose that $l \leq k - 1$. This is the final case in the proof of Claim 6.14. Let $k^* = l + 1$ and let $g^* = q_{l+2}\ \texttt{tdOn}_{d_{l+2}}\ \ldots\ q_k\ \texttt{tdOn}_{d_k}\ g$. The connection between g^* and f is presented in the following picture. f is equal to $q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*$.

$$\begin{aligned} p\ \texttt{tdOn}_d\ u = p\ \texttt{tdOn}_d\ q_1\ \texttt{tdOn}_{d_1}\ \ldots\ldots\ q_l\ \texttt{tdOn}_{d_l}\ f, \\ u = \qquad\qquad q_1\ \texttt{tdOn}_{d_1}\ \ldots\ldots\ q_l\ \texttt{tdOn}_{d_l}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*. \end{aligned}$$

Note that U_2 is a germ for $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g^*$, as $k^* \leq k$. D is at least the depth of $p\ \texttt{tdOn}\ f$ and $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g^*$ (both infons have the same depth), since $p\ \texttt{tdOn}\ f$ belongs to U_1.

We have:

$$
\begin{aligned}
q_1\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^* \ &\dashv^e_D \{p\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*, p\ \texttt{said}\ q_{l+1}\ \texttt{exists},\\
&\qquad\qquad p\ \texttt{said}\ q_1\ \texttt{tdOn}\ g^*\}\\
&\dashv^e_D \{p\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*, p\ \texttt{said}_d\ q_{l+1}\ \texttt{exists},\\
&\qquad\qquad p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g^*\}\\
&\dashv^e_D \{p\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*, p\ \texttt{said}_d\ q_{l+1}\ \texttt{exists}\} \cup U_2 \qquad \text{(d)}\\
&= \{p\ \texttt{tdOn}\ f\} \cup U_2\\
&\subseteq U_1 \cup U_2.
\end{aligned}
$$

The first $\dashv^e_D$ uses wDel.1, with $x = g^*$, $q = q_1$, $r = q_{l+1}$, and $d = d_{l+1}$. The second $\dashv^e_D$ uses Said0∞ if $d = 0$, and nothing if $d = \infty$. The third $\dashv^e_D$ uses Claim 6.9, to deduce $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g^*$ from its germ U_2. (Only the third, tagged deduction may use Del and Sum.3.) Let $U = \{q_1\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*\} \cup \{q_2, \ldots, q_l\}\ \texttt{exist}$. Then $U \dashv^e_D U_1 \cup U_2$ by condition (ii) and the deduction above, and U is a germ for u. This completes the final, fourth case. □(Fourth Case)

□(Proof of Claim 6.14)

Remark 6.15. In each of the cases in the proof of Claim 6.14 we produced a germ U for u, so that $U \dashv^e_D U_1 \cup U_2$. The deduction witnessing that $U \dashv^e_D U_1 \cup U_2$ may use Del and Sum.3, but these rules were needed only in the following places (marked here by the letters used to tag them during the proof):

(a) Deducing infon $p\ \texttt{said}_d\ u$ from $\{p\ \texttt{tdOn}_d\ u\} \cup U_2$, where U_2 is a germ for the infon, and the infon has depth $\leq D$.
(b) Deducing infon $p\ \texttt{tdOn}\ x$ from $U_1 \cup \{p\ \texttt{said}\ q_1\ \texttt{tdOn}_e\ x\}$, where U_1 is a germ for the infon, and the infon has depth $< D$.
(c) Deducing infon $p\ \texttt{tdOn}\ g$ from $U_1 \cup \{p\ \texttt{said}\ q_1\ \texttt{tdOn}\ g\}$, where U_1 is a germ for the infon, and the infon has depth $< D$.
(d) Deducing infon $p\ \texttt{said}_d\ q_1\ \texttt{tdOn}\ g^*$ from $\{p\ \texttt{tdOn}\ q_{l+1}\ \texttt{tdOn}_{d_{l+1}}\ g^*\} \cup U_2$, where U_2 is a germ for the infon, and the infon has depth $\leq D$.

We are now ready to prove that, if y and z have depths $\leq D$, and $y \dashv^b z$, then $y \dashv^e_D z$. We prove the following, stronger result:

Lemma 6.16. *Let Z be a set of infons of depths $\leq D$. Let y be an infon. Suppose that $y \dashv^b Z$. Then there is a germ Y for y so that $Y \dashv^e_D Z$.*

Proof. Fix y and Z. The lemma is proved by induction on the length of the basic deduction of y from Z, using Claims 6.12 and 6.14 to replace the rules applied in the basic deduction by depth-conservative deductions given by the claims.

To be precise, let $x_1, \ldots, x_r = y$ be the shortest basic deduction of y from Z. Let R be the last rule applied in the deduction. Let $j_1, \ldots, j_k < r$ be such that in the last step of the deduction, y is derived from $x_{j_1}, \ldots, x_{j_k}$. So $y \dashv x_{j_1}, \ldots, x_{j_k}$ is a SaidMon extension of R. By induction there are germs X_i for x_{j_i} so that:

(i) $X_i \dashv^e_D Z$.

By Claims 6.12 and 6.14, R is depth-conservative on germs. By Claim 6.11, SaidMon extensions of R are also depth-conservative on germs. Applying this to the instance $y \dashv x_{j_1}, \ldots, x_{j_k}$ and the germs X_i for x_{j_i} it follows that there is a germ Y for y so that:

(ii) $Y \dashv^e_D X_1 \cup \cdots \cup X_k$.

Combining the deductions in (i) and (ii) we get $Y \dashv^e_D Z$. □

Corollary 6.17. *Let y and z be infons of depth $\leq D$. Then $y \leq z$ iff $y \dashv^e_D z$.*

Proof. The right-to-left direction is obvious as $y \dashv^e_D z \Rightarrow y \dashv^e z \Rightarrow y \leq z$. For the left to right direction, suppose $y \leq z$. Then $y \dashv^b z$ by Lemma 6.3. By the last lemma, there is a germ Y for y so that $Y \dashv^e_D z$. Since y has depth $\leq D$, $y \dashv^e_D Y$ by Claim 6.9. We have $y \dashv^e_D Y \dashv^e_D z$, so by composing deductions, $y \dashv^e_D z$. □

Corollary 6.17 is already enough to conclude that whether or not $y \leq z$ can be decided by an algorithm. The algorithm lists all infons which are $\dashv^e_D z$, where D is the maximum depth of y and z, avoiding duplications in sums. So long as duplications in sums are avoided, there are only finitely many infons to go over.

Still, the number of infons to go over is too large. We now obtain a more efficient algorithm for deciding whether $y \leq z$, by more carefully sifting the relevant infons.

Let `relevant` be a new synthetic function of type Info → Info. We shall add to the computation of whether $y \dashv^e Z$ a computation of whether certain infons are relevant (meaning relevant to the computation), and restrict Del and Sum.3 to the following weaker rules:

(wDel.3) p `tdOn` q `tdOn`$_d$ $x \dashv p$ `tdOn` x, q `exists`, `relevant`$(p$ `tdOn` q `tdOn`$_d$ $x)$.

(wSum.3) $x + y \dashv x$, y, `relevant`$(x + y)$.

In a deduction, wSum.3 allows deducing infon $x+y$ from infons x, y, and `relevant`$(x+y)$. It is the addition of `relevant`$(x+y)$ to the requirement list that differentiates wSum.3 from the unrestricted rule Sum.3. A similar addition requiring the relevance of the conclusion differentiates wDel.3 from the unrestricted Del.

We view `relevant`(y) as a "virtual" infon; it is only important internally, to the computation. Most of the house rules are such that they will not produce "real" infons (meaning infons in the original vocabulary) from virtual ones. The exception is Exists, and we therefore weaken it to:

(wExists) q `exists` $\dashv x$, for q and x so that q `regcomp` $x \wedge$ `original`(x).

Here `original`(x) holds iff x is in the original vocabulary, without `relevant`. Computation of `original`(x) is easy, and without loss of generality we may assume that the relation is a substrate relation.

The virtual infons *pref* `relevant`(x) are added only as a technical tool in computing the fixed point. The following rules allow deducing such infons during the computation. The rules are designed to allow deducing enough virtual infons so as to compensate for the added

restriction in wDel.3 and wSum.3.

(Relevant.1) $p\ \texttt{said}_d\ \texttt{relevant}(x) \dashv p\ \texttt{tdOn}_d\ x,\ p\ \texttt{said}_d\ y,$ if $\texttt{quoteDepth}(y) \geq \texttt{quoteDepth}(x)$.

(Relevant.2) $\texttt{relevant}(p\ \texttt{tdOn}\ x) \dashv p\ \texttt{said}\ q\ \texttt{tdOn}_d\ x.$

(Relevant.3a) $\texttt{relevant}(x) \dashv \texttt{relevant}(x + y).$

(Relevant.3b) $\texttt{relevant}(y) \dashv \texttt{relevant}(x + y).$

(Relevant.4) $p\ \texttt{said}_d\ \texttt{relevant}(x) \dashv \texttt{relevant}(p\ \texttt{said}_d\ x),\ p\ \texttt{exists}.$

(Relevant.5) $\texttt{relevant}(p\ \texttt{tdOn}\ x) \dashv \texttt{relevant}(p\ \texttt{tdOn}_0\ x).$

(Relevant.6) $\texttt{relevant}(q\ \texttt{tdOn}_d\ x) \dashv \texttt{relevant}(p\ \texttt{tdOn}_d\ x),\ q\ \texttt{exists}.$

(Relevant.7) $\texttt{relevant}(p\ \texttt{tdOn}\ x) \dashv \texttt{relevant}(p\ \texttt{tdOn}\ q\ \texttt{tdOn}_d\ x).$

The first rule expresses the intuition that if $p\ \texttt{tdOn}_d\ x$, and p says anything at all of the same quotation depth as x or of greater quotation depth, then it is important to deduce whether he says x (in which case we should later deduce x). The second rule expresses the intuition that if p says that $q\ \texttt{tdOn}_d\ x$, then it is important to deduce whether $p\ \texttt{tdOn}\ x$ (in which case we should later deduce $q\ \texttt{tdOn}_d\ x$). The remaining rules extract $+$ and $\texttt{said}_d$ from `relevant`, close relevance downward, and (in the case of 5 and 6) add some sideways closure. They are important for technical reasons.

Definition 6.18. A *careful ensue-deduction* from a set X of canonical infons is a sequence $x_1, \ldots, x_r$ satisfying the conditions in Definition 6.2 with the following modification: Del, Sum.3, and Exists are removed from the list of rules whose SaidMon extensions may be used in the deduction, replaced by wDel.1–3, wSum.3, and wExists, and Relevant is added.

A careful deduction is thus a deduction using the rules allowed for an enhanced deduction except for Del, Sum.3, and Exists; using restricted counterparts wDel.3, wSum.3, and wExists for these rules; and using Relevant. We write $y \dashv^c Z$ to indicate that there is a careful deduction of y from Z. Note that there is no need to consider depth restrictions, since none of the rules allowed in a careful deduction increases depth.

We shall see that the time complexity of computing whether $y \dashv^c Z$ only has the quotation depth of Z in the exponent, and is polynomial in all other parameters. For y and Z in the original vocabulary that does not include `relevant`, we shall see that $y \dashv^b Z$ iff $y \dashv^c Z \cup \{\texttt{relevant}(y)\}$.

We start with the latter task, and check to begin with that, for infons in the original vocabulary, no more can be deduced using $\dashv^c$ than can be deduced using $\dashv^e$.

Claim 6.19. *Let X, y, and Z be in the original vocabulary (without* `relevant`*). Suppose $y \dashv^c X \cup \{\texttt{relevant}(z) \mid z \in Z\}$. Then $y \dashv^e X$.*

Proof. By induction on the length of the careful deduction leading to y, dividing into cases depending on the rule used in the last step of the deduction.

The rule used in the last step cannot be Relevant, since it would not lead to an infon in the original vocabulary.

Suppose the rule used in the last step is wDel.3. Then y has the form *pref* $p\ \texttt{tdOn}\ q\ \texttt{tdOn}_d\ x$, and *pref* $p\ \texttt{tdOn}\ x$ and *pref* $q\ \texttt{exists}$ appear earlier in the deduction. By induction both are $\dashv^e X$. Using a SaidMon extension of Del it follows that so is y.

A similar argument works when the rule used in the last step is wSum.3.

Suppose the rule used in the last step is Sum.1 or Sum.2. Then y has the form *pref* u or *pref* v, where the infon *pref* $(u+v)$ appears earlier in the deduction. We need the following subclaim: u and v are in the original vocabulary.

In fact, by induction on the length of the careful deduction it is easy to prove the stronger subclaim, that the deduction cannot reach infons of any of the following forms:

(1) *pref* p $\texttt{tdOn}_d$ f where f is not in the original vocabulary.
(2) Infons with nested occurrences of `relevant`.
(3) Infons *pref* $(f+g)$ with $f+g$ not in the original vocabulary.

Item 1 is used in the proof of 2, which in turn is used in the proof of 3. The subclaim that u and v are in the original vocabulary follows from 3.

Returning to the main proof, knowing that u and v are both in the original vocabulary, and *pref* $(u+v)$ appears earlier in the deduction, we conclude by induction that *pref* $(u+v) \dashv^e X$. An application of a SaidMon extension of Sum.1 and Sum.2 now shows *pref* u and *pref* v are both $\dashv^e X$.

