DKAL: DISTRIBUTED-KNOWLEDGE AUTHORIZATION LANGUAGE

YURI GUREVICH AND ITAY NEEMAN

ABSTRACT. DKAL is an expressive declarative authorization language based on existential fixed-point logic. It was inspired by SecPAL but is considerably more expressive within the same bounds of computational complexity. Distributed knowledge is the most conspicuous distinguishing feature of DKAL; in particular it makes DKAL appropriate for user-centric access control. Other distinguishing features include

- targeted communication that is beneficial with respect to efficiency, privacy, security and liability aspects,
- information order of facts that makes computations more efficient,
- reflection that allows principals to speak about what has been said to them,
- stronger delegation.

CONTENTS

Comments are welcome.

1. Introduction

This paper takes its departure in SecPAL, a declarative authorization language developed by Becker, Fournet, and Gordon [2]. SecPAL covered many access control scenarios and substantially advanced the state of art in the area. We noticed that the right way to see SecPAL is in the context of existential fixed-point logic. This provides a solid model-theoretic foundation for SecPAL. Then we realized that the same foundation supports a far more expressive language within the same bounds of computational complexity. The new and more expressive authorization language is called Distributed Knowledge Authorization Language, in short DKAL.

We tried to make this paper self-contained. In particular, we do not presume that the reader is familiar with SecPAL, but the familiarity with SecPAL is beneficial in a few places where we discuss SecPAL. We do presume that the reader is familiar with the basics of first-order logic.

Existential fixed-point logic (EFPL). EFPL 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 [3]. The least fixed-point operator enables induction. For example, given a directed graph with edge relation E, a logic program

$$\begin{aligned} T(x,y) &\leftarrow E(x,y), \\ T(x,y) &\leftarrow T(x,z) \wedge E(z,y). \end{aligned}$$

computes the transitive closure T of relation E. The given structure is the *substrate*, and the relations computed by the program are *superstrate relations*. In our example, the directed graph is the substrate, the edge relation E is a substrate relation, and the transitive closure T is a superstrate relation. In the appendix, we recall EFPL in the form appropriate for this article.

SecPAL in the EFPL context. View the SecPAL domain of constants as a multi-sort structure. Extend this structure by means of verbphrases and facts produced by SecPAL predicates and SecPAL constructs can-say, can-say_0 and can-act-as. The extended structure is the substrate, and SecPAL predicates as well as the three constructs are substrate functions. SecPAL has three deduction rules, called (cond), (cansay) and (can act as), that "capture the semantics of the language." In the new framework, the (cond) rule is not needed as it is absorbed into the EFPL. The other two rules together with the given assertions form a logic program that computes the superstrate relation says over the substrate in the same way the logic program above computes the relation T over a given directed graph with edge relation E. Theorem A.4 gives the complexity of the computation. The SecPAL polynomial data complexity theorem [2, Theorem 8.2] easily follows from Theorem A.4.

Distributed knowledge. Distributed knowledge is the most conspicuous distinguishing feature of DKAL, Distributed Knowledge Authorization Language. In the final account, authorization is about knowledge: a resource manager allows you to use his resource if he[1] knows that you are authorized to use the resource. Accordingly the key relation of DKAL is a relation knows between principals and facts. Because of the communication among the principals, the knowledge relation is recursive.

[1]It is convenient to use anthropomorphic language in connection with principals. But of course principals could be and often are inanimate.

The substrate and superstrate of a state of knowledge. A state of knowledge splits into a substrate and a superstrate. The substrate is a multi-sort structure. A part of its universe consists of *regular elements*, notably principals. This part is determined by the intended application. For example, some elements may denote buildings, offices, printers, etc. The regular part supports various relations and functions. For example, there may be a binary relation "p is a secretary of q" or a unary function manager(p). The rest of the substrate universe is constructed from the regular elements by free constructors. Some of the free constructors are application-independent. For example, there is a free constructors said of type Principal × Fact → Speech.

The superstrate consists of relations knows, knows_0, saysto, $\mathsf{says}_0\text{-}\mathsf{to}$, ensues. The superstrate relations are given by a logic program that operates over the substrate. Relations knows and knows_0 are of type Principal×Fact. Relation knows_0 reflects the internal (or initial, prior) knowledge of principals. Relations saysto and $\mathsf{says}_0\text{-}\mathsf{to}$ are of type Principal×Fact×Principal. Intuitively they are communications. A communication of the form p says_0 foo to q is based on the internal knowledge of p. The intended meaning of communications is discussed in §2.1. Here, in the introduction, we speak more about the relations knows and saysto than their counterparets knows_0 and $\mathsf{says}_0\text{-}\mathsf{to}$.

Remark 1.1 (Facts). The term fact is used in DKAL in an unorthodox way. Facts are substrate elements. It is never a question whether a fact foo is a true fact. A typical question of interest is whether a principal p knows foo.

Targeted communication. The concept of knowledge would not be very interesting if, as in SecPAL, the same facts are communicated to all principals. But DKAL is different. DKAL communication is targeted. Normally communication is restricted to some addressees, presumably those who need to know the information. The house rule

$$p \ \mathsf{knows}\ q\ \mathsf{said}\ \mathsf{foo} \leftarrow q\ \mathsf{says}\ \mathsf{foo}\ \mathsf{to}\ p \tag{1}$$

relates a targeted communication q says foo to p with the knowledge of p. Note that, by house rule 1, p does not learn foo; he learns only that q said foo. The principal p is the intended audience for the communication. The audience restriction appears already in SAML [9]. But in our case learning is more consequential. The knowledge of a principal includes (a) his internal knowledge, (b) facts communicated to him by other principals, and (c) facts that ensue (a) + (b).

SecPAL communication is via relation says, a principal says a fact. This is really a broadcast. It can be modeled in DKAL by says to all where all is a fresh principal variable. Targeted communication is important for privacy and security. It allows us to cut out irrelevant knowledge of principals and thus make the system more efficient; a principal has to deal only with communications addressed to him. There is also a liability issue. Consider for example two states, S_1 and S_2 such that, in S_1, an 18 year old can buy alcohol, whereas, in S_2, one needs to be at least 21 to buy alcohol. An agency A of state S_1 issues Bob a document addressed to S_1 wine shops that allows the shops to sell wine to Bob. If Bob buys alcohol from a wine shop in state S_2, in violation of S_2 law, agency A is not liable. The document has not been addressed to wine shops in S_2.

Open-world scenario and the user-centric approach. One obvious DKAL scenario is a close security system of an organization, e.g. a large corporation. A distributed system of principals is aided by a trusted DKAL engine that computes the queries on behalf of the principals. But there is also another DKAL scenario, an open-world scenario of communicating entities. The entities could be of different sizes, from single users to large corporations. Each entity has its own state of knowledge influenced by communications from the other entities. Each entity has its own DKAL engine.