Suppose the rule used in the last step is any of Said0∞, SelfQuote, Trusted0∞, TrustApplication, Del-, Role.1–2, wDel.1, and wDel.2. Let $y \dashv y_1, \ldots, y_k$ be the instance of the rule that is used in the last step. Note that in all those rules, every non-regular variable appearing on the right-hand-side of $\dashv$ also appears on the left-hand-side. From this and the fact that y is in the original vocabulary (without `relevant`), it follows that each of the infons $y_1, \ldots, y_k$ is in the original vocabulary. These infons are deduced before y, and by induction therefore they are all $\dashv^e X$. So $y \dashv^e X$.

Suppose finally that the rule used in the last step is wExists. Then y has the form *pref* q `exists`, and there is a previous infon of the form *pref* x in the deduction, with q `regcomp` x, and $\texttt{original}(x)$. By induction *pref* $x \dashv^e X$, and using a SaidMon extension of Exists it follows from this that *pref* q `exists` $\dashv^e X$. □

We now work to show that if $y \dashv^e z$, then $y \dashv^c z + \texttt{relevant}(y)$.

Claim 6.20. *If X is a germ for y, then $y \dashv^c X \cup \{\texttt{relevant}(y)\}$.*

Proof. Let $x_1, \ldots, x_r = y$ be a deduction of y from X, using only SaidMon extensions Del, Trust0∞, and Sum.3. We work by induction on the length of the deduction, dividing into cases depending on which rule is used in the last step of the deduction.

Suppose that the last step in the deduction is a SaidMon extension of the rule Sum.3. Then y has the form *pref* $(u+v)$, and both *pref* u and *pref* v occur previously in the deduction. Using Relevant.3, `relevant`(*pref* u) and `relevant`(*pref* v) are $\dashv^{cr}$ `relevant`(y). By induction, *pref* $u \dashv^{cr} X \cup \{\texttt{relevant}(\textit{pref}\ u)\}$, and similarly with v. By Relevant.4, *pref* `relevant`$(u+v) \dashv^{cr}$ `relevant`(y). Combining all these deductions we see that *pref* u, *pref* v, and *pref* `relevant`$(u+v)$ are $\dashv^{cr} X \cup \{\texttt{relevant}(y)\}$. Now using the SaidMon extension of wSum.3 obtained by prepending *pref* to the rule we conclude that $y =$ *pref* $(u+v) \dashv^{cr} X \cup \{\texttt{relevant}(y)\}$.

A similar argument handles the case that the last step in the deduction involves Del or Trust0∞, using Relevant.7 and wDel.3 rather than Relevant.3 and wSum.3 in the case of Del, and using Relevant.5 and Trust0∞ in the case of Trust0∞. □

Remark 6.21. Suppose as in the last claim that X is a germ for y. Suppose further that y has the form p $\texttt{said}_d$ v. Then $y \dashv^{cr} X \cup \{p\ \texttt{said}_d\ \texttt{relevant}(v)\}$. To see this, note that

by Remark 6.8, X contains a set p $\mathtt{said}_d$ U where U is a germ for v. By the last claim $v \dashv^{cr} U \cup \{\mathtt{relevant}(v)\}$, and prepending p $\mathtt{said}_d$ to the deduction witnessing this it follows that $y \dashv^{cr} X \cup \{p \ \mathtt{said}_d \ \mathtt{relevant}(v)\}$.

Lemma 6.22. *Suppose that $y \dashv^{b} Z$. Then there is a germ Y for y so that $Y \dashv^{cr} Z$.*

Proof. We repeat the proof of Lemma 6.16 and the results leading to it (namely Claims 6.12 and 6.14), replacing $\dashv^{e}_{D}$ by $\dashv^{cr}$ throughout. The replacement can be done trivially in all cases where Del and Sum.3 are not used, as all other rules allowed in enhanced deductions are also allowed in careful deductions. (Exists is not allowed in careful deductions, but all its instances on infons in the original vocabulary are allowed, and only these instances show up in the proof of Lemma 6.16.) The uses of Del and Sum.3 are listed in Remark 6.15. We go over them now, checking that $\dashv^{e}$ can be replaced by $\dashv^{cr}$.

Consider first item (a) in Remark 6.15, which indicates a possible use of Del and Sum.3 in the equation p $\mathtt{said}_d$ $u \dashv^{e}_{D} \{p \ \mathtt{tdOn}_d \ u\} \cup U_2$, tagged (a) in the proof of Claim 6.14, where U_2 is a germ for p $\mathtt{said}_d$ u. We must check that p $\mathtt{said}_d$ $u \dashv^{cr} \{p \ \mathtt{tdOn}_d \ u\} \cup U_2$, so that $\dashv^{e}_{D}$ can be changed to $\dashv^{cr}$ in the equation. Now U_2 is a germ for p $\mathtt{said}_d$ u. The rules allowed in the definition of a germ do not increase quotation depth, and indeed do not introduce any new prefixes. So there must be an infon p $\mathtt{said}_d$ v in U with $\mathtt{quoteDepth}(v) \geq \mathtt{quoteDepth}(u)$. Applying Relevant.1 we therefore get p $\mathtt{said}_d$ $\mathtt{relevant}(u) \dashv^{cr} \{p \ \mathtt{tdOn}_d \ u\} \cup U_2$. U_2 is a germ for p $\mathtt{said}_d$ u, and so by Claim 6.20, or More precisely Remark 6.21, p $\mathtt{said}_d$ $u \dashv^{cr} U_2 \cup \{\mathtt{relevant}(p \ \mathtt{said}_d \ u)\}$. Combining these deductions we get p $\mathtt{said}_d$ $u \dashv^{cr} \{p \ \mathtt{tdOn}_d \ u\} \cup U_2$, as needed.

Consider next item (b) in Remark 6.15. We have to check that p $\mathtt{tdOn}$ $x \dashv^{cr} U_1 \cup \{p \ \mathtt{said} \ q_1 \ \mathtt{tdOn}_e \ x\}$, where U_1 is a germ for p $\mathtt{tdOn}$ x. By Relevant.2, $\mathtt{relevant}(p \ \mathtt{tdOn} \ x) \dashv^{cr} p \ \mathtt{said} \ q_1 \ \mathtt{tdOn}_e \ x$. By Claim 6.20, p $\mathtt{tdOn}$ $x \dashv^{cr} U_1 \cup \{\mathtt{relevant}(p \ \mathtt{tdOn} \ x)\}$. Combining the two deductions we get p $\mathtt{tdOn}$ $x \dashv^{cr} U_1 \cup \{p \ \mathtt{said} \ q_1 \ \mathtt{tdOn}_e \ x\}$, as required.

Item (c) in Remark 6.15 is a special case of item (b), with $e = \infty$.

Consider finally item (d). We have to check that p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ g^* $\dashv^{cr}$ $\{p \ \mathtt{tdOn} \ q_{l+1} \ \mathtt{tdOn}_{d_{l+1}} \ g^*\} \cup U_2$, where U_2 is a germ for p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ g^*.

Note first that U_2, being a germ for p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ g^*, must have an infon of the form p $\mathtt{said}_d$ v. Using Relevant.1 we therefore get that:

(i) p $\mathtt{said}_d$ $\mathtt{relevant}(q_{l+1} \ \mathtt{tdOn}_{d_{l+1}} \ g^*) \dashv^{cr} \{p \ \mathtt{tdOn} \ q_{l+1} \ \mathtt{tdOn}_{d_{l+1}} \ g^*\} \cup U_2$.

Since U_2 is a germ for p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ g^*, there must be an infon of the form p $\mathtt{said}_d$ v in U_2 with q_1 $\mathtt{regcomp}$ v. So p $\mathtt{said}_d$ q_1 $\mathtt{exists} \dashv^{cr} U_2$ by an application of a SaidMon extension of wExists. Extending the deduction leading to (i) with the SaidMon extension of Relevant.6 and (if $d_{l+1} = 0$) Relevant.5 obtained by prepending p $\mathtt{said}_d$, we get:

(ii) p $\mathtt{said}_d$ $\mathtt{relevant}(q_1 \ \mathtt{tdOn} \ g^*) \dashv^{cr} \{p \ \mathtt{tdOn} \ q_{l+1} \ \mathtt{tdOn}_{d_{l+1}} \ g^*\} \cup U_2$.

By Claim 6.20, or more precisely Remark 6.21, p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ $g^* \dashv^{cr} U_2 \cup \{p \ \mathtt{said}_d \ \mathtt{relevant}(q_1 \ \mathtt{tdOn} \ g^*)\}$. Combining this with (ii) we get p $\mathtt{said}_d$ q_1 $\mathtt{tdOn}$ $g^* \dashv^{cr} \{p \ \mathtt{tdOn} \ q_{l+1} \ \mathtt{tdOn}_{d_{l+1}} \ g^*\} \cup U_2$, as required. □

Corollary 6.23. *The following are equivalent for infons y and z in the original vocabulary (without* `relevant`*):*

(1) $y \dashv^{b} Z$.
(2) $y \dashv^{cr} Z + \mathtt{relevant}(y)$.

Proof. Suppose first that $y \dashv^b Z$. By Lemma 6.22, there is a germ Y for y so that $Y \dashv^{cr} Z$. Now using Lemma 6.20 it follows that $y \dashv^{cr} Z + \texttt{relevant}(y)$.

Conversely, suppose that $y \dashv^{cr} Z + \texttt{relevant}(y)$. Then by Claim 6.19, $y \dashv^e z$, and hence $y \dashv^b Z$. □

Recall that our aim in this subsection is to reach an algorithm which, given a set of infons Z and another set I of infons of interest, produces all infons in I which ensue Z. By Corollary 6.23 the algorithm need simply produce all infons which are $\dashv^{cr} Z \cup \{\texttt{relevant}(w) \mid w \in I\}$. This can be done by repeatedly going over the deduction rules, adding to a list all infons which follow from previously added infons using one of the rules. The process must be iterated until it ceases to add new infons. The time complexity of the algorithm is polynomial in length$(Z \cup I)$ multiplied by the number of infons which might be added (this number bounds the number of iterations which may be needed). We end this subsection by finding a bound on the number (and indeed the set) of infons which may be produced by a careful deduction from $Z \cup \{\texttt{relevant}(w) \mid w \in I\}$.

Let Z and I be given. Let B be the set of regular components of infons in Z. Let δ be the largest quotation depth of an infon in Z.

Claim 6.24. *If $y \dashv^b Z$ then all regular components of y belong to Z, and the quotation depth of y is at most δ.*

Proof. Immediate by induction on the deduction of y from Z; none of the deduction rules increases quotation depth, or adds new regular components. □

In light of the claim we may assume that all regular components of infons in I belong to B, and all infons in I have quotation depth $\leq \delta$; if I has infons which do not satisfy these conditions we might as well remove them, as they cannot be deduced from Z.

We say that x is a *component* of y if x labels a node in the semantic tree of y.

Let T_1 consist of all infons of the following forms:

(1) Components of infons in $Z \cup I$.
(2) p *attribute* where $p \in B$ and *attribute* is an attribute component of an infon in $Z \cup I$, or `exists`.
(3) p $\texttt{tdOn}_d$ x where $p \in B$ and either $\texttt{tdOn}_0$ x or `tdOn` x is an attribute component of an infon in $Z \cup I$.

Let T consist of all infons of the forms p_1 $\texttt{said}_{d_1}$ $\ldots$ p_l $\texttt{said}_{d_l}$ x and p_1 $\texttt{said}_{d_1}$ $\ldots$ p_l $\texttt{said}_{d_l}$ $\texttt{relevant}(x)$ where $x \in T_1$, $l + \texttt{quoteDepth}(\text{x}) \leq \delta$, $p_i \in B$, and $d_i \in \{0, 1\}$.

Claim 6.25. *T is closed under careful deductions.*

Proof. The proof is immediate by inspection of the rules allowed in careful deductions. Let us only comment that the restriction in Relevant.1 that there must exist some y so that p $\texttt{said}_d$ y and $\texttt{quoteDepth}(y) \geq \texttt{quoteDept}(x)$ is used to guarantee that SaidMon extensions of the rules, when applied to infons in T, only produce infons *pref* $\texttt{relevant}(x)$ with the depth of *pref* plus the quotation depth of x being at most δ. □

Claim 6.26. *The size of T is bounded by $3 \cdot (\delta + 1) \cdot N \cdot (2 \cdot b)^{\delta+1}$, where δ is the largest quotation depth in Z, N is the number of components of infons in $Z \cup I$, and b is the number of regular components of infons in Z.*

Proof. The size of T_1 is at most $N + 2 \cdot N \cdot b$, which is bounded by $3 \cdot N \cdot b$. (We may assume that Z has at least one regular component, for otherwise the only infons which ensue Z are those which belong to Z, and there is nothing to compute.) The size of T is at most $2 \cdot (\delta + 1) \cdot (2 \cdot b)^\delta$ times the size of T_1. □

Since $Z \cup \{\texttt{relevant}(w) \mid w \in I\}$ is contained in T by the definition of T, it follows from the last two claims that the number of infons which are $\dashv^{cr} Z \cup \{\texttt{relevant}(w) \mid w \in I\}$ is at most $3 \cdot (\delta+1) \cdot N \cdot (2 \cdot b)^{\delta+1}$. This induces a bound polynomial in $(\text{length}(Z \cup I)) \cdot (2 \cdot b)^{\delta+1}$ on the time complexity of the algorithm producing all these infons. By Corollary 6.23, this algorithm produces all infons in I which ensue Z, achieving our initial goal for this subsection. But we shall not use the algorithm directly, and instead fold it into the algorithm for answering basic queries, in the next subsection.