The user-centric access control is conducive to the open-world scenario. A user is a separate entity. She may want to buy some products from a company A without giving the company A her credit card information. To this end, she may engage a security company B that will mediate between the user and the company A. The company B will not know what the user buys exactly, and the company A will have no information about the credit card of the user. In this little example we have already three distinct entities with separate states of knowledge.

This paper is not restricted to the close-world scenario but it is restricted to one vocabulary and one global state of knowledge. The communication among entities involves interesting issues. If an entity α communicates a fact to a different entity β, then the fact should be in the vocabulary of β. What exactly should α know about β? How does α get that information? We intend to consider the open-world scenario more generally elsewhere.

The ensue relation. One innovation of DKAL is the information order on facts. If a fact x is less informative or precisely as informative as a fact y, we say that x ensues y and we write $x \lesssim y$. (We resurrect the transitive meaning of ensue: “To follow as a result or consequence; to result from. Obs.” [10].) The house rule

$$p \textsf{ knows } x \leftarrow x \lesssim y \wedge (p \textsf{ knows } y) \tag{2}$$

reflects our intention behind the ensue relation: the knowledge of every principal is closed under the information order. The introduction of the information order that is independent of any particular principal, enhances efficiency. The (can say) and (can act as) rules of SecPAL gave rise to some ensue clauses of DKAL. One of these clauses is

$$p \ \mathit{attribute} \ \lesssim \ (q \ \mathit{attribute}) \ \& \ (p \textsf{ canActAs } q). \tag{3}$$

Remark 1.2 (Function &). One may wonder why we use function & rather than logical conjunction $\wedge$ in rule 3. This is related to the role that facts play in DKAL. Facts are not formulas; they are substrate elements that don't have truth values. The function & is the conjunction of facts.

Delegation. Delegation is a strong side of SecPAL. DKAL is stronger yet in this respect. DKAL has a powerful house rule

$$q \textsf{ cansay } r \textsf{ cansay } x \lesssim (q \textsf{ cansay } x) \ \& \ (r \textsf{ exists}) \tag{4}$$

Remark 1.3. The fact r exists may seem confusing: Are there non-existent principals? No, there are no non-existen principals. But look at house rule 4 from the point of view of a principal p of the house rule 2. If p knows q cansay x and wants to jump to the conclusion q cansay r cansay x, he needs to know of r. This helps us to control the knowledge of p from exploding. Rule 4 does not require that q necessarily knows of r. $\square$

In Section 6, we give a natural translation of SecPAL into a “crippled” version of DKAL, called Open DKAL, where in particular the audience restriction is removed. Under this translation, every provable SecPAL formula is provable in Open DKAL. In addition, due to the powerful delegation rule, some justifiable SecPAL formulas, unprovable in SecPAL, are provable in Open DKAL. Thus more justifiable requests expressible in SecPAL get positive answers in DKAL.

Reflection. The Speaks-For [1] calculus enables one to speak about speeches which is impossible in SecPAL. For example, a Speaks-For formula may have the form A says B says x. It may seem that reflection is not available in languages based on existential fixed-point logic where superstrate relations cannot be nested. For example, one cannot have an assertion A says (B says x to A) to C in DKAL. We get around this obstacle by means of a built-in function said of type Principal $\times$ Fact $\to$ Speech. The missing assertion can be expressed thus: A says (B said x) to C. And the function said can be nested. So we may have for example assertions of the form A says (B said C said D said x) to E.

Worst-case complexity. A basic query has the form $p\ \mathsf{knows}\ t(v_1,\dots,v_k)$ or $p\ \mathsf{knows}_0\ t(v_1,\dots,v_k)$ where p is a ground (that is with no variables) term of type Principal, t is a term of type Fact, and the variables v_i range over regular elements. In Section 5, 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 a Q under a given authorization policy $\mathcal{A}$. Here the complete answer is the set of tuples $(b_1,\dots,b_k)$ of regular elements of appropriate types such that the principal p knows the fact $t(b_1,\dots,b_k)$ under the authorization policy $\mathcal{A}$. The runtime of the algorithm is bounded by a polynomial in $\ell^{\,\delta+w}$ 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 said_0 (possibly mixed) are nested in the policy and query term, and
- w is the maximal number of variables 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 §3.4, we introduce more general queries: first-order queries 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.

Acknowledgements. Perceptive comments of Andreas Blass helped us greatly in debugging the paper. Conversations with Slava Kavsan were educational. Close collaboration with M. Paramasivam was most helpful.

2. 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

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

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 $(F, b_1, \dots, b_j)$ 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.

2.1. **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.

2.1.1. **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, Fact, and Speech. The regular sorts include at least the sort Principal.

Element of regular sorts *regular elements*, and elements of synthetic sorts are *synthetic elements*. A basic substrate function is a *synthetic function* if it takes values of a synthetic sort; otherwise it is a *regular function*. All basic substrate relations are *regular relations*. Variables of regular sorts are *regular variables*, and variables of synthetic sorts are *synthetic variables*. Compound terms $f(t_1, \dots, t_j)$, where j may be zero, are *regular terms* if f is regular, and are *synthetic terms* 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 $b = F(b_1, \dots, b_n)$ for a some synthetic function F and some elements $b_1, \dots, b_n$ where F and the tuple $(b_1, \dots, b_n)$ 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).

2.1.2. **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 a fact $f =$ 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 term t define a *syntactic tree* of t as follows:

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

A subterm s of t is a *regular component* of t if s is a leaf of syntree(t). For example, consider a fact term $t = \mathsf{manager}(v)$ can sing. Then the subterm $\mathsf{manager}(v)$ is a regular component of t (but v is not).

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

- Functions said and said_0 of type Fact → Speech.
- Functions cansay and cansay_0 of type Fact → Attribute.
- A function fact of type (Principal × Speech) ∪ (Regular × Attribute) → Fact. Here Regular is the union of all regular sorts.
- A function & (pronounce "and") of type Fact × Fact → Fact.
- Functions canActAs and canSpeakAs of type Principal → Attribute.
- A constant exists of type Attribute.

Convention 2.1. Function symbols said and cansay can be written as said_∞ and cansay_∞ respectively. Thus said_d denotes said when $d = \infty$ and denotes said_0 when $d = 0$, and similarly for cansay_d. In the case of functions said_d, cansay_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 fact we generally omit the function name altogether writing just Bob is a user rather than fact(Bob, is a user). □

Proviso 2. Every additional (to the house constructors) synthetic function takes attribute values.

2.1.4. **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 facts. Function fact has two subfunctions, one of the type Principal× Speech → Fact, and the other of the type Regular × Attribute → Fact. The first generates *spoken facts*, and the second *attribute facts*.