6.2. **The algorithm.** Fix a substrate X, an authorization policy $\mathcal{A}$, and a query $Q = (a\ \texttt{knows}_d\ t(v_1, \ldots, v_k))$. (Below we refer to d as $d(Q)$, and to k as $k(Q)$.) We describe how to compute the answer to Q under $\mathcal{A}$.

We intend to reduce the computation of the answer to a computation of a fixed point $\bar{\Pi}(\mathring{X})$ where $\bar{\Pi}$ is a logic program derived from $\mathcal{A}$ and the house rules, $\mathring{X}$ is a *finite* partial substructure of X. The reduction will use the results of the previous subsection, and the following safety conditions, which as we noted in the preamble to this section were imposed as part of the assertion form in Subsection 4.1:

- Variables range only over regular elements.
- Variables must be "knowledge bound" in the sense that if a variable v, other than the target of the assertion, occurs in the assertion, then the premise of the assertion must include knowledge of an infon which has v as a regular component.
- Non-ground regular expressions in the assertion head must also be knowledge bound.

We begin though by producing the program $\bar{\Pi}$. It will consist of the assertions in $\mathcal{A}$, the house rules K0∞ and Say2know, and enough additional rules to compute the consequences of KMon (and KSum) without having to compute any more of ensue than is necessary and possible within our intended time bounds. These additional rules will be the KMon consequences of the deduction rules of the previous subsection, and assertions on the relevance of infons occurring in the query Q or in bodies of assertions in $\mathcal{A}$.

Definition 6.27. The *companion* of an assertion $A :_d x \leftarrow x_1, \ldots, x_n, con$ or $A :_d x\ \texttt{to}\ p \leftarrow x_1, \ldots, x_n, con$ is the assertion

$$A :_d \texttt{relevant}(x_1 + \cdots + x_n).$$

The companion to the query $Q = (a\ \texttt{knows}_{d(Q)}\ t(v_1, \ldots, v_{k(Q)}))$ is the assertion

$$a :_{d(Q)} \texttt{relevant}(t).$$

Here, as in the previous subsection, `relevant` is a new synthetic function, not occurring in the original vocabulary of the policy $\mathcal{A}$ and the query Q.

Definition 6.28. By a *knowledge consequence* of a deduction rule

$$R = (y \dashv x_1, \ldots, x_n \text{ if } body)$$

we mean the rule

$$p\ \texttt{knows}_d\ y \leftarrow p\ \texttt{knows}_d\ x_1, \ldots, p\ \texttt{knows}_d\ x_n,\ body.$$

(The part *body* is empty except in the rules Relevant.1, wExists, and Exists, and in these rules it is a substrate constraint.) The SaidMon extensions of the knowledge consequence of R are the rules of the form

$$p\ \texttt{knows}_d\ \mathit{pref}\ y \leftarrow p\ \texttt{knows}_d\ \mathit{pref}\ x_1, \ldots, p\ \texttt{knows}_d\ \mathit{pref}\ x_n,\ \mathit{body}$$

where *pref* has the form $p_1\ \texttt{said}_{d_1} \ldots p_l\ \texttt{said}_{d_l}$. The *depth* of the extension is the maximum depth of the infons $\mathit{pref}\ x_1, \ldots, \mathit{pref}\ x_n$.

To give a quick example,

$$p\ \texttt{knows}_d\ q\ \texttt{said}_0\ r\ \texttt{exists} \leftarrow p\ \texttt{knows}_d\ q\ \texttt{said}_0\ x,\ r\ \texttt{regcomp}(x),\ \texttt{original}(x)$$

is a SaidMon extension of a knowledge consequence of the rule wExists of the previous subsection.

Let δ be the maximum quotation depth of an assertion in $\mathcal{A}$. Let $\bar{\Pi}$ be the program consisting of:

(1) All the assertions in $\mathcal{A}$.
(2) The companions of the assertions in $\mathcal{A}$ and of the query Q.
(3) The house rules K0∞ and Say2know.
(4) The SaidMon extensions, of depth $\leq \delta$, of the knowledge consequences of the rules allowed in careful deductions, namely the rules Sum.1 and Sum.2, Said0∞, SelfQuote, Trusted0∞, TrustApplication, Del$^-$, Role.1–2, wDel.1–3, wSum.3, wExists, and Relevant.

Algorithm. It will follow from the proofs below that the answer to Q under $\mathcal{A}$ can be computed by running the program $\bar{\Pi}$ on the substrate generated by synthetic functions from the regular element interpreting ground regular expressions in $\mathcal{A}$ and in Q, and this can be taken as the algorithm. The substrate has a finite regular layer, as $\mathcal{A}$ is finite, but an infinite synthetic layer. The program $\bar{\Pi}$ terminates in finite time, and indeed polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$, because of the restriction in clause 4 of its definition to rules allowed in *careful* deductions. Still the program correctly computes the answer to Q. These facts follow from the proofs below. The proofs use the results of the previous subsection, relating careful deductions to the ensue order.

The actual algorithm we write below is slightly different, in that we restrict the substrate further, to be finite, so as to derive the polynomial time bound directly from Theorem A.3.

Claim 6.29. (1) *The length of* $\bar{\Pi}$ *is linear in* $\text{length}(\mathcal{A}) + \text{length}(Q) + \delta \cdot 2^{\delta+1}$.
(2) *The width of* $\bar{\Pi}$ *is at most* $\max\{\text{width}(\mathcal{A}), \delta + 3\}$.

Proof. The fourth clause in the definition of $\bar{\Pi}$ contributes a number of rules proportional to $2^{\delta+1}$, as there are $2^{\delta+1}$ prefix expressions of depth $\leq \delta$ (determined by the alternation pattern of $\texttt{said}$ and $\texttt{said}_0$). The lengths of the rules contributed by this clause is proportional to δ, as the length of the prefix is proportional to δ and the rest is fixed. The third clause contributes two rules of fixed length, and the contribution of the first two clauses is proportional to $\text{length}(\mathcal{A}) + \text{length}(Q)$. This proves condition 1 in the claim.

As for condition 2, $\text{width}(\mathcal{A})$ bounds the width of the rules in the first two clauses in the definition of $\bar{\Pi}$, 3 bounds the width of the rules in the third clause, and $\delta + 3$ bounds the width in the fourth clause. □

Let B consists of the values of the following regular expressions:

(1) All principals who own assertions in $\mathcal{A}$, and the principal a whose knowledge is to be queried.
(2) All ground regular expressions appearing in heads of assertions in $\mathcal{A}$.

Let T_0 consist of all expressions (including subexpressions) which appear in $\mathcal{A}$ or in Q, and all expressions of the form p $\texttt{said}_d$ x where x is the head of an assertion of form (As2) in $\mathcal{A}$. Let T_1 consist of all expressions in T_0 plus expressions of the following forms:

(1) p *attribute* where p is a principal variable and *attribute* is an attribute expression in T_0, or `exists`.
(2) p $\texttt{tdOn}_d$ x where p is a principal variable and either $\texttt{tdOn}_0$ x or `tdOn` x is an attribute expression in T_0.

Let T consist of all expressions of the forms p_1 $\texttt{said}_{d_1}$ $\ldots p_l$ $\texttt{said}_{d_l}$ x and p_1 $\texttt{said}_{d_1}$ $\ldots p_l$ $\texttt{said}_{d_l}$ $\texttt{relevant}(x)$ where $x \in T_1$, $l + \texttt{quoteDepth}(\text{x}) \leq \delta$, p_i are Principal variable, and $d_i \in \{0, 1\}$.

Let $\mathring{X}$ be the restriction of the substrate X to the universe consisting of the values of all expressions in T under assignments of values from B to their variables. The computation leading to Claim 6.26 shows that the size of $\mathring{X}$ is bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1}$. (The number of expressions which appear in $\mathcal{A}$ and in Q is bounded by $\text{length}(\mathcal{A}) + \text{length}(Q)$. The size of $|B|$ too is bounded by $\text{length}(\mathcal{A}) + \text{length}(Q)$.)

We intend to prove:

Lemma 6.30. *The formula "a $\texttt{knows}_{d(Q)}$ $t(b_1, \ldots, b_{k(Q)}) \wedge a$ $\texttt{knows}_{d(Q)}$ b_1 `exists` $\wedge \ldots \wedge a$ $\texttt{knows}_{d(Q)}$ $b_{k(Q)}$ `exists`" holds in $\Pi(X)$ iff it holds in $\bar{\Pi}(\mathring{X})$. Further, if $\bar{\Pi}(\mathring{X}) \models a$ $\texttt{knows}_{d(Q)}$ b `exists`, then $b \in B$.*

Granted the lemma, we can compute the answer to Q under $\mathcal{A}$ by following these steps:

(1) Determine δ from $\mathcal{A}$ and Q.
(2) Determine the restricted substrate $\mathring{X}$, and the program $\bar{\Pi}$.
(3) Compute the fixed point $\bar{\Pi}(\mathring{X})$.
(4) Produce the set of tuples $(b_1, \ldots, b_k)$, with elements taken from B, so that the formula "a $\texttt{knows}_{d(Q)}$ $t(b_1, \ldots, b_{k(Q)}) \wedge a$ $\texttt{knows}_{d(Q)}$ b_1 `exists` $\wedge \ldots \wedge a$ $\texttt{knows}_{d(Q)}$ $b_{k(Q)}$ `exists`" holds in $\bar{\Pi}(\mathring{X})$.

The time required to perform task 1 is linear in $\text{length}(\mathcal{A}) + \text{length}(Q)$. The time required for task 2 is polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1}$. (The time required to produce T_1 is linear in $\text{length}(\mathcal{A}) + \text{length}(Q)$, and the size of B is bounded by $\text{length}(\mathcal{A}) + \text{length}(Q)$, but the time required to produce the prefixes in forming T and instantiate them in forming $\mathring{X}$ introduces the factor $(2 \cdot |B|)^{\delta+1}$. The time required to produce $\bar{\Pi}$ is proportional to its length, which by Claim 6.29 is certainly bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1}$.) By Theorem A.3, the time required for task 3 is bounded by a polynomial in $|\mathring{X}| \cdot \text{length}(\bar{\Pi}) \cdot |B|^{\text{width}(\bar{\Pi})}$, which in turn is bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$. Finally task 4 involves going over $|B|^{k(Q)}$ tuples, and as w bounds the width $k(Q)$ of the query, the time required for this task too is bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$. The following theorem is thus a corollary of Lemma 6.30:

Theorem 6.31. *There is an algorithm which given a policy $\mathcal{A}$and a query Q computes the answer to Q under A in time complexity bounded by $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$, where δ*

is the maximal quotation depth of assertions in $\mathcal{A}$, and w is the maximal width of assertions in $\mathcal{A}$.

The rest of this subsection is dedicated to the proof of Lemma 6.30.

Claim 6.32. *Let $r \in B$.*

- *If $\bar{\Pi}(\mathring{X}) \models r$ $\texttt{knows}_d$ z and b $\texttt{regcomp}$ z, then $b \in B$.*
- *If $\bar{\Pi}(\mathring{X}) \models q$ $\texttt{says}_d$ z $\texttt{to}$ r, then $q \in B$, and if b $\texttt{regcomp}$ z then $b \in B$.*

Proof. The two conditions of the claim would follow by simultaneous induction on the stage in the iteration used to compute the least fixed point for $\bar{\Pi}$, provided we can prove that:

(1) If R is an instance of a rule in $\bar{\Pi}$ with conclusion r $\texttt{knows}_d$ z, and b $\texttt{regcomp}$ z, then:
   (a) $b \in B$, or
   (b) there is a clause r $\texttt{knows}_{d'}$ z' in the premise of R so that b $\texttt{regcomp}$ z', or
   (c) there is a clause q $\texttt{says}_{d'}$ z' $\texttt{to}$ r in the premise of R so that $b = q$ or b $\texttt{regcomp}$ z'.

(2) If R is an instance of a rule in $\bar{\Pi}$ with conclusion q $\texttt{says}_d$ z $\texttt{to}$ r, then $q \in B$, and for every regular component b of z we have:
   (a) either $b \in B$,
   (b) or else there is a clause r $\texttt{knows}_{d'}$ z' in the premise of R so that b $\texttt{regcomp}$ z'.

Inspection of the rules in items 3 and 4 in the definition of $\bar{\Pi}$ immediately establishes conditions 1 and 2 for their instances. (The third possibility in condition 1 appears because of the inclusion of rule Say2know.) It remains to check the conditions for instances of assertions in $\mathcal{A}$, and their companions.

We deal first with instances of assertions of the form (As1). Note that this includes also the companions of all assertions in Π. Suppose that

$$A :_d \ x \ \leftarrow \ x_1, \ldots, x_n, con$$

is an assertion of this form. Recall from Subsection 4.1 that the full assertion is

$$\begin{aligned} A \ \texttt{knows}_d \ x \ \leftarrow A \ \texttt{knows}_d \ x_1 \wedge \cdots \wedge A \ \texttt{knows}_d \ x_n \wedge con \wedge \\ A \ \texttt{knows}_d \ \tau_1 \ \texttt{exists} \ \ \wedge \cdots \wedge A \ \texttt{knows}_d \ \tau_k \ \texttt{exists} \end{aligned}$$

where $\tau_1, \ldots, \tau_k$ include (among other things) all non-ground regular components of x. The values of the ground components of x are by definition elements of B. Thus, if an instance of the assertion leads to a conclusion A $\texttt{knows}_d$ z, and b $\texttt{regcomp}$ z, then either $b \in B$ or else there is a clause A $\texttt{knows}_d$ b $\texttt{exists}$ in the premise of the instance, as required for 1.

Suppose next that

$$A :_d \ x \ \texttt{to} \ p \ \leftarrow \ x_1, \ldots, x_n, con$$

is an assertion of the form (As2) in $\mathcal{A}$. Recall from Subsection 4.1 that the full assertion is

$$\begin{aligned} A \ \texttt{says}_d \ x \ \texttt{to} \ p \ \leftarrow A \ \texttt{knows}_d \ x_1 \wedge \cdots \wedge A \ \texttt{knows}_d \ x_n \wedge con \wedge \\ A \ \texttt{knows}_d \ \tau_1 \ \texttt{exists} \ \ \wedge \cdots \wedge A \ \texttt{knows}_d \ \tau_k \ \texttt{exists} \end{aligned}$$

where $\tau_1, \ldots, \tau_k$ include all non-ground regular components of x, other than p. p is instantiated to r which is assumed to be an element of B. The value of A belongs to B by definition, and so do the values of the ground regular components of x. Condition 2 follows. □

Claim 6.33. *For infon f in the original vocabulary (without* `relevant`*):*

(1) *If $\bar{\Pi}(\mathring{X}) \models p$ $\texttt{says}_d$ f $\texttt{to}$ q, then $\Pi(X) \models p$ $\texttt{says}_d$ f $\texttt{to}$ q.*

(2) *If* $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ f*, then* $\Pi(X) \models p$ `knows`$_d$ f.

Proof. Let $\Gamma_{\bar{\Pi}}$ be the immediate action operator of the program $\bar{\Pi}$. Let $\bar{\Pi}(\Pi(X))$ be the least fixed point of $\Gamma_{\bar{\Pi}}$ above $\Pi(X)$, namely the limit of the structures X_n where $X_0 = \Pi(X)$ and $X_{n+1} = \Gamma_{\bar{\Pi}}(X_n)$.

By Claim 6.19, repeated applications of $\Gamma_{\bar{\Pi}}$ to $\Pi(X)$ do not add to `knows`$_d$ any instances p `knows`$_d$ y with y in the original vocabulary. It follows that $\Pi(X)$ and in $\bar{\Pi}(\Pi(X))$ agree on `says`$_d$ and on the restriction of `knows`$_d$ to infons in the original vocabulary.

Certainly if $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ f, then $\bar{\Pi}(\Pi(X)) \models p$ `knows`$_d$ f, and by the above this implies that $\Pi(X) \models p$ `knows`$_d$ f for infon f in the original vocabulary. This establishes condition 2 of the claim, and a similar implication chain establishes condition 1. □

Let `knows`$^*_d$ be defined by the following clauses. Infons here are in the original vocabulary.

(1) If $p \in B$ then p `knows`$^*_d$ f just in case that there is a set Y of infons so that:
- $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ y for each $y \in Y$, and
- Y is a germ for f.

(2) If $p \notin B$ then p `knows`$^*_d$ f for all infons f.

Let `saysto`$^*_d$ be defined by the following clauses. Again infons are in the original vocabulary.

(1) If $p \in B$ then q `says`$^*_d$ f `to` p just in case that $\bar{\Pi}(\mathring{X}) \models q$ `says`$_d$ f `to` p.
(2) If $p \notin B$ then q `says`$^*_d$ f `to` p for all f and all q.

Claim 6.34. (1) *If* p `knows`$^*_d$ f*, and* p `knows`$^*_d$ g*, then* p `knows`$^*_d$ $f + g$.
(2) *If* p `knows`$^*_d$ f *and* $g \leq f$ *then* p `knows`$^*_d$ g.
(3) *If* p `knows`$^*_0$ f *then* p `knows`* f.
(4) *If* $p \in B$ *and* p `knows`$^*_d$ q `exists` *then* $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ q `exists`.

Proof. All conditions are obvious for $p \notin B$. So suppose $p \in B$. Condition 1 in this case follows from the fact that if F is a germ for f, and G a germ for g, then $F \cup G$ is a germ for $f+g$. Condition 3 follows from the fact that $\bar{\Pi}(\mathring{X})$ satisfies the house rule K0∞. Condition 4 holds because any germ for the infon q `exists` must actually include the infon. Let us prove condition 2. The proof makes heavy use of the results of the previous subsection, connecting $\leq$ with $\dashv^{cr}$.

We use p `knows`$_d$ Y to abbreviate the conjunction of p `knows`$_d$ y for $y \in Y$.

Suppose Y is a germ for f, and $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ Y. Since $g \leq f$, and since Y is a germ for f, $g \leq Y$. It follows by Lemma 6.22 that there is a germ Z for g so that $Z \dashv^{cr} Y$. By Claim 6.25 and the definition of $\mathring{X}$, all steps of the careful deduction leading from Y to Z involve infons in $\mathring{X}$. From this, the fact that $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ Y, and the inclusion of the knowledge consequences of the careful deduction rules in $\bar{\Pi}$, it follows that $\bar{\Pi}(\mathring{X}) \models p$ `knows`$_d$ Z. Since Z is a germ for g, p `knows`$^*_d$ g. □

Let X^* be the superstrate structure defined by the following clauses:

(1) The relation `knows`$_d$ of X^* is the relation `knows`$^*_d$.
(2) The relation `saysto`$_d$ of X^* is the relation `saysto`$^*_d$.
(3) The relation $\leq$ of X^* is the relation $\leq$ of $\Pi(X)$.

Claim 6.35. *Let* $x(v_1, \ldots, v_k)$ *be a expression for an infon in the original vocabulary. Let* ξ *be an assignment of values* $b_1, \ldots, b_k$ *to the variables of* x*. Suppose that the assertion* $A :_d$

$\texttt{relevant}(x(v_1,\dots,v_k))$ *is included in* $\bar{\Pi}$. *Let* c *be the value of* A, *and let* $f = x(b_1,\dots,b_k)$ *be the value of* x *under* ξ.

Suppose that $c\ \texttt{knows}_d\ f \wedge c\ \texttt{knows}_d\ b_1\ \texttt{exists} \wedge \cdots \wedge c\ \texttt{knows}_d\ b_k\ \texttt{exists}$ *holds in* X^*. *Then it holds also in* $\bar{\Pi}(\mathring{X})$.

Proof. Suppose $c\ \texttt{knows}^*_d\ f$. By definition of $\texttt{knows}^*_d$ this means that there is a germ Y for f so that:

(i) $\bar{\Pi}(\mathring{X}) \models c\ \texttt{knows}_d\ Y$.

If $\tau(v_1,\dots,v_k)$ is a regular component of $x(v_1,\dots,v_k)$ then $\tau(b_1,\dots,b_k)$ is a regular component of f, and so it must be a regular component of an infon in Y. It therefore follows from (i) that:

(ii) $\bar{\Pi}(\mathring{X}) \models c\ \texttt{knows}_d\ \tau(b_1,\dots,b_k)\ \texttt{exists}$, for each regular component τ of x.

A similar argument, but starting from $c\ \texttt{knows}^*_d\ b_j\ \texttt{exists}$ instead of $c\ \texttt{knows}^*_d\ f$, shows that:

(iii) $\bar{\Pi}(\mathring{X}) \models c\ \texttt{knows}_d\ b_j\ \texttt{exists}$, for each $j \le k$.

By assumption the assertion $A :_d \texttt{relevant}(x(v_1,\dots,v_k))$ is included in $\bar{\Pi}$. The body of the assertion has only clauses requiring the existence of $v_1,\dots,v_k$ and the existence of all regular components of $x(v_1,\dots,v_k)$. By conditions (ii) and (iii), the instance of the body given by the assignment ξ is true in $\bar{\Pi}(\mathring{X})$, and it follows that:

(iv) $\bar{\Pi}(\mathring{X}) \models c\ \texttt{knows}_d\ \texttt{relevant}(x(b_1,\dots,b_k))$.

By Corollary 6.23, $f = x(b_1,\dots,b_k) \dashv^{cr} Y \cup \{\texttt{relevant}(f)\}$. By Claim 6.25 and the definition of $\mathring{X}$, all steps of the careful deduction leading from $Y \cup \{\texttt{relevant}(f)\}$ to f involve infons in $\mathring{X}$. From this, conditions (i) and (iv), and the inclusion of the knowledge consequences of the careful deduction rules in $\bar{\Pi}$, it follows that

(v) $\bar{\Pi}(\mathring{X}) \models c\ \texttt{knows}_d\ f$.

This and condition (iii) complete the proof of the claim. □

Lemma 6.36. X^* *is a fixed point for* Π.

Proof. The relation $\le$ of X^* is by definition a fixed point for the ensue rules in Π. By Claim 6.34, X^* is a fixed point for K0∞, KSum and KMon. X^* is a fixed point for instances of Say2Know with $p \notin B$, simply because $p\ \texttt{knows}^*_d\ f$ for all infons f in this case. X^* is a fixed point for instances of Say2Know with $p \in B$ by the following chain of implications:

$$\begin{aligned} q\ \texttt{says}^*_d\ f\ \texttt{to}\ p \ &\Rightarrow\ \bar{\Pi}(\mathring{X}) \models q\ \texttt{says}_d\ f\ \texttt{to}\ p \\ &\Rightarrow\ \bar{\Pi}(\mathring{X}) \models p\ \texttt{knows}\ q\ \texttt{said}_d\ f \\ &\Rightarrow\ p\ \texttt{knows}^*\ q\ \texttt{said}_d\ f. \end{aligned}$$

The first implication holds by the definition of $\texttt{saysto}^*_d$ in the case $p \in B$, the second by rule Say2Know in $\bar{\Pi}(\mathring{X})$, and the third by the definition of $\texttt{knows}^*_d$.

It remains to show that X^* is a fixed point for each of the assertions in $\mathcal{A}$. We handle assertions of the form (As2). The argument for assertions of the form (As1) is similar and slightly easier.

Fix an assertion

(a) $$A :_d\ x\ \texttt{to}\ p \leftarrow x_1,\dots,x_n, con$$

in $\mathcal{A}$, and an assignment ξ of values to the variables of the assertion, so that the body of the instance of the assertion generated by making the assignment ξ is true in X^*. Let

$$c :_d \ f \ \texttt{to} \ b \ \leftarrow \ f_1, \ldots, f_n, con \tag{b}$$

be the instance of (a) obtained by making the assignment ξ. We have to show that $c \ \texttt{says}^*_d \ f \ \texttt{to} \ b$.

If $b \notin B$ then by definition $c \ \texttt{says}^*_d \ g \ \texttt{to} \ b$ for all infons g, and there is nothing further to show. So we may assume that $b \in B$.

Recall from Subsection 4.1 that the full form of (a) is

$$\begin{aligned} A \ \texttt{says}_d \ x \ \texttt{to} \ p \ \leftarrow \ & A \ \texttt{knows}_d \ x_1 \wedge \cdots \wedge A \ \texttt{knows}_d \ x_n \wedge con \wedge \\ & A \ \texttt{knows}_d \ \tau_1 \ \texttt{exists} \ \ \wedge \cdots \wedge A \ \texttt{knows}_d \ \tau_k \ \texttt{exists} \end{aligned} \tag{c}$$

where $p, \tau_1, \ldots, \tau_k$ include all the variables of the assertion. ξ must assign value b to p, so as to obtain the instance (b) above. Let b_j be the values of τ_j under ξ.

Since the instance of (c) obtained by making the assignment ξ is true in X^*, $X^* \models c \ \texttt{knows}_d \ b_j \ \texttt{exists}$ for each j. By condition 4 of Claim 6.34 it follows that $\bar{\Pi}(\mathring{X}) \models c \ \texttt{knows}_d \ b_j \ \texttt{exists}$, and by Claim 6.32 this implies that $b_j \in B$.

We have now that all values assigned by ξ to the variables of the assertion (a) are in B, and hence in $\mathring{X}$. The instance (c) therefore holds in $\bar{\Pi}(\mathring{X})$. The body of the instance is true in X^*. Using Claim 6.35, and the inclusion of the companion assertion to (a) in $\bar{\Pi}$, it follows that the body is true in $\bar{\Pi}(\mathring{X})$. So the head must be true in $\bar{\Pi}(\mathring{X})$, meaning that $\bar{\Pi}(\mathring{X}) \models c \ \texttt{said}_d \ f \ \texttt{to} \ b$. By the definition of $\texttt{saysto}^*_d$ it follows that $c \ \texttt{says}^*_d \ f \ \texttt{to} \ b$. □

Remark 6.37. The "A-bound" restrictions in the assertion forms in Subsection 4.1 are used several times in the proofs above. The restriction that all non-ground regular components of x must be A-bound is used in the proof of Claim 6.32, to see that the rules of $\bar{\Pi}$ do not introduce any regular elements not explicitly named in the policy. The restriction that all variables (other than p, in the case of form (As2)) must be A-bound is used in the proof of Lemma 6.36 to see that the assignment ξ uses only regular elements from B.