Internal knowledge and undelegatable authorization. The difference between p said foo and p said_0 foo is that the latter reflects the internal (initial, prior) knowledge of the principal p. The difference between p cansay foo and p cansay_0 foo is that the first allows p to delegate the authority and the second does not.

Can internal knowledge be acquired? One may argue that a fact of the form p 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 facts. 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 4.5 in this connection.

Conjunction of facts. 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 facts $f\&g$ and $g\&f$ are different.

Example 2.1. Here are examples of attributes, speeches and facts.

- canActAs Director is an attribute where Director is a principal. Syntactically it is an attribute term where Director is a principal constant.
- Alice canActAs Director is a fact obtained by applying the function fact to the principal Alice and the attribute canActAs Director. Syntactically it is an attribute fact term with principal constants Alice and Director.
- canRead *file* is an attribute term 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 a fact term obtained by applying the function fact to a principal variable p and the attribute constant is a trusted merchant.
- said_0 p is a trusted merchant is a speech term obtained by applying the function said_0 to the fact term p is a trusted merchant.
- Alice said_0 Bob is a trusted merchant is a spoken fact obtained by applying the function fact to the principal Alice and the speech said_0 Bob is a trusted merchant. Syntactically it is a spoken-fact term.
- STS said Alice said_0 Bob is a trusted merchant is a spoken fact obtained by applying the function fact to the principal STS and the speech said Alice said_0 Bob is a trusted merchant. Syntactically it is a spoken-fact term.

2.2. **Superstrate relations.** In our approach, facts are state elements rather than propositions that can be true or false. One of the superstrate relations is knows of type Principal × Fact. Intuitively it consists of pairs (p, x) so that fact x is known to principal p. We write p knows x rather than $\mathsf{knows}(p, x)$.

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 × Fact × Principal. Formally saysto consists of triples (p, x, q) such that principal p says fact x to principal q, but we write p says x to q rather than $\mathsf{saysto}(p, x, q)$.

Intentionally p says x to q is more than a simple communication. In DKAL (and in SecPAL for that matter), saying is always an attempt to speak authoritatively. In other words, p says x to q means that principal p makes an attempt to say fact x to principal q authoritatively. Whether the attempt is successful depends on whether p has the authority to promulgate x. In DKAL, the key property is the implication

$$(q \ \mathsf{knows}\ p \ \mathsf{said}\ x) \wedge (q \ \mathsf{knows}\ p \ \mathsf{cansay}\ x) \to q \ \mathsf{knows}\ x.$$

(See Lemma 3.4.2.) If q knows that p has attempted to say x authoritatively and if q knows that p has the authority to promulgate x then q knows the fact x.

We wish to keep track of whether a principal's knowledge of a fact does or does not rely on communications from other principals. For this we have a third superstrate relation knows_0 of type Principal × Fact. Formally it consists of pairs (p, x) such that fact x is known to principal

p internally, independently of assertions made by other principals. We write $p\ \mathsf{knows}_0\ x$ rather than $\mathsf{knows}_0(p,x)$.

A principal may inform other principals not only that he knows a fact, but that he knows it on the basis of his internal knowledge. For this purpose we have a fourth superstrate relation saysto_0 of type Principal × Fact × Principal. Formally it consists of triples (p,x,q) such that principal p says to principal q that he knows fact 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 $p\ \mathsf{says}_0\ x\ \mathsf{to}\ q$ rather than $\mathsf{saysto}_0(p,x,q)$.

Formulas

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

can be written in the form

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

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

Our final superstrate relation ensues is of type Fact × Fact. 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 $x \lesssim y$ instead of $\mathsf{ensues(x,y)}$.

The superstrate relations satisfy various policy rules of the form:

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

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* $P(t_1,\dots,t_r)$ should be true as well.

For example, according to rule

$$q\ \mathsf{knows}\ p\ \mathsf{said}\ \mathit{fact}\ \leftarrow p\ \mathsf{says}\ \ \mathit{fact}\ \mathsf{to}\ q$$

the following holds for any p,q and any *fact*: if the triple $(p, fact, q)$ satisfies relation saysto then the pair $(p,\ q\ \mathsf{said}\ \mathit{fact})$ 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 2.2. 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.

3. 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.*

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

$$\text{(As1)} \quad \begin{aligned} A \ \mathsf{knows}_d\ x \ \leftarrow\ & A\ \mathsf{knows}_d\ x_1 \wedge \cdots \wedge A\ \mathsf{knows}_d\ x_n \wedge con \wedge \\ & A\ \mathsf{knows}_d\ t_1\ \mathsf{exists}\ \wedge \cdots \wedge A\ \mathsf{knows}_d\ t_k\ \mathsf{exists} \end{aligned}$$

in short

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

The second form is

$$\text{(As2)} \quad \begin{aligned} A \ \mathsf{says}_d\ x\ \mathsf{to}\ p \ \leftarrow\ & A\ \mathsf{knows}_d\ x_1 \wedge \cdots \wedge A\ \mathsf{knows}_d\ x_n \wedge con \wedge \\ & A\ \mathsf{knows}_d\ t_1\ \mathsf{exists}\ \wedge \cdots \wedge A\ \mathsf{knows}_d\ t_k\ \mathsf{exists} \end{aligned}$$

in short

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

Here

- A is a ground principal term, d is zero or infinity (and the infinity subscript is usually skipped);
- $x, x_1, \ldots, x_n$ are fact terms, and con is a substrate constraint, that is a quantifier-free substrate formula;
- all variables are regular, and p 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 principal 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).

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 $A\ \mathsf{knows}_d\ t_i\ \mathsf{exists}$.

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).

3.1.1. **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, 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.

3.1.2. **A-bound variables and regular components.** For any variable v of R different from p, the body of R contains the conjunct $A\ \mathsf{knows}_d\ v\ \mathsf{exists}$. 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 $A\ \mathsf{knows}\ t\ \mathsf{exists}$. In that sense t is A-bound. Thus R leads to the knowledge by A, or to the communication from A to p, of a fact f such that every regular component of f, with a possible exception of p, either is already known to A or else is explicitly named in R by means of a ground term. Notice that, in the case of communication, the sender itself is explicitly named by the ground term A.

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

$$A :_d \ x \text{ to } all \ \leftarrow \ x_1, \dots, x_n, con$$

where all is a "fresh" principal variable that does not appear in $x, x_1, \dots, x_k, con$ 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 term.

3.1.3. **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 requirement that, with a possible exception of p, the variables of R and the non-ground regular components of x be A-bound. The requirement 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 zero, addition, multiplication and the successor function S. By [8], 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 foo be a ground fact.