We are now ready to complete the proof of Lemma 6.30, and with it the proof of Theorem 6.31. Recall that we have to show that the formula "$a \ \texttt{knows}_d \ t(b_1, \ldots, b_k) \wedge a \ \texttt{knows}_d \ b_1 \ \texttt{exists} \wedge \ldots \wedge a \ \texttt{knows}_d \ b_k \ \texttt{exists}$" holds in $\Pi(X)$ iff it holds in $\bar{\Pi}(\mathring{X})$. (The extra clause of the lemma, that $b_j \in B$, follows from $a \ \texttt{knows}_d \ b_j \ \texttt{exists}$ by Claim 6.32.) a and t here are taken from the query Q, $d = d(Q)$, and $k = k(Q)$.

Proof of Lemma 6.30. The right-to-left direction is immediate from Claim 6.33. Let us prove the left-to-right direction.

Suppose "$a \ \texttt{knows}_d \ t(b_1, \ldots, b_k) \wedge a \ \texttt{knows}_d \ b_1 \ \texttt{exists} \wedge \ldots \wedge a \ \texttt{knows}_d \ b_k \ \texttt{exists}$" holds in $\Pi(X)$. By Lemma 6.36, X^* is a fixed point for Π. $\Pi(X)$ is the least fixed point. So all instances of $\texttt{knows}_d$ true in $\Pi(X)$ are true in X^*. Thus, the formula is true in X^*. The assertion $a :_d \texttt{relevant}(t(v_1, \ldots, v_k))$ is a companion to the query Q and was included in the program $\bar{\Pi}$. By Claim 6.35 it follows that "$a \ \texttt{knows}_d \ t(b_1, \ldots, b_k) \wedge a \ \texttt{knows}_d \ b_1 \ \texttt{exists} \wedge \ldots \wedge a \ \texttt{knows}_d \ b_k \ \texttt{exists}$" holds in $\bar{\Pi}(\mathring{X})$. □

With Lemma 6.30 at hand we know that the answer to Q is the same in $\Pi(X)$ and in $\bar{\Pi}(\mathring{X})$. The latter can be computed in time polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+1+w}$. This completes the proof of Theorem 6.31.

7. SecPAL-to-DKAL Translation

We describe a natural translation τ of SecPAL into DKAL. To this end, we assume that the reader is familiar with SecPAL though we recall some of the SecPAL definitions. We presume, without loss of generality that the sort, constant, function and relation names introduced explicitly in Section 3 do not occur in SecPAL.

Let p `says` x abbreviate p `says` x `to` *all*, where *all* is a fresh variable, distinct from p and not occurring in x. Let Open DKAL be the special version of DKAL obtained by augmenting DKAL with double rules

(1) p $\texttt{says}_d$ $x \leftarrow p$ $\texttt{knows}_d$ x
(2) p $\texttt{knows}_d$ $x \leftarrow p$ $\texttt{says}_d$ x

We will translate SecPAL into Open DKAL. The double rule 1 reflects the all-knowledge-is-common nature of SecPAL. The double rule 2 adds a mere convenience. Without it the `says` of SecPAL assertions would be translated into the `knows` of DKAL assertions; the double rule 2 allows us to translate `says` to `says`. In the rest of this section, by default, DKAL means Open DKAL.

Remark 7.1. It is possible to translate SecPAL to the original DKAL rather than Open DKAL. We mentioned already that double rule 2 is not essential for translation. The necessary instances of double rule 1 can be incorporated into the translation of SecPAL assertions; see Remark 7.4 in this connection. By translating SecPAL to the original DKAL we gain access to the complexity results in Section 6, specifically Theorem 4.7.

7.0.1. **Substrate.** The SecPAL document [3] speaks about constraint domains. What is a constraint domain? In our understanding a constraint domain can be faithfully viewed as a many-sorted first-order structure over which their constraints are evaluated. We consider a fixed constraint domain and we call the corresponding structure CD which is an allusion to "constraint domain". In accordance to the SecPAL restriction of constraint domains, there is a polynomial time algorithm for evaluating ground quantifier-free formulas over CD. Without loss of generality, we assume that the domain of any SecPAL variable is a sort of CD.

Remark 7.2. As far as the basic constraint domain of [3] is concerned, it is straightforward to view it as a many-sorted first-order structure, except for the relation e `matches` *pattern*. Suppose that e is a constant. What exactly matches the pattern? The name of the constant or its values? According to first-order logic, it should be the value of e, but, in [3], it is the name of e. One way to deal with this problem is to declare that the values of constants *are* their names. A more flexible approach is to introduce an additional function s that assigns strings to constants and to change e `matches` *pattern* to $s(e)$ `matches` *pattern*.

We define a DKAL substrate, called Sub for brevity, appropriate for the given domain structure CD. Sub is an extension of CD in the following sense.

- The vocabulary of CD is a part of the vocabulary of Sub, so that $\tau F = F$ for every member of the CD vocabulary. In particular, all CD sorts are Sub, and all CD function and relation symbols (including constants) preserve their types.
- The regular elements of Sub are precisely the elements of CD.
- All CD relations and functions (including nullary) have the same interpretation in both structures.

7.0.2. **Variables and Principal Constants.** If e is a SecPAL variable of a CD sort then τe, syntactically equal to e, is a variable of the same sort. The same applies to function and relation symbols of CD (including constants).

7.0.3. **Constraints.** If *con* is a SecPAL constraint then $\tau(con) = con$.

7.0.4. **Predicates.** If P is a SecPAL predicate of arity j, then τP, syntactically equal to P, is a j-ary synthetic function with Attribute values in Sub. The domain type of τP in Sub is the domain value of P in SecPAL. If $c_1, \ldots, c_j$ are SecPAL constants such that $(c_1, \ldots, c_j) \in \mathrm{Dom}(P)$ then $\tau P(\tau c_1, \ldots, \tau c_j)$, syntactically equal to $P(c_1, \ldots, c_j)$, is an attribute in Sub. In particular, if $j = 0$ then τP is an Attribute constant. For example, a nullary predicate `is a friend` becomes an Attribute constant.

7.0.5. **Verbphrases and facts.** SecPAL verbphrases and facts are defined by simultaneous recursion. We recall the definition and give the translation. Ground SecPAL verbphrases become attributes in our model, and ground SecPAL facts become infons in our model.

- In SecPAL, if P is a predicate and $e_1, \ldots, e_k$ are SecPAL expressions (variables or constants) of appropriate sorts then $P\ e_1, \ldots, e_k$ is a verbphrase. Accordingly $\tau(P\ e_1, \ldots, e_k) = (\tau P)(\tau e_1, \ldots, \tau e_k)$.
- In SecPAL, if e is a principal expression and V is a verbphrase then $e\ V$ is an fact. Accordingly
$$\tau(e\ V) = \mathcal{I}(\tau e, \tau V) = (\tau e)\ (\tau V).$$
- In SecPAL, if f is a fact then $\texttt{can say}_0\ f$ and $\texttt{can say}_\infty\ f$ are verbphrases. Accordingly $\tau(\texttt{can say}_0\ f) = \ \texttt{tdOn}_0\ \tau(f)$, and $\tau(\texttt{can say}_\infty\ f) = \ \texttt{tdOn}\ \tau(f)$.
- In SecPAL, if e is a principal expression then `can act as` e is a verbphrase. Accordingly $\tau(\texttt{can act as}\ e) = \ \texttt{canActAs}\ \tau e$.

It is easy to check by induction that, if we identify SecPAL's `can say` with DKAL's `tdOn`, and ignore the difference between `can act as` and `canActAs`, we have the following. For every SecPAL verbphrase V and every SecPAL fact f, the translation τV is syntactically equal to V and the translation τf is syntactically equal to f. In light of this we sometimes write just f and V instead of τf and τV below.

Lemma 7.3. *For every SecPAL fact f and substitution θ, we have $\tau(\theta(f)) = \theta(\tau(f))$.*

The proof is obvious.

7.0.6. **Assertions.** A SecPAL assertion has the form $A\ \texttt{says}\ f\ \texttt{if}\ f_1, \ldots, f_n, con$, where A is a constant, $n \geq 0$, f is a fact, every f_i is a fact, and *con* is a constraint. We define $\tau(A\ \texttt{says}\ f\ \texttt{if}\ f_1, \ldots, f_n, con)$ to be the following DKAL double assertion:

$$A_d : \ f\ \texttt{to}\ all\ \leftarrow\ f_1, \ldots, f_n, con.$$

Remark 7.4. This simple translation takes advantage of the new house rules introduced in the beginning of this section. If one prefers to translate SecPAL into DKAL without any additional house rules, one has to work a bit harder. In SecPAL, "a fact is *flat* when it does not contain `can say`," and every fact f has the form $e_1\ \texttt{can say}_{d_1}\ \ldots\ e_n\ \texttt{can say}_{d_n}\ g$ where $n \geq 0$ and g is flat. We refer to g as the *flat seed* of f. We refer to each of

the facts e_{k+1} `can say`$_{d_{k+1}}$ $\dots$ e_n `can say`$_{d_n}$ g, $0 \le k \le n$, as a *subfacts* of f. Define $\tau(A$ `says` f `if` $f_1, \dots, f_n, con)$ to be the set of the following DKAL assertions:

$$A_d : f \leftarrow f_1, \dots, f_n, con$$
$$A_d : f' \texttt{ to } all \leftarrow f'$$

where f' ranges over the subfacts of f. Notice that A broadcasts not only knowledge of f, but also knowledge of the proper subfacts f' of f. The following example shows that this is necessary: In SecPAL , assertions

$$A \texttt{ says } B \texttt{ can say foo} \leftarrow$$
$$B \texttt{ says foo} \leftarrow$$

imply A `says foo`. The translation of these assertions to DKAL leads to assertions which imply A `knows foo`. But if f' were to range only over $\{f\}$ in our translation of SecPAL assertions, that knowledge would not be shared with other principals. □

7.0.7. **Assertion context.** In SecPAL, an *assertion context AC* is a set $\{\alpha_1, \dots, \alpha_n\}$ of assertions. Accordingly τ *AC* is the union of the sets $\tau\alpha_i$.

SecPAL semantics is given by three deduction rules [3, §2]. It is common in logic, to use symbol $\vdash$ for derivability and symbol $\models$ for satisfaction in a structure. In [3], symbol $\models$ is used for both purposes. Here we stick to the standard usage. Accordingly some occurrences of $\models$ in [3] will be replaced by $\vdash$ in our exposition.

Theorem 7.5 (Embedding Theorem)**.** *Let AC be a safe SecPAL assertion context, and let* Π *be the open DKAL program consisting of the house rules and the assertions* $\bigcup_{\alpha \in AC} \tau\alpha$*. Further, let A be a SecPAL principal constant, f be a SecPAL ground flat fact expression, and d range over* $\{0, \infty\}$*. If*

$$AC, d \vdash A \texttt{ says } f$$

in SecPAL then

$$\Pi(\mathrm{Sub}) \models A \texttt{ says}_d \ f$$

in open DKAL

Proof. Induction on the length ℓ of the given SecPAL deduction, proving the implication not only for flat facts f, but also for nested facts provided they include only constants which appear in AC. The inductive steps are obvious, and we make only the following two comments.

First, the proof takes advantage of the additional house rules that essentially equate `knows` and `says`.

Second, the SecPAL safety condition guarantees that the only variable assignments relevant to computing flat consequences of AC are those with values *explicitly mentioned* as constants in the flat atomic assertions of AC (namely assertions of the form A `says` g where g is flat). Using the open DKAL rules, every principal knows of the existence of each of these elements. Note also that, according to SecPAL syntax, all regular components of the expression τf (aka f) are variables. Thus, the parts A `knows` t_i `exists` in the full forms of assertions $A :_d f$ `to` $all \leftarrow f_1, \dots, f_n, con$ in τ(AC) hold automatically under all relevant assignments. □

Remark 7.6. The following analog of the embedding theorem is true if one uses the translation of Remark 7.4, avoiding the extra rules of open DKAL: Let AC be a safe SecPAL assertion context, and let Π be the DKAL program consisting of the house rules and the assertions $\bigcup_{\alpha\in \mathrm{AC}} \tau\alpha$ with the translation τ of Remark 7.4. Further, let A be a SecPAL principal constant, f be a SecPAL ground fact expression, and d range over $\{0, \infty\}$. Suppose f is either flat, or nested with all its constants appearing in AC. If

$$\mathrm{AC}, d \vdash A \texttt{ says } f \tag{A}$$

in SecPAL then

$$\Pi(\mathrm{Sub}) \models A \texttt{ knows}_d \ f, \tag{B1}$$

$$\Pi(\mathrm{Sub}) \models e \texttt{ knows } A \texttt{ said}_d \ f \tag{B2}$$

in DKAL, and the assertions

$$A_{d'} : \ f' \texttt{ to } \mathit{all} \leftarrow f' \tag{B3}$$

belong to Π, where e ranges over all principals, $d' \in \{0, \infty\}$, and f' ranges over subfacts of f other than f itself. The proof is again an obvious induction on the length of the given SecPAL deduction. The implication if (A) then (B1) is the analog of the embedding theorem; the addition of the implications if (A) then (B2) and if (A) then (B3) is a strengthening needed in the inductive proof, as replacement for the first extra double rule of open DKAL.

Theorem 7.7. *The converse of the embedding theorem is not true. There is an assertion context AC and a SecPAL query* A `says` f *such that* $\Pi(\mathrm{Sub}) \models A$ `says` f *but* $\mathrm{AC}, \infty \not\vdash A$ `says` f.

Proof. Consider the following SecPAL assertion context AC:

1. A `says` B `can say` D `can say foo`.
2. B `says` C `can say foo`. (It's `foo`, not D `can say foo`.)
3. C `says` D `can say foo`.
4. D `says foo`.