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

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

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

$$\begin{aligned} &A \text{ knows zero exists} \\ A \text{ knows } S(u) \text{ exists} &\leftarrow A \text{ knows}_d\ u \text{ exists}, \\ A \text{ knows foo} &\leftarrow G(u_1, \dots, u_4) = \text{zero} \wedge \\ &\qquad A \text{ knows } u_1 \text{ exists} \ \wedge \ \cdots \ \wedge \ A \text{ knows } u_4 \text{ exists} \end{aligned}$$

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 fact term ($S(u)$ exists); the non-ground regular component $S(u)$ of the fact term is not A-bound. In both counter-examples knows could be replaced with knows_0.

3.1.4. **Undesirable liberalization.** One may be tempted to liberalize (As2) by replacing A says_d x to p with A says_d x to $t(p)$ where $t(p)$ is a term 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 fact. 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

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

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.

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

3.2.1. **K0∞ house rule:**

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

Internal knowledge is knowledge.

3.2.2. **Say2know house double rule:**

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

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 fact q said$_d$ x, not the fact x itself.

3.2.3. **K & house double rule:**

$$p\ \mathsf{knows}_d\ x\&y \leftarrow (p\ \mathsf{knows}_d\ x) \wedge (p\ \mathsf{knows}_d\ y).$$

The converse will follow form the KMon and E& policy rules.

3.2.4. **KMon house double rule:**

$$p\ \mathsf{knows}_d\ x \leftarrow x \lesssim y \wedge (p\ \mathsf{knows}_d\ y).$$

Knowledge is monotone with respect to the ensue relation.

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

3.2.5. **EOrder house rules:**

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

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

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

3.2.6. **E & house rules:**

$$x \lesssim x\&y,$$
$$y \lesssim x\&y,$$
$$x\&y \lesssim z \leftarrow (x \lesssim z) \wedge (\ y \lesssim z).$$

Thus $x\&y$ is the least upper bound for x and y in the preorder $\lesssim$. It follows that the conjunction is commutative, associative and idempotent with respect to the equivalence relation on facts.

Corollary 3.1. *For any facts x, y, z in the state of knowledge, we have*

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

3.2.7. **Exists house rule:**

$$q \ \mathsf{exists} \ \lesssim x \ \leftarrow \ q \ \mathsf{regcomp} \ x.$$

Here (q regcomp x) is a substrate constraint. To understand the Exists rule, consider an instance of the rule where all variables have been instantiated so that q is a particular principal and x is a particular fact. Then $f = (q \ \mathsf{exists})$ is a particular fact, and facts do not have truth values in DKAL. Think of the fact f in the context of the knowledge of some principal p. The relevant question is not whether f is true but whether p knows this fact, in other words whether p is aware of (the existence of) q. If p knows some fact x with regular component q then p is aware of q. More precisely, given ($p \ \mathsf{knows}_d \ x$) and (q regcomp x), the Exists and KMon rules allows us to derive ($p \ \mathsf{knows}_d \ q$ exists).

3.2.8. **Said0∞ house rule:**

$$p \ \mathsf{said} \ x \ \lesssim \ p \ \mathsf{said}_0 \ x.$$

Saying based on internal knowledge is saying. (See also Lemma 3.5.1 in this connection.)

3.2.9. **SaidMon house double rule:**

$$p \ \mathsf{said}_d \ x \lesssim p \ \mathsf{said}_d \ y \ \leftarrow \ x \lesssim y.$$

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

3.2.10. **Said & house double rule:**

$$p \ \mathsf{said}_d \ x\&y \ \lesssim \ (p \ \mathsf{said}_d \ x) \ \& \ (p \ \mathsf{said}_d \ y).$$

The converse follows from rules SaidMon and E&. Thus said and said_0 distribute over &.

Corollary 3.2. $p \ \mathsf{said}_d \ x\&y \ \sim \ (p \ \mathsf{said}_d \ x) \ \& \ (p \ \mathsf{said}_d \ y)$.

3.2.11. **SelfQuote house double rule:**

$$p \ \mathsf{said}_d \ x \ \lesssim \ p \ \mathsf{said}_d \ p \ \mathsf{said}_d \ x.$$

3.2.12. **Cansay0∞ house rule:**

$$p \ \mathsf{cansay}_0 \ x \ \lesssim \ p \ \mathsf{cansay} \ x.$$

The authority to promulgate a fact on the basis of internal knowledge follows from the authority to promulgate the fact.

3.2.13. **CansaySaid house double rule:**

$$x \lesssim (p\ \textsf{cansay}_d\ x)\ \&\ (p\ \textsf{said}_d\ x).$$

A principal p with the authority to promulgate x, promulgates x by saying it; that double rule is basically inheritted from SecPAL. Note that the subscript of said should match that of cansay. For, consider the stronger rule $x \lesssim (B\ \textsf{cansay}_0\ x)\ \&\ (p\ \textsf{said}\ x)$ and suppose A knows B $\textsf{cansay}_0$ 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 $B : x$ to $A \leftarrow B$ knows C said x. If C places an assertion $C : x$ to B, we have B says x to A by B's assertion, hence we have A knows B said x by Say2know, and therefore we have A knows x by CansaySaid.

3.2.14. **Del house double rule:**

$$p\ \textsf{cansay}\ q\ \textsf{cansay}_d\ x \lesssim (p\ \textsf{cansay}\ x)\ \&\ (q\ \textsf{exists}).$$

This is the delegation double rule. The authority to promulgate x includes the authority to delegate the authority to promulgate x.

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

$$p\ \textsf{cansay}_d\ x \lesssim p\ \textsf{cansay}_d\ p\ \textsf{cansay}_d\ x.$$

3.2.16. **Role house rules:**

$$p\ \textit{attribute} \lesssim (q\ \textit{attribute})\ \&\ (p\ \textsf{canActAs}\ q),$$
$$q\ \textit{speech} \lesssim (p\ \textit{speech})\ \&\ (p\ \textsf{canSpeakAs}\ q).$$

Here *attribute* and *speech* are variables of types Attribute and Speech respectively. The functions can act as and can speak as 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."

3.3. **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.

3.3.1. **Subrecursions.**

Proposition 3.3.
- *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 facts f such that*

$$p\ \textsf{knows}_0\ f\ \textit{holds in the state of knowledge}$$

and the set of elements b such that

$$p\ \textsf{knows}_0\ b\ \textsf{exists}\ \textit{holds in the state of knowledge}$$

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 $\textsf{says}_0$ f 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, $\textsf{knows}_0$ and $\textsf{saysto}_0$. □

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

Lemma 3.4. *The following formulas are universally true in the state of knowledge. The subscript d could be 0 or ∞.*

(1) $(p \ \mathsf{knows}\ x) \wedge (q \ \mathsf{regcomp}\ x) \to p \ \mathsf{knows}\ q \ \mathsf{exists}$.
(2) $(p \ \mathsf{knows}\ q \ \mathsf{cansay}_d\ x) \wedge (p \ \mathsf{knows}\ q \ \mathsf{said}_d\ x) \to p \ \mathsf{knows}\ x$.
(3) $q_2 \ \mathsf{cansay}_d\ x \ \lesssim\ (q_1 \ \mathsf{cansay}_d\ x)\ \&\ (q_1 \ \mathsf{said}\ q_2 \ \mathsf{cansay}_d\ x)$,
$(p \ \mathsf{knows}\ q_1 \ \mathsf{cansay}_d\ x) \wedge (p \ \mathsf{knows}\ q_1 \ \mathsf{said}\ q_2 \ \mathsf{cansay}_d\ x) \to p \ \mathsf{knows}\ q_2 \ \mathsf{cansay}_d\ x$.
(4) $p \ \mathsf{canActAs}\ r \ \lesssim\ (p \ \mathsf{canActAs}\ q)\&(q \ \mathsf{canActAs}\ r)$.

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

2. Suppose that p knows facts $q \ \mathsf{cansay}_d\ x$ and $q \ \mathsf{said}_d\ x$. By the appropriate K& house rule, p knows the fact $f = (q \ \mathsf{cansay}_d\ x)\&(q \ \mathsf{said}_d\ x)$. By the CansaySaid house rule, $x \lesssim f$. By the KMon house rule, $p \ \mathsf{knows}\ x$.

3. It suffices to prove only the first formula because the second formula follows from the first by the appropriate KMon house rule. Let $f = (q_1 \ \mathsf{cansay}\ x)\ \&\ (q_1 \ \mathsf{said}\ q_2 \ \mathsf{cansay}\ x)$. By 1 and the appropriate Del house rule, we have $q_1 \ \mathsf{cansay}\ q_2 \ \mathsf{cansay}\ x \ \lesssim\ f$. Now apply the appropriate CansaySaid house rule.

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

3.3.3. **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 3.5. *The following rules, where $d \in \{0, \infty\}$, are redundant.*

(1) $p \ \mathsf{says}\ x \ \mathsf{to}\ q \leftarrow p \ \mathsf{says}_0\ x \ \mathsf{to}\ q$.
(2) $p \ \mathsf{says}_d\ x\&y \ \mathsf{to}\ q \leftarrow (p \ \mathsf{says}_d\ x \ \mathsf{to}\ q) \wedge (p \ \mathsf{says}_d\ y \ \mathsf{to}\ q)$.
(3) $p \ \mathsf{says}_d\ x \ \mathsf{to}\ q \leftarrow x \lesssim y \wedge (p \ \mathsf{says}_d\ y \ \mathsf{to}\ q)$.

Proof. We start with an obvious claim: For each d, there are no assertions with saysto_d in the premise, and there is only one house rule where relation 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 $p \ \mathsf{says}\ z \ \mathsf{to}\ q$ can be used: to derive a fact $q \ \mathsf{knows}\ p \ \mathsf{said}\ z$. So it suffices to prove an implication

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

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

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

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

Suppose $(p \ \mathsf{says}_d\ x \ \mathsf{to}\ q)$ and $(p \ \mathsf{says}_d\ y \ \mathsf{to}\ q)$. Then $(p \ \mathsf{says}_d\ x \ \mathsf{to}\ q)$ and $(p \ \mathsf{says}_d\ y \ \mathsf{to}\ q)$. By Say2know, $q \ \mathsf{knows}\ p \ \mathsf{said}_d\ x$ and $q \ \mathsf{knows}\ p \ \mathsf{said}_d\ y$. By K&, $q \ \mathsf{knows}\ \big((p \ \mathsf{said}_d\ x)\ \&\ (p \ \mathsf{said}_d\ y)\big)$. By KMon and Said&, $q \ \mathsf{knows}\ p \ \mathsf{said}_d\ x\&y$.

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

$$x \lesssim y \wedge (p \ \mathsf{says}\ y \ \mathsf{to}\ q) \to q \ \mathsf{knows}\ p \ \mathsf{said}\ x.$$

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

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

$\mathsf{Vacuous} \lesssim \mathsf{x}.$

Any principal that knows anything would know the Vacuous fact.

By Lemma 3.4.4, the canActAs relation happens to be transitive. We also have

$$r\ \mathsf{says\ foo} \le (p\ \mathsf{says\ foo})\&(p\ \mathsf{canSpeakAs}\ q)\&(q\ \mathsf{canSpeakAs}\ r)$$

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 $p\ \mathsf{knows}\ p\ \mathsf{exists}$. But the effect of the Cartesian rule is achieved by a single assertion

A: $(p\ \mathsf{exists})\ \mathsf{to}\ p$

where A could be e.g. the system itself. By Say2know, it follows that $p\ \mathsf{knows}\ p\ \mathsf{exists}$ for every principal p.

We considered rules

(1) $p\ \mathsf{cansay}_d\ x \lesssim p\ \mathsf{cansay}_d\ x\&y$,
(2) $p\ \mathsf{cansay}_d\ x\&y \lesssim (p\ \mathsf{cansay}_d\ x)\&(p\ \mathsf{cansay}_d\ y)$

as candidates for house rules but did not endorse them. Rule 1 asserts that cansay_d is monotone. But this is not necessarily so. For example a low-level administrator may be allowed to give 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 4.7.

3.4. **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 5 where the two theorems are also proved. Then we generalize the notion of basic queries and prove that the two theorems remain valid.

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

- p is a ground principal term in the substrate vocabulary,
- $v_1, \ldots, v_k$ are variables of regular types,
- t is a fact term 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

$Q = (p\ \mathsf{knows}_d\ t(v_1,\dots,v_k))$ under authorization policy $\mathcal{A}$ is denoted $\mathrm{ans}_{\mathcal{A}}(Q)$. It is the set of tuples $\langle b_1,\dots,b_k\rangle$ of substrate elements such that the type of b_i is that of v_i and

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

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 $b_1,\dots,b_k$ to the variables $v_1,\dots,v_k$ respectively.

If v_i is a regular component of $t(v_1,\dots,v_k)$ then the requirement $p\ \mathsf{knows}_d\ b_i\ \mathsf{exists}$ is superfluous as it follows from $p\ \mathsf{knows}_d\ t(b_1,\dots,b_k)$ 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 term $t(\mathit{file}) = (\mathsf{master}(\mathit{file})\ \mathsf{cansay\ foo})$ where *file* is a regular variable and foo is a ground fact term. Term $\mathsf{master}(\mathit{file})$ is a regular component of $t(\mathit{file})$ but the variable *file* is not. And the answer to a query $p\ \mathsf{knows}\ t(\mathit{file})$ may be infeasibly large, even infinite, if we require that it contains all files whose master can say 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`.