In SecPAL, you get only these consequences:

5. C `says foo` (from 3 and 4).
6. B `says foo` (from 2 and 5).

But you do not get A `says foo`. On the other hand, making the translation to DKAL, and then translating the consequences back to SecPAL, you also get:

7. B `says` C `can say` D `can say foo` (from 2).
8. B `says` D `can say foo` (from 7 and 3).
9. A `says` D `can say foo` (from 1 and 8).
10. A `says foo` (from 9 and 4).

The difference between SecPAL and DKAL appears in line 7. In line 2, B trust C on `foo` and lets C delegate the trust with no depth restrictions. In DKAL, but not in SecPAL, this results in line 7, which formulates an instance of delegatability of the trust on the part of C. □

We see Theorem 7.7 as an advantage of DKAL: more justified requests get positive answers.

8. Discussion

Related work. The "speaks-for" paper [1] introduced the use of logic for expressing authorization policies in decentralized systems, and introduced the "says" modality, which has been used in trust management languages later on. Though the principals calculus of [1] is not part of the later, EFPL based languages, the constructs `delegates` and `represents` in Delegation Logic, `can say` and `can act as` in SecPAL, and `tdOn`, `canActAs`, and `canSpeakAs` in DKAL, all have functions similar to those of the operators `controls` and `speaks for` of [1]. DKAL incorporates a feature of speaks-for which is not present in the earlier EFPL languages, and that is the ability to nest quotations.

The expression "trust management" was coined in [8], which introduced PolicyMaker. PolicyMaker evolved into KeyNote [9]. KeyNote allows a principal to delegate a subset of his rights to another principal. An end principal has a right on a resource if there is a delegation chain for this right leading to him from the authority on the resource. KeyNote has thresholds, and so more involved scenarios, using directed graphs rather than chains, can come up. Still there are common authorization scenarios that cannot be expressed with KeyNote. Typically they involve a situation where a right is to be granted on the basis of an attribute, and the attribute originates from a source not directly related to the right. For an example see the introduction to [16].

Delegation Logic [15, 16], Binder [11], and SecPAL [4] are more recent languages, all based on EFPL. Binder builds very directly on Datalog, plus the modality "says" which allows Datalog programs of one principal to rely on assertions made by others. Trust by principal A in another principal B on a relation $pred(x_1, \ldots, x_n)$ is expressed by A placing the Datalog rule $pred(x_1, \ldots, x_n)$:- B `says` $pred(x_1, \ldots, x_n)$. Delegation Logic, SecPAL, and DKAL, all have vocabulary specifically designed for authorization policies. In all three, the semantics for the constructs which expressed trust, `delegates` in the case of Delegation Logic, `can say` in the case of SecPAL, and `tdOn` in the case of DKAL, are similar to the Datalog rule for trust in Binder. The similarity is greatest in DKAL. Binder and Delegation Logic are relational, except for the modality "says". SecPAL goes beyond that, by allowing nesting of `can say` and `can say`$_0$. In translation to Datalog with constraints, one introduces a new predicate for each nested fact occurring in the policy being translated. SecPAL also introduces a distinction between two kinds of "says", which we refer to as `says`$_\infty$ and `says`$_0$. (In the SecPAL terminology they correspond to $AC, \infty \models p$ `says` f and $AC, 0 \models p$ `says` f.) The distinction between the two is used to define semantics that prevent circumventing of delegation bounds.

The RT family languages [18] are also based on relational EFPL: Datalog in some cases and Datalog with constraints in others. The languages have roles instead of attributes, and principals may condition membership in a role they control on membership in a roles controlled by other principals. Both RT and SecPAL extend Datalog with constraints. In RT tractability with constraints is obtained by assuming that the constraint domain satisfies quantifier elimination. SecPAL uses instead a syntactic safety condition that guarantees that constraint variables are instantiated at the time of evaluation.

A recent language SeNDlog [2] adds targeting of communication to Binder. The targeting of communication, inspired by a database query language NDlog, adds to confidentiality and, as in DKAL, can be used to avoid information leakage. Though DKAL is more expressive, its semantics for targeting of communication are similar to those of SeNDlog.

Conclusion and future work. We designed an authorization language DKAL which exceeds the expressivity of previous languages in the literature in several respects, yet maintains the same time complexity bounds for answering authorization queries. The language has several innovative features, including targeted communication and a distinction between knowing and saying, very flexible formation of expressions allowing unrestricted use of functions that can be nested and mixed, extended use of an underlying substrate which may be very rich, strong semantics for quotations, and an information order that contributes to succinctness and understanding of the language.

We showed that policies written using SecPAL, a very expressive language recently proposed in [4], can be translated into DKAL, and in particular all authorization scenario expressible using SecPAL are also expressible using DKAL. We presented several situations, both user-centric and traditional, where the new features of DKAL are useful: for example in prevention of information leakage, abstraction of cryptographic signatures, and flexible design of a modular and distributed authorization policy. We presented an algorithm for answering queries to DKAL authorization policy, within the time complexity bounds of previous languages.

The DKAL query answering algorithm is currently implemented in Prolog. Future work includes developing criteria for ensue rules which may be added to the house rules by users without harming the time complexity results, syntax and semantics for targeting assertions (at the moment only targeting of infons is expressible within the language), and deployment.

Appendix A. Logic

Existential fixed-point logic, EFPL, was introduced in [7]. Here we present EFPL in a way that is convenient for our purposes. We also extend EFPL with first-order queries. The extension is straightforward but the resulting logic is more expressive than EFPL and may be called EFPL^{++}.

We presume some that the reader is familiar with some introductory text on mathematical logic, e.g. [13] or [22]. This appendix is essentially self-contained but it isn't a textbook.

A.1. **First-order logic.** We recall many-sorted first-order logic.

A.1.1. **Vocabulary.** A vocabulary consists of sort symbols, function symbols and relation symbols. Each function symbol and each relation symbol has a non-negative integer arity and a type. Nullary function symbols are known as constant symbols. It is presumed that there are only finitely many sort symbols, relation symbols and function symbols of positive arity. It is not excluded that the number of constant symbols is infinite.

The type of a function of positive arity r has the form

$$(D_1 \cup \cdots \cup D_n) \to S$$

where S is a sort symbol, $n \geq 1$, each component D_m has the form $S_1 \times \cdots \times S_r$ where each S_i is a sort symbol, and different components are distinct. Typically $n = 1$, but we have in the main part of the paper a binary function $\mathcal{I}$ of the type

$$(\text{Principal} \times \text{Speech}) \cup (\text{Regular} \times \text{Attribute}) \to \text{Info}.$$

The type of an r-ary relation has the same form except that S is replaced with the name Boole of the set that consists of the truth values "true" and "false". The type of a constant symbol is a sort symbol. The type of a nullary relation symbol is Boole. Every vocabulary

contains the equality sign which is a binary relation symbol. If $S_1, \ldots, S_\ell$ are all sort symbols, then the type of the equality symbol is

$$(S_1 \times S_1) \cup \cdots \cup (S_\ell \times S_\ell).$$

Each sort S of the vocabulary is endowed with an infinite list of variables, the variables of type S. All vocabulary symbols and all the variables are strings in a fixed finite alphabet.

A.1.2. **Total structures.** A total structure X of a vocabulary Υ consists of a nonempty set called the universe of X, together with interpretations of the vocabulary symbols over the universe. The sorts, that is the interpretations of the sort symbols, are subsets of the universe. Accordingly the domain $D_1 \cup \cdots \cup D_n$ of an r-ary function or relation consists of r-tuples of elements of (the universe of) X. The union of all sorts is the entire universe. A sort may be empty, and the sorts are not necessarily disjoint.

A function symbol F is interpreted by a function, called F or F_X, whose type is given by the type of the symbol F. Similarly a relation symbol P is interpreted by a relation, called P or P_X, whose type is given by the type of the symbol P. For example the binary function interpreting the function symbol $\mathcal{I}$ mentioned above has (i) one argument of type Principal and another argument of type Speech or (ii) one argument of type Regular and another argument of type Attribute. In either case, its values are of type Info.

The interpretations of the function and relation symbols are the basic functions and relations of X. A nullary basic function designates a particular element of X, a constant of X. The equality sign is interpreted in the obvious way. The vocabulary symbols are the identifiers of their interpretations in the structure. For example, two constants with the same value are different constants.

Typically the components D_m of the domain $D_1 \cup \cdots \cup D_n$ of a basic function or relation are disjoint but we do not require that. Indeed, if there are non-disjoint sorts S and S' then the components $S \times S$ and $S' \times S'$ of the equality domain are not disjoint.

A variable assignment over a structure X is a mapping from a set of variables to the universe of X. A variable of type S is mapped to an element of type S.

A.1.3. **expressions.** A constant of type S is a expression of type S. Similarly a variable of type S is a expression of type S. Composite expressions are constructed from constants and variables by means of function symbols of positive arity. If f is an r-ary function symbol of type $D \to S$ and $t_1, \ldots, t_r$ are expressions of types $S_1, \ldots, S_r$ respectively and $S_1 \times \cdots \times S_r$ is a component of D, then $f(t_1, \ldots, t_r)$ is a expression of type S.

Let X be a total structure and ξ a variable assignment over X. Given a expression t with variables in $\mathrm{Dom}(\xi)$, compute the value $Val^\xi_X(t)$ of t in X under ξ as follows. If t is a variable then $Val^\xi_X(t) = \xi(t)$. Suppose that $t = f(t_1, \ldots, t_r)$ where r may be zero, and let $Val^\xi_X(t_i) = a_i$ for $i = 1, \ldots, r$. Then $Val^\xi_X(t) = f_X(a_1, \ldots, a_r)$.

A.1.4. **Atomic formulas.** An atomic formula has the form $P(t_1, \ldots, t_r)$ where P is an r-ary relation symbol and $t_1, \ldots, t_r$ are expressions of types $S_1, \ldots, S_r$ such that $S_1 \times \cdots \times S_r$ is a component of the type of P.

Let X be a total structure, ξ a variable assignment over X, $P(t_1, \ldots, t_r)$ an atomic formula with variables in $\mathrm{Dom}(\xi)$, and $Val^\xi_X(t_i) = a_i$ for $i = 1, \ldots, r$. Then the truth value of $P(t_1, \ldots, t_r)$ is that of $P(a_1, \ldots, a_r)$.

A.1.5. **Formulas: syntax.** Atomic formulas are formulas, and all variables of an atomic formula φ are free in φ. Composite formulas are built from atomic formulas by means of propositional connectives and quantifiers $\forall$ and $\exists$. We will use only three propositional connectives: conjunction, disjunction and negation.

If φ is a formula then the negation $\neg\varphi$ is a formula, and the free variables of $\neg\varphi$ are those of φ. If φ and ψ are formulas, then the conjunction $\varphi \wedge \psi$ and the disjunction $\varphi \vee \psi$ are formulas. In either case, the free variables are those of φ plus those of ψ.

If φ is a formula then $(\forall v)\varphi$ and $(\exists v)\varphi$ are formulas. In either case, the free variables are those of φ minus v.

A.1.6. **Formulas: semantics.** Let X be a total structure and ξ a variable assignment over X. Consider formulas φ with free variables in $\mathrm{Dom}(\xi)$. Each such formula has a truth value $\mathrm{TrVal}^{\xi}_{X}(\varphi)$ in the structure X under ξ. If $\mathrm{TrVal}^{\xi}_{X}(\varphi)$ is "true", we say that φ holds (or is true) in X under ξ. And if $\mathrm{TrVal}^{\xi}_{X}(\varphi)$ is "false", we say that φ fails (or is false) in X under ξ. The computation of $\mathrm{TrVal}^{\xi}_{X}(\varphi)$ proceeds by induction on φ.

The case when φ is atomic was addressed above. If $\varphi = \neg\psi$ then the truth value of φ is the opposite of that of ψ. Suppose that φ is $\psi_1 \wedge \psi_2$ or φ is $\psi_1 \vee \psi_2$. In the case of conjunction, φ holds in X under ξ if both ψ_1 and ψ_2 hold; otherwise φ fails. In the case of disjunction, φ holds in X under ξ if at least one of the formulas ψ_i holds; otherwise φ fails.

Let v be a variable of type S, and suppose that φ is $(\forall v)\psi(v)$ or φ is $(\exists v)\psi(v)$. First, we consider two degenerated cases. If S is empty, then $(\forall v)\psi(v)$ holds and $(\exists v)\psi(v)$ fails in X under ξ. If S is nonempty but v is not a free variable of ψ then $\mathrm{TrVal}^{\xi}_{X}(\varphi) = \mathrm{TrVal}^{\xi}_{X}(\psi)$.

Suppose that S is nonempty and that v is a free variable of ψ. To emphasize the latter, we'll write $\psi(v)$ instead of ψ. For every element a of the sort S, let $\psi(a)$ be the truth value of $\psi(v)$ in X under the variable assignment obtained from ξ by mapping v to a (whether ξ was defined at v or not). If every $\psi(a)$ equals "true", then both $(\exists v)\psi(v)$ and $(\forall v)\psi(v)$ hold in X under ξ. If every $\psi(a)$ equals "false", then both $(\exists v)\psi(v)$ and $(\forall v)\psi(v)$ fail. Otherwise $(\exists v)\psi(v)$ holds but $(\forall v)\psi(v)$ fails.