3.4.2. **Computing basic queries.** In Section 5, 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 terms 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 term or formula. The *length* of an authorization policy is the sum of the lengths of its assertions.

Theorem 3.6. *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 5.

Theorem 3.7. *The time that our algorithm needs to answer a basic query Q under an authorization policy $\mathcal{A}$ is bounded by a polynomial in $\big(\mathrm{length}(\mathcal{A}) + \mathrm{length}(Q)\big)^{\delta+w}$, where δ bounds the depth to which* said *and* said_d *(possibly mixed) are nested in the policy and query term, and w bounds the number of free variables allowed in assertions and in the query term. In particular, assuming a fixed bound on the nesting depth of* said_d *and on the assertion and query width, the computation time of the answering algorithm is polynomial in* $\mathrm{length}(\mathcal{A}) + \mathrm{length}(Q)$.

The theorems are proved in Section 5.

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

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

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

Consider for example a query $v\ \mathsf{knows}\ t$ where v is a variable and t has no variables and thus evaluates to a fact in X. The answer to the query would be the list of all principals that know the fact. If the fact is broadcasted then the list includes all principals in X which may well be 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 $Q = (p\ \mathsf{knows}\ t(v))$. 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.

3.4.4. **Single-principal-centric queries.** Let's fix an arbitrary principal term 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 $q\ \mathsf{knows}_d\ t(v_1, \dots, v_k)$ is *p-centric* if the term q is literally the term p. A more general definition of p-centric queries is needed.

Remark 3.8. The requirement that term q is literally term 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

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

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

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

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 $q\ \mathsf{knows}\ t$ 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 $\varphi(b_1, \dots, b_k)$ 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 $p\ \mathsf{knows}_d\ t$, 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 $\varphi(v_1, \dots, v_k)$ under an authorization policy $\mathcal{A}$ is the p-bounded answer mentioned above:

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

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

Lemma 3.9. *For any p-centric queries Q and R, we have*

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

The proof is obvious.

3.4.5. **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

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

We have

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

Lemma 3.10. *There is a polynomial time algorithm that,*

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

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

Proof. We explain how to compute $\operatorname{ans}_{\mathcal{A}}(Q)$ 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 $\operatorname{ans}_{\mathcal{A}}(\varphi)$ and $\operatorname{env}_{\mathcal{A}}(\varphi)$ 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: $\operatorname{ans}(Q) = \operatorname{env}(R) - \operatorname{ans}(R)$, and $\operatorname{env}(Q) = \operatorname{env}(R)$.

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, $\operatorname{ans}_{\mathcal{A}}(Q_1(u_1, \ldots, u_j, v_1, \ldots, v_k))$ is a table with $j + k$ columns. Similarly, $\operatorname{ans}_{\mathcal{A}}(Q_2(v_1, \ldots, v_k, w_1, \ldots, w_\ell))$ 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 $\operatorname{ans}_{\mathcal{A}}(Q)$. Similarly $\operatorname{env}(Q)$ is the join of $\operatorname{env}(R_1)$ and $\operatorname{env}(R_2)$.

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, $\operatorname{ans}(Q)$ is the projection

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

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

Theorem 3.11. *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 $\operatorname{ans}(Q)$.*

Proof. The desired algorithm is the algorithm of Lemma 3.10 that uses the substrate evaluation algorithm Eval and the basic query evaluation algorithm of Theorem 3.6 to compute the givens of Lemma 3.10. Let Q be a p-centric query, and $\varphi(v_1,\dots,v_k)$ be an atomic subformula of Q. Since

$$\text{env}_{\mathcal{A}}(\varphi(v_1,\dots,v_k)) = \text{ans}_{\mathcal{A}}(p \textsf{ knows } (v_1 \textsf{ exists } \ \& \ \cdots \ v_k \textsf{ exists})),$$

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

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

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

Theorem 3.12. *The time bound of Theorem 3.7 remains valid for our algorithm for answering single-principal-centric queries. In other words, Theorem 3.7 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 $\text{length}(Q)$. According to the previous proof, the number of calls to the basic query evaluation algorithm is $O(\text{length}(Q))$, 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. □

4. Examples

We give a few examples illustrating some features of DKAL. Later examples may build on the previous ones. We say that a principal p knows of q if p knows the fact q exists.

Example 4.1 (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 fact 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 cansay r canRead *file* $\leftarrow$ owner(*file*) $= p$.

Recall that we write assertions in an abbreviated form §3.1. 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 cansay 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 §3.2. 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 cansay 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 3.4.1, 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 cansay 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 cansay Cathy canRead Alice/Recipe.

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

6. RR knows Alice cansay Bob cansay Cathy canRead Alice/Recipe.

From 6 and 4′, by Lemma 3.4.2, we have

7. RR knows Bob cansay 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 4.2 (Post-authentication)**.** When a 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, and 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 exressed by assertions

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

Example 4.3 (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 A1S requires only the password authentication for reading, while it requires the password and the smartcard authentication for writing. (This is a system security requirement.) 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.

Example 4.4 (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 cansay q canRead record(p).

It's 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 4.5 (Internal knowledge acquired a priori)**.** A principal called Guest owns no files but other principals may permit Guest to read some of their files. Employees of a partner company Parcom are given guest privileges provided that they have a proper confirmation from Parcom.

1. RR: p canActAs Guest $\leftarrow$ Parcom said p is a Parcom employee.

The authentication server A1S is responsible for validating the confirmation. The server cannot delegate the trust to others.

2. RR: A1S cansay$_0$ Parcom said p is a Parcom employee.

Now suppose that A1S validated Parcom's confirmation that Alice is a Parcom employee. Note that the validation procedure is not visible to our system. As far as the system is concerned, the server's speech is based on its internal knowledge.

3. A1S:$_0$ (Parcom said Alice is a Parcom employee) to RR.

From 3, by the appropriate Say2know rule, we have

3′. RR knows A1S said$_0$ Parcom said Alice is a Parcom employee.

It follows that RR knows of Alice. This gives us an appropriate instance of assertion 2:

2′. RR knows A1S cansay$_0$ Parcom said Alice is a Parcom employee.

From 2′ and 3′, by Lemma 3.4.2, we have

4. RR knows Parcom said Alice is a Parcom employee.

Finally, from 4 and (the appropriate instance of) 1, we have

5. RR knows Alice canActAs Guest.

Example 4.6 (A user centric scenario)**.** Alice finds an article on Charlie's website. Alice writes to Charlie, and he allows her to download the article. But Charlie does not have the copyright. The article is published by the Best publishing house which has the copyright. Alice writes to Best, and Best replies to her that Charlie can allow the download.

Alice uses DKAL to verify that she is allowed to download the article. She places the following assertions where OK stands for the fact that she has the right to download that article.