A.1.7. **Bounded quantification.** We extend the syntax of formulas with an additional formula-formation rule: If $\beta(v)$ and $\psi(v)$ are formulas then so are

$$(\forall v : \beta(v))\psi(v), \quad (\exists v : \beta(v))\psi(v).$$

The colon reads "such that" here. The free variables of either formula are the free variables of $\beta(v)$ distinct from v plus the free variables of $\varphi(v)$ distinct from v.

Accordingly we extend the inductive definition of the semantics of formulas with an additional case where v is a variable of type S and φ is $(\forall v : \beta(v))\psi(v)$ or φ is $(\exists v : \beta(v))\psi(v)$. For every element a of type S, let $\beta(a)$ be the truth value of $\beta(a)$ in X under the variable assignment obtained from ξ by mapping v to a; and let s be the set of elements a such that $\beta(a)$ equals "true". The truth values of formulas $(\forall v : \beta(v))\psi(v)$ and $(\exists v : \beta(v))\psi(v)$ are defined as those of formulas $(\forall v)\psi(v)$ and $(\exists v)\psi(v)$ respectively except that v ranges over s rather than over S. In other words, think of s as a new type of the variable v.

Normally there is no need to extend the syntax of formulas by means of bounded quantification because the displayed formulas are equivalent to

$$(\forall v)[\beta(v) \to \varphi(v)], \quad (\exists v)[\beta(v) \wedge \varphi(v)]$$

respectively. One problem is that we don't want to use the implication sign. More importantly, we are about to introduce partial structures. The equivalences do not survive the generalization.

A.1.8. **Partial structures.** A *partial structure* is like a total structure except that basic functions may be partial. In particular some constants may be undefined. Total structures are special partial structures.

Contrary to prevailing logic tradition, our structures are by default partial. Accordingly, any non-empty subset U of a structure X gives rise to a *substructure* of X with universe U. Even if X is total, the substructure may be partial.

We described above how to evaluate expressions and formulas in a given total structure X under a sufficiently broad variable assignment ξ. The evaluation process is similar in the case when X is partial. Just take into account that, since basic functions may be undefined, the value of a expression may be undefined, and the truth value of a formula may be undefined.

Fortunately we do not have to deal with undefined values in the main part of the paper. Whenever we evaluate a formula φ in a structure X, the domains of the basic functions of X are sufficiently broad, so that undefined values do not come up during the evaluation of φ.

A.1.9. **Monotonicity lemma.**

Lemma A.1. *Let φ be a formula where negation is applied only to atomic formulas, and let P be a relation symbol such that no atomic formula with relation symbol P is negated in φ. Then φ is monotone in P in the following sense. Suppose that φ holds in a structure X under a variable assignment ξ. If you enlarge the interpretation P_X of P in X, without changing the rest of the structure or the variable assignment, then φ holds in the modified structure under the assignment ξ.*

The lemma is well known and is easily proved by induction on φ.

A.2. **Logic Programs.** Fix a vocabulary Υ. In this subsection, by default, expressions and formulas are of vocabulary Υ.

We presume that vocabulary Υ is split into two disjoint part, the *substrate part* Υ^- and *superstrate part* $\Upsilon - \Upsilon^-$, and that the superstrate part consists of relation symbols only. If Y is an Υ structure then the *substrate* of Y is the reduct of Y to Υ^-. In other words, the substrate is obtained from Y by forgetting the interpretations of the superstrate relation symbols. Those interpretations form the *superstrate* of Y. An atomic formula with a substrate (resp. superstrate) relation symbol is *substrate* (resp. *superstrate*) *atomic formula*.

A.2.1. **Syntax.** A *substrate constraint* is a quantifier-free formula in the substrate vocabulary.

A *logic rule* R of vocabulary Υ has the form $H \leftarrow B$ where H is a superstrate atomic formula and B is a conjunction of superstrate atomic formulas and at most one substrate constraint. (We could allow a rule to have several substrate constraints, but the conjunction of substrate constraints is a substrate constraint. Thus the "at most one substrate constraint" requirement does not restrict generality.) H is the *head* of the rule, and B is the *body*. B can be empty in which case R is *bodiless*. We typically write H alone for a bodiless rule $H \leftarrow B$.

Since our constants are strings in a fixed finite alphabet, and the same applies to our variables, the rule R is a string in a fixed finite alphabet. The *length* of R is the length of that string. The *width* of R is the number of variables in R.

If σ is a substitution, that is a function from variables to expressions, then $\sigma(R)$ is the rule obtained from R by simultaneously replacing every variable v with expression $\sigma(v)$; the rule $\sigma(R)$ is a *substitution instance* of R. Given an Υ structure X, we say that an assignment ξ of elements of X to the variables of R is *safe* for R over X if the value of every expression in R is defined.

A *logic program* Π of vocabulary Υ is a finite set of logic rules of vocabulary Υ. The head relation of any Π rule is a *head relation* of Π. The *length* of Π is the sum of the lengths of its rules. The *width* of Π is the maximum of the widths of its rules.

A.2.2. **Syntactic sugar.** Notation

$$H_1, \ldots, H_m \leftarrow B$$

stands for m rules $H_i \leftarrow B$. Notation

$$H \leftarrow B_1 \vee \cdots \vee B_n$$

stands for n rules $H \leftarrow B_j$. The two abbreviations can be used together. Notation

$$H_1, \ldots, H_m \leftarrow B_1 \vee \cdots \vee B_n$$

stands for mn rules $H_i \leftarrow B_j$.

A.2.3. **Semantics.** Given a structure X of substrate vocabulary Υ^-, a logic program Π computes the superstrate relations over X and thus computes a Υ structure $\Pi(X)$ with substrate X. We will describe the computation. In general, the computation is infinite but the case of interest to us is when X is finite. In that case, the computation is finite.

Partially order partial Υ structures with substrate X as follows: $Y \leq Z$ if $P_Y \subseteq P_Z$ for every superstrate relation symbol P. The program Π gives rise to an *immediate-action operator* Γ_Π on Υ structures with substrate X. If Y is an Υ structure then $\Gamma_\Pi(Y) \geq Y$. If P is a superstrate relation symbol of arity r then the interpretation $P_{\Gamma_\Pi(Y)}$ of P in $\Gamma_\Pi(Y)$ is the union of P_Y and the set of tuples $(a_1, \ldots, a_r)$ satisfying the following condition: there exists a rule $P(t_1, \ldots, t_r) \leftarrow B$ in Π and there exists a safe assignment ξ of elements of Y to the variables of the rule such that, in structure Y under assignment ξ, B holds and every t_i evaluates to a_i.

An Υ structure Y such that $\Gamma_\Pi(Y) = Y$ is a *fixed point* of Γ_Π. Since Γ_Π is monotone, by Knaster-Tarski theorem [25], there is the least fixed point of Γ_Π. That fixed point is the desired structure $\Pi(X)$ uniquely determined by Π over X. The original structure X is the substrate of $\Pi(X)$.

Here is one way to construct $\Pi(X)$. Let X_0 be the Υ structure obtained from X by initializing all superstrate relations to the empty relations of appropriate types. For each n, let $X_{n+1} = \Gamma_\Pi(X_n)$. Finally let X_ω be the limit of structures X_n which means that $P_{\Pi(X)} = \bigcup_n P_{X_n}$ for every superstrate relation symbol P. The limit structure X_ω is a fixed point of Γ_Π [7, Theorem 9]. It is easy to check by induction on n that $X_n \leq Y$ for every fixed point Y of Γ_Π. It follows that $X_\omega \leq Y$ for every such Y so that X_ω is the least fixed point of Γ_Π and therefore $\Pi(X) = X_\omega$. The Υ structures X_n will be called *standard approximations* to X_ω.

Remark A.2. In set theory, ω is the least infinite ordinal; that explains the use of ω here. Notice that the limit X_ω can be reached at some finite stage X_m in which case $X_n = X_m = X_\omega$ for all $n > m$.

A.2.4. **Complexity.** We analyze the fixed-point computation, that is the computation of $\Pi(X)$ described above, under some assumptions about the logic program Π and substrate X.

First, reflecting the peculiarity of our applications, we assume that the substrate elements split into two disjoint layers, *regular* and *synthetic*, so that we have *regular elements* and *synthetic elements*. Every substrate sort is a part of one of the two layers, so that we have *regular sorts* and *synthetic sorts*. All variables of Π are regular, that is of regular sorts. (The reader not interested in the splitting of X into two layers may want to concentrate on the special case where the synthetic layer is empty.)

Second, we assume that there is an algorithm Eval that evaluates the basic functions and relations of X. In the function case, given a function name F and an element tuple $\bar{a}$ of the appropriate length, Eval determines whether $F(\bar{a})$ is defined and, if yes, computes the value. In the relation case, given a relation name R and an element tuple $\bar{a}$ of the appropriate length, Eval determines whether $R(\bar{a})$ is true or false. Furthermore, we assume that Eval works in constant time. This allows us to abstract from the presentation form of the elements of X. In addition, the constant-time assumption simplifies the complexity analysis of the fixed-point computation. Essentially we will count only the number of Eval calls and will ignore Eval's computation time. Alternatively we could make a natural assumption that elements of X are given as strings and that Eval works in time bounded by a polynomial of the maximal string length. That polynomial would have to be taken into account in the following theorem but would not affect our exposition in any essential way. The analysis of Eval is orthogonal to the main issue of this paper.

Theorem A.3. *The time of the fixed-point computation is bounded by*

$$k \cdot N^r \cdot n^w \cdot O(\ell) \cdot o(N)$$

where k is the number of superstrate relations, ℓ is the length of Π, N is the total number of elements of X, r is the maximal arity of superstrate relations, n is the number of regular elements of X, and w is the width of Π.

Proof. The number of true (as well as all) instances of superstrate relations in $\Pi(X)$ is $k \cdot N^r$. An application of the immediate-action operator Γ_Π produces at least one new true instance of a superstrate relation unless the fixed point has been reached. It follows that the fixed point is reached in $k \cdot N^r$ steps, and so it remains to prove that the computation time of one application of Γ_Π is bounded by $n^w \cdot O(\ell) \cdot o(N)$.

Without loss of generality the whole program Π uses only w distinct variables. To compute the new true instances of the superstrate relations, it suffices to evaluate Π under each of the n^w assignments of regular elements of X to the variables of Π. It remains to prove that the evaluation time of Π under an assignment ξ is bounded by $O(\ell) \cdot o(N)$.

To evaluate Π under ξ, we traverse the parse tree for Π in the depth-first fashion. At some nodes, we call Eval to evaluate an instance of a substrate function or relation. At some other nodes, that belong to the bodies of logic rules, we check whether an instance of a superstrate relation is in the current table of the relation. Finally, at some of the remaining nodes, that belong to the heads of logic rules, we check whether an instance of a superstrate relation is

in the current table of the relation and, if not, then we insert it there. It suffices to show that the time needed to handle any single node is $o(N)$. This is trivial in the case of Eval due to our assumption that it works in constant time. This is also obvious for the table operations. The entries in the tables are in the lexicographical order, and binary search is used. □

Recall that the vocabulary is fixed. It follows that k is fixed.

Corollary A.4. (1) *Restrict attention to logic programs of bounded width. Then the computation time is bounded by ℓ times a polynomial in N. For a fixed program, the computation time is bounded by a polynomial in N.*

(2) *Restrict attention to logic programs of bounded width and assume that the total number N of substrate elements is bounded by a polynomial of the number n of regular elements. Then the computation time is bounded by ℓ times a polynomial in n. For a fixed program, the computation time is bounded by a polynomial in n.*

A.2.5. **Equivalence.** Logic programs Π_1 and Π_2 are *equivalent* if $\Pi_1(X) = \Pi_2(X)$ for every substrate structure X. Rules R_1 and R_2 are *equivalent* if $\Pi \cup \{R_1\}$ is equivalent to $\Pi \cup \{R_2\}$ for every program Π. The following lemma is obvious

Lemma A.5. *A rule $P(t_1, \ldots, t_r) \leftarrow B$ is equivalent to*

$$P(v_1, \ldots, v_r) \leftarrow B \wedge v_1 = t_1 \cdots \wedge v_r = t_r.$$

where $v_1, \ldots, v_r$ are fresh variables.

A.3. **EFPL and EFPL^{++}.** We recall the notion of EFPL formulas and introduce EFPL^{++}queries.

Again we fix a vocabulary that is split into a substrate and a purely relational superstrate. In [7], substrate relations are called negatable, and superstrate relations are called positive.

A.3.1. **Existential fixed-point logic.** EFPL formulas are defined by induction in [7]. They start with atomic formulas and the negations of atomic substrate formulas. They use only two propositional connectives to build new formulas: conjunction and disjunction. They use existential quantification, and do not use universal quantification; that is behind the E in the notation EFPL. In our case, where structures are partial, we would use also bounded existential quantification. And that there is recursion, in the form of a fixed point; that explains the FP in the notation EFPL. Logic programs are used to populate superstrate relations. We do not use EFPL formulas in the main part of this paper, and so we refer the interested reader to paper [7] for details.

Let us mention, however, that every EFPL formula with nested recursions can be rewritten in an equivalent form with no nested recursions. The following proposition illustrates the un-nesting process.

Proposition A.6. *Consider logic programs Π_1 and Π_2 such that (a) the head relations of Π_1 may occur only in the bodies of Π_2 rules and (b) the head relations of Π_2 do not occur at all in Π_1. And let X be an arbitrary substrate structure. Then $\Pi_2(\Pi_1(X)) = (\Pi_1 \cup \Pi_2)(X)$.*

Conditions (a) and (b) guarantee that structure $\Pi_2(\Pi_1(X))$ is well defined.