1. Alice: Best cansay OK.
2. Best: (Charlie cansay OK) to Alice
3. Charlie: OK to Alice.

It follows that Alice knows OK (and thus can go ahead and download the article). The key in the derivation is an application of the Del house rule:

$$\textsf{Best cansay Charlie cansay OK} \lesssim (\textsf{Best cansay OK}) \ \& \ (\textsf{Charlie exists}).$$

By 1, Alice knows that Best can say OK, and of course she knows of Charlie (e.g. from 3). So she knows that Best can say that Charlie can say OK. Taking into account assertion 2, she knows that Charlie can say OK. Taking into account assertion 3, she knows OK.

Example 4.7 ($\textsf{cansay}\ x\&y \not\lesssim (\ \textsf{cansay}\ x)\ \&\ (\ \textsf{cansay}\ y)$)**.** A composer Alice produced two CD's, which we imaginatively call CD1 and CD2, to be sold by shops. A shop pays a fee for the right to sell a CD. (The shop also pays the composer a portion of the proceeds but that is irrelevant for us here.) Accordingly an attribute $\textsf{canSell}(v_1, v_2)$ takes two arguments: a CD and the fee amount. Bob is Alice's agent. Alice could couple the two CDs by asserting only that Bob

1. cansay (shop canSell(CD1,$100) & shop canSell(CD2,$100)).

Then Bob would be unable allow a shop to sell just one of the two CDs. But Alice does not want the CDs to be coupled:

2. Alice: Bob cansay (shop canSell(CD1,$100)) &
   Bob cansay (shop canSell(CD2,$100))

Bob may be flexible if he wants. He can allow a shop to sell only CD1 for example. All he needs to do is to say to Alice that a shop can sell CD1 for a $100 fee.

Now, Bob wants to delegate sell authorization to Charlie. He should not and does not have an option to couple the two CDs and delegate to Charlie only the right 1. But the rejected house rule

$$\textsf{cansay}\ x\&y \lesssim (\ \textsf{cansay}\ x)\ \&\ (\ \textsf{cansay}\ y)$$

would give him just such an option. Indeed, using the rejected rule, one can derive

3. Alice knows Bob cansay (shop canSell(CD1,$100) & shop canSell(CD2,$100)).

from 2. Now, using Del, Bob can delegate the right 1 to Charlie.

5. Answering basic queries

In this section, we describe an algorithm for answering basic queries. As in the appendix, substructures are in general partial.

Let X be a substrate structure, and let $\mathcal{A}$ be an authorization policy. Fix a basic query $Q = (a\ \textsf{knows}_d\ t(v_1, \ldots, v_k))$ where a is a ground principal term and t is a term of type Fact with the k variables as shown. The basic idea behind the algorithm is quite simple:

isolate a *finite* (partial) substructure $\overset{\circ}{X}$ of the substrate X which is rich enough so that the computation of the answer can be done by means of a logic program $\bar{\Pi}$ over $\overset{\circ}{X}$. As $\overset{\circ}{X}$ is finite, the fixed point for $\bar{\Pi}$ can be computed, see Theorem A.4.

The finite substructure $\overset{\circ}{X}$ must satisfy the following conditions.

(1) Consider any assertion in $\mathcal{A}$. It has the form (As1) or (As2) in §3.1. $\overset{\circ}{X}$ contains the principal $\mathrm{Val}_X(A)$ that placed the assertion as well as the values in X of all ground regular components of the fact x in the head of the assertion. In addition, $\overset{\circ}{X}$ contains the value of the term a in the query Q.
(2) Consider any term τ that occurs in $\mathcal{A}$ or occurs in Q or has the form A said_d x such that $\mathcal{A}$ has an assertion with the head A says_d x to p. Consider any assignment ξ of values (of appropriate types) obtained in 1 to the variables, if any, of τ. The value of τ in X under the assignment ξ belongs to $\overset{\circ}{X}$ or is a conjuction of elements in $\overset{\circ}{X}$.
(3) The restriction of the ensue relation to $\overset{\circ}{X}$ can be computed over $\overset{\circ}{X}$.

We can satisfy the first two conditions by putting the relevant elements into $\overset{\circ}{X}$. Since $\mathcal{A}$ is finite, the resulting set $\overset{\circ}{X}$ is finite too.

Condition 3 is more difficult to satisfy. We cannot close $\overset{\circ}{X}$ under ensue consequences; that may lead to an infinite set due to the Del rule. And we cannot ignore consequences which do not belong to $\overset{\circ}{X}$ as the following example demonstrates. Suppose that facts

(a) p cansay r cansay foo,
(b) p said q cansay foo, and
(c) q cansay r cansay foo,

belong to $\overset{\circ}{X}$ but facts

(d) p cansay q cansay r cansay foo, and
(e) p said q cansay r cansay foo

do not. Note that (d) ensues (a) by Del, (e) ensues (b) by Del and SaidMon, and (c) ensues (d) & (e) by CansaySaid. Hence (c) ensues (a) & (b), which may not be provable over $\overset{\circ}{X}$ that does not contain (d) and (e).

Our solution is not to increase $\overset{\circ}{X}$, as this may force us to deal with an infinite model, but add a rule to the ensue program used over $\overset{\circ}{X}$ that brings the missing ensue relationships into the fixed point. The rule we add is identified in §5.1 as (P-Superfluous). The example described above is taken care of by the third subrule of (P-Superfluous). The other two subrules take care of similar but simpler problems, relieving us of the need to ensure that p said f belongs to $\overset{\circ}{X}$ whenever p cansay f belongs to $\overset{\circ}{X}$, and vice versa.

It turns out that the addition of the superfluous rule allows computing the restriction of the ensue relation to $\overset{\circ}{X}$ using a logic program over $\overset{\circ}{X}$. We prove this in §5.1. Then in §5.2 we show how to use this in computing the answer to a basic query. Our algorithm proceeds as follows: First, compute the relevant finite substrate $\overset{\circ}{X}$. Second, create a logic program $\bar{\Pi}$ over $\overset{\circ}{X}$ that incorporates all the assertions of the given policy, the house rules Say2know and K0∞, and the knowledge consequences of the ensue house rules (for example the rules p knows q said foo $\leftarrow$ p knows q said_0 foo for all facts foo in $\overset{\circ}{X}$; these rules are the knowledge consequences of the ensue rule Said0∞), including also the knowledge consequences of the superfluous rule. $\bar{\Pi}$ does not include K&; we separate each fact into a conjunction of facts that we call canonical, and compute only knowledge of canonical facts. Third, compute the fixed point $\bar{\Pi}(\mathring{X})$. This is typically the most intensive part computationally. It can be done using Theorem A.4, as $\mathring{X}$ is finite. Fourth, go over all instances of the query term using regular elements in $\mathring{X}$ to instantiate its variables, and check for each instance whether all of its canonical components hold in $\bar{\Pi}(\mathring{X})$. If yes, then write down the variable assignment leading to this instance of the query term. We prove that the set of these variable assignments is the answer to the basic query.