Proof. To simplify notation, assume that Π_1 has only one head relation P, and Π_2 has only one head relation Q. Let Γ_1, Γ_2, Γ_3 be the operators Γ_{Π_1}, Γ_{Π_2} and $\Gamma_{\Pi_1 \cup \Pi_2}$ respectively. Γ_1 operates on structures of vocabulary $\Upsilon^- \cup \{P\}$ with Υ^- reduct X; let Y_1 be the least fixed

point of Γ_1. Γ_2 operates on structures of vocabulary Υ with $\Upsilon^- \cup \{P\}$ reduct Y_1; let Y_2 be the least fixed point of Γ_2. And Γ_3 operates on structures of vocabulary Υ with Υ^- reduct X; let Z be the least fixed point of Γ_3. We need to show that $Y_2 = Z$. We prove that $Y_2 \leq Z$ and $Z \leq Y_2$.

As far as relation P is concerned, there is no difference between Π_1 and $\Pi_1 \cup \Pi_2$, and so $P_{Y_1} = P_Z$. It follows that Y_1 is the Υ^- reduct of Z and thus Γ_2 is applicable to Z. Since $\Gamma_3(Z) = Z$, no rule in $\Pi_1 \cup \Pi_2$ produces any new tuples at Z. It follows that $\Gamma_2(Z) = Z$ and so $Y_2 \leq Z$ by the definition of Y_2. Further, Γ_3 is applicable to Y_2. Since $\Gamma_2(Y_2) = Y_2$, no Π_2 rule produces any new tuples at Y_2. Since Π_1 does not use any head relation symbols of Π_2, the program Π_1 operates on the $\Upsilon^- \cup \{P\}$ reduct of Y_2 which is Y_1 and which is equal to $\Gamma_1(Y_1)$; so no Π_1 rule produces any new tuples at Y_2. It follows that $\Gamma_3(Y_2) = Y_2$ and so $Z \leq Y_2$ by the definition of Z. □

A.3.2. **EFPL^{++}.** We introduce EFPL^{++}queries and quickly discuss what should EFPL^{++}formulas be.

Syntactically an EFPL^{++} *query* is simply a first-order formula where the negation can be applied only to atomic substrate formulas. Note that universal quantification is allowed.

Let $\varphi(v_1, \ldots, v_k)$ be a query with free variables as shown. To compute the truth value of a query $\varphi(v_1, \ldots, v_k)$, we need three things. First we need a substrate structure X. Second we need a logic program Π such that every superstrate relation of φ is a head relation of Π. Finally we need a variable assignment ξ over X whose domain includes $\{v_1, \ldots, v_k\}$. The desired truth value of φ given appropriate X, Π and ξ is the truth value of the first-order formula φ at structure $\Pi(X)$ under ξ.

In addition to the truth value of the query $\varphi(v_1, \ldots, v_k)$, we define the answer to $\varphi(v_1, \ldots, v_k)$. This requires only two things: a substrate structure X and a logic program Π such that every superstrate relation of $\varphi(v_1, \ldots, v_k)$ is a head relation of Π. Let $\bar{v}$ be the tuple $(v_1, \ldots, v_k)$, so that $\varphi(\bar{v})$ is $\varphi(v_1, \ldots, v_k)$. For any tuple $\bar{a} = (a_1, \ldots, a_k)$ of elements of X, let $\bar{v} \mapsto \bar{a}$ be the variable assignment ξ such that $\xi(v_i) = a_i$ for all $i = 1, \ldots, k$. We say that $\varphi(\bar{a})$ holds in X if $\varphi(\bar{v})$ holds in X under the variable assignment $\bar{v} \mapsto \bar{a}$. The *answer* to query $\varphi(\bar{v})$ at X under Π is the set $\{\bar{a} : \ \varphi(\bar{a}) \text{ holds at } X\}$.

We do not use EFPL^{++}formulas in the main part of the paper. Such a formula is given by a logic program Π and a query φ whose superstrate relations are head relations of the program. A formula given by Π and φ is evaluated at a substrate structure X under an appropriate variable assignment ξ. Its truth value is the truth value of query φ at X under Π and ξ.

Appendix B. Tables of vocabulary, house rules, and assertion forms

The tables below present the DKAL vocabulary and house rules. DKAL assertion forms are summarized following the second table.

Regular Sort	::=	Regular
	|	Principal
	|	...
Synthetic Sort	::=	Synthetic
	|	Info
	|	Speech
	|	Attribute
Regular Function Symbol	::=	...
Synthetic Function Symbol	::=	$\texttt{said}_d$: Info $\to$ Speech
	|	$\texttt{tdOn}_d$: Info $\to$ Attribute
	|	`canActAs`: Principal $\to$ Attribute
	|	`canSpeakAs`: Principal $\to$ Attribute
	|	$\mathcal{I}$: (Regular $\times$ Attribute) $\cup$ (Principal $\times$ Speech) $\to$ Info
	|	`exists`: *Attribute*
	|	...
Substrate Relation Symbol	::=	`regcomp` : Regular$\times$Synthetic
	|	...
Superstrate Relation Symbol	::=	ensues : Info$\times$Info
	|	$\texttt{knows}_d$: Principal$\times$Info
	|	$\texttt{saysto}_d$: Principal$\times$Info$\times$Principal

Here subscript $d \in \{0, \infty\}$, and subscript ∞ is usually omitted. $\mathcal{I}$ usually omitted in writing. Relation `ensues` is abbreviated to $\leq$.

Figure 6. DKAL vocabulary

(Say2know)	p `knows` q `said`$_d$ x	$\leftarrow$	q `says`$_d$ x `to` p
(KMon)	a `knows`$_d$ x	$\leftarrow$	a `knows`$_d$ y, $x \leq y$
(KSum)	a `knows`$_d$ $x_1 + x_2$	$\leftarrow$	a `knows`$_d$ x_1, a `knows`$_d$ x_2
(K0∞)	p `knows` x	$\leftarrow$	p `knows`$_0$ x
(EOrder)	$x \leq x$		
	$x \leq z$	$\leftarrow$	$x \leq y, y \leq z$
(ESum)	$x \leq x + y$		
	$y \leq x + y$		
	$x + y \leq z$	$\leftarrow$	$x \leq z, y \leq z$
(SaidSum)	p `said`$_d$ $(x + y) \leq p$ `said`$_d$ $x + p$ `said`$_d$ y		
(Exists)	t `exists` $\leq x$	$\leftarrow$	t `regcomp` x
(TrustApp)	$x \leq p$ `said`$_d$ x `+` p `tdOn`$_d$ x		
(Del)	p `tdOn` $(q$ `tdOn`$_d$ $x) \leq p$ `tdOn` $x + q$ `exists`		
(SaidMon)	p `said`$_d$ $x \leq p$ `said`$_d$ y	$\leftarrow$	$x \leq y$
(Trust0∞)	p `tdOn`$_0$ $x \leq p$ `tdOn` x		
(Said0∞)	p `said` $x \leq p$ `said`$_0$ x		
(SelfQuote)	p `said`$_d$ $x \leq p$ `said`$_d$ p `said`$_d$ x		
(Del$^-$)	p `tdOn`$_d$ $x \leq p$ `tdOn`$_d$ p `tdOn`$_d$ x		
(Role)	p *attribute* $\leq q$ *attribute* $+ p$ `canActAs` q		
	q *speech* $\leq p$ *speech* $+ p$ `canSpeakAs` q		

Again, subscript $d \in \{0, \infty\}$, and subscript ∞ is usually omitted.

FIGURE 7. DKAL House Rules

Assertion forms. There are two forms of DKAL assertions:

1. `A` :$_d$ x $\leftarrow$ $x_1, \ldots, x_n$, *con*,
2. `A` :$_d$ x `to` p $\leftarrow$ $x_1, \ldots, x_n$, *con*.

Here A in both forms is a ground principal expression denoting the *owner* of the assertion; d is either ∞ or 0, and ∞ is typically omitted; $x, x_1, \ldots, x_n$ are infon expressions; and *con* is a *substrate constraint*, that is a conjunction of possibly-negated atomic formulas in the substrate vocabulary. All variables are of type Regular; p is a principal variable called the *target variable*. Assertion 1 is a *knowledge assertion*. It does not have a target variable, and

it gives rise to rule

$$\begin{aligned}&\texttt{A knows}_d\ x \leftarrow\\ &\quad \texttt{A knows}_d\ x_1\texttt{,...,A knows}_d\ x_n,\\ &\quad \texttt{A knows}_d\ t_1\ \texttt{exists,...,}\\ &\quad \texttt{A knows}_d\ t_k\ \texttt{exists,}\ \mathit{con}\end{aligned}$$

where the list $t_1, \ldots, t_k$ consists of the variables in $x, x_1, \ldots, x_n$, *con* and of the non-ground regular components of assertion head x. Assertion 2 is a *speech assertion* and gives rise to rule

$$\begin{aligned}&\texttt{A says}_d\ x\ \texttt{to}\ p \leftarrow\\ &\quad \texttt{A knows}_d\ x_1\texttt{, ..., A knows}_d\ x_n\texttt{,}\\ &\quad \texttt{A knows}_d\ t_1\ \texttt{exists,...,}\\ &\quad \texttt{A knows}_d\ t_k\ \texttt{exists,}\ \mathit{con}\end{aligned}$$

where the list $t_1, \ldots, t_k$ consists of the variable of the assertion and the non-ground regular components of the assertion head x, with the exception of the target variable p.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin, "A calculus for access control in distributed systems," ACM Transactions on Programming Languages and Systems, 15:4, 706–734, 1993.

[2] Martín Abadi and Boon Thau Loo, "Towards a Declarative Language and System for Secure Networking", in International Workshop on Networking Meets Databases (NetDB '07), 2007

[3] Moritz Y. Becker, Cédric Fournet and Andrew D. Gordon, "SecPAL: Design and Semantics of a Decetralized Authorizatoin Language", Technical Report MSR-TR-2006-120, Microsoft Research, September 2006.

[4] Moritz Y. Becker, Cédric Fournet and Andrew D. Gordon, "SecPAL: Design and Semantics of a Decetralized Authorizatoin Language", 20th IEEE Computer Security Foundations Symposium (CSF), 3–15, 2007.

[5] Moritz Y. Becker and Peter Sewell, "Cassandra: distributed access control policies with tunable expressiveness", in IEEE 5th International Workshop on Policies for Distributed Systems and Networks, 159-168, 2004.

[6] Moritz Y. Becker and Peter Sewell, "Cassandra: Flexible trust management, applied to electronic health records", in IEEE Computer Security Foundations Workshop, 139-154, 2004.

[7] Andreas Blass and Yuri Gurevich, "Existential fixed-point logic", Springer Lecture Notes in Computer Science 270 (1987), 20–36.

[8] Matt Blaze, and Joan Feigenbaum, and Jack Lacy, "Decentralized trust management", in Proceedings 1996 IEEE Symposium on Security and Privacy, 164–173, 1996.

[9] Matt Blaze, Joan Feigenbaum, Angelos D. Keromytis, "The Role of Trust Management in Distributed Systems Security", in Secure Internet Programming, 185–210, 1999.

[10] S. Cantor, J. Kemp, R. Philpott, and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`.

[11] John DeTreville, "Binder, a logic-based security language", in IEEE Symposium on Security and Privacy, 105-113, 2002.

[12] Blair Dillaway, "A unified Approach to Trust, Delegation, and Authorization in Large-Scale Grids," Technical Paper, Microsoft Corporation, September 2006.

[13] Herbert Enderton, "A Mathematical Introduction to Logic", 2nd edition, Academic Press.

[14] Wilfrid Hodges, "Model Theory", Cambridge University Press, 1993.

[15] Ninghui Li, "Delegation Logic: A Logic-based Approach to Distributed Authorization", Ph.D. thesis, New York University, September 2000.

[16] Ninghui Li, Benjamin N. Grosof and Joan Feigenbaum, "Delegation logic: A logic-based approach to distributed authorization", ACM Transactions on Information and System Security (TISSEC) 6:1 (February 2003), 128–171.

[17] Ninghui Li and John C. Mitchell, "Datalog with Constraints: A Foundation for Trust Management Languages", PADL 2003, V. Dahl and P. Wadler (Eds.), LNCS 2562, 58–73, 2003.

[18] Ninghui Li and John C. Mitchell, "RT: A role-based trust-management framework", in Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX III), 201–212, April 2003.

[19] Ninghui Li, William H. Winsborough, and John C. Mitchell, "Beyond Proof-of-compliance: Safety and Availability Analysis in Trust Management", in Proceedings of 2003 IEEE Symposium on Security and Privacy, 123–139, May 2003.

[20] B.T. Loo, T. Condie, M. Garofalakis, D.E. Gay, J.M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative Networking: Language, Execution and Optimization." In Proceedings of ACM SIGMOD International Conference on Management of Data (June 2006).

[21] Yuri Matiyasevich, *Hilbert's Tenth Problem,* MIT Press, 1993.

[22] Elliott Mendelson, "Introduction to Mathematical Logic," 4th edition, Academic Press.

[23] OASIS. Security Assertion Markup Language (SAMcfL). `www.oasis-open.org/committees/security`.

[24] Oxford English Dictionary, 2nd edition, Oxford University Press, 1989.

[25] Alfred Tarski, "A lattice-theoretical fixpoint theorem and its applications", *Pacific Journal of Mathematics* 5:2 (1955), 285–309.