5.1. **Downward closure under ensue.** The results here will allow us to compute enough of the ensue relation to answer queries about knowledge in a superstrate for a given authorization policy.

A fact (that is a fact term) is *canonical* if it has the form $p_1\ \mathsf{said}_{d_1}\ p_2\ \mathsf{said}_{d_2}\ \ldots\ p_k\ \mathsf{said}_{d_k}$ *q attribute*. The number k, which may be zero, is the *quotation depth* of the fact. We refer to $p_1\ \mathsf{said}_{d_1}\ \ldots\ p_i\ \mathsf{said}_{d_i}$ for $i \le k$ as *prefixes* of the fact, and in general refer to sequences of the form $p_1\ \mathsf{said}_{d_1}\ \ldots\ p_i\ \mathsf{said}_{d_i}$ as *prefixes*.

Claim 5.1. *Every fact f is equivalent to a conjunction of canonical facts.*

Proof. Facts are produced only by means of the function fact of type $\big((\text{Principal} \times \text{Speech}) \cup (\text{Regular} \times \text{Attribute})\big) \to \text{Fact}$ and the function & of type Fact × Fact → Fact. The proof is by induction on fact f. The only non-trivial case is when f is of the form $p\ \mathsf{said}_d\ g$ and g is a conjunction of canonical facts g_i. By Corollary 3.2, said and said_0 distribute over &. So f is equivalent to the conjuction of canonical facts $p\ \mathsf{said}_d\ g_i$. □

Note that the proof is constructive and gives rise to a simple procedure for converting a given fact f into an equivalent conjunction of canonical facts.

In the rest of this subsection we describe an algorithm to decide whether a canonical fact y ensues the conjunction $\&\,Z$ of a set Z of canonical facts. Fix a set Z of canonical facts.

We begin by describing rules which allow us to deduce all canonical facts that ensue $\&\,Z$. The rules paraphrase the ensue house rules in §3.2, with two changes: we absorb the SaidMon double rule into the other rules; and we distribute said_d over conjunctions, pulling conjunctions out whenever possible, so that the rules deal only with canonical facts. The list of rules is as follows. In the list, *pref* ranges over prefixes, f and $f_1, \ldots, f_n$ range over facts, and p and q range over regular elements (of just the type Principal when appearing before a speech).

(P-Exists) If $q\ \mathsf{regcomp}\ f$, then $\mathit{pref}\ q\ \mathsf{exists} \lesssim \mathit{pref}\ f$.

(P-Said0∞) $\mathit{pref}\ p\ \mathsf{said}\ f \lesssim \mathit{pref}\ p\ \mathsf{said}_0\ f$.

(P-SelfQuote) $\mathit{pref}\ p\ \mathsf{said}_d\ f \lesssim \mathit{pref}\ p\ \mathsf{said}_d\ p\ \mathsf{said}_d\ f$.

(P-Cansay0∞) $\mathit{pref}\ p\ \mathsf{cansay}_0\ f \lesssim \mathit{pref}\ p\ \mathsf{cansay}\ f$.

(P-CansaySaid) For each $f \in \{f_1, \ldots, f_n\}$: $\mathit{pref}\ f \lesssim \&\{\mathit{pref}\ p\ \mathsf{cansay}_d\ f_1 \& \ldots \& f_n,\ \mathit{pref}\ p\ \mathsf{said}_d\ f_1, \ldots, \mathit{pref}\ p\ \mathsf{said}_d\ f_n\}$.

(P-Del) $\mathit{pref}\ p\ \mathsf{cansay}\ q\ \mathsf{cansay}_d\ f \lesssim \&\{\mathit{pref}\ p\ \mathsf{cansay}\ f,\ \mathit{pref}\ q\ \mathsf{exists}\}$.

(P-Del$^-$) $\mathit{pref}\ p\ \mathsf{cansay}_d\ f \lesssim \mathit{pref}\ p\ \mathsf{cansay}_d\ p\ \mathsf{cansay}_d\ f$.

(P-Role) *pref* p *attribute* $\lesssim$ &$\{$*pref* q *attribute*, *pref* p $\mathsf{canActAs}$ $q\}$. Similarly, *pref* q *speech* $\lesssim$ &$\{$*pref* p *speech*, *pref* p $\mathsf{canSpeakAs}$ $q\}$.

Definition 5.2. An *ensue-deduction* from a set X of canonical facts is a sequence of canonical facts $x_1, \dots, x_r$ so that for each $i \le r$, either $x_i \in X$, or $x_i \lesssim \&\{x_{j_1}, \dots, x_{j_n}\}$ is an instance of one of the above rules, with $j_1, \dots, j_n < i$. An *ensue-deduction of* y is an ensue-deduction $x_1, \dots, x_r$ so that $x_r = y$. And y is *ensue-deducible* from X if there is an ensue-deduction of y from X.

In this subsection we only deal with ensue-deductions, and so we will refer to them simply as deductions.

Lemma 5.3. *A canonical fact* y *ensues* & Z *(according to the ensue house rules in Section 3) if and only if* y *is deducible from* Z *(according to the previous definition).*

Proof. The right-to-left direction is straightforward by induction on the length of the given deduction. We prove the left-to-right direction.

By the previous claim, any fact y is equivalent to a conjuction of canonical facts, the *canonical components* of y. Let R be the following relation on facts: $y \; R \; z$ if and only if every canonical component of y is deducible from the set of canonical components of z. It suffices to show that R is a fixed point for the ensue house rules of Section 3. Indeed suppose that R is a fixed point. Since $\lesssim$ is the least fixed point, it is a subset of R. If a canonical fact $y \lesssim \& Z$ then $y \; R \; \& Z$ and therefore y is deducible from Z.

R is a fixed point for EOrder since deductions can be composed. R is by definition a fixed point for E& and for Said&. Each of the remaining ensue rules except Said-Mon have exact counterparts among the (P) rules, and from this it follows directly that R is a fixed point for these rules. Thus it remains to show that R is a fixed point for SaidMon. To this end, we assume that $y \; R \; z$ and we show that $p \; \mathsf{said}_d \; y \; R \; p \; \mathsf{said}_d \; z$. Let $y_1, \dots, y_m$ be the canonical components of y, and $z_1, \dots, z_n$ be the canonical components of z. Then $p \; \mathsf{said}_d \; y_1, \dots, p \; \mathsf{said}_d \; y_m$ are the canonical components of $p \; \mathsf{said}_d \; y$, and $p \; \mathsf{said}_d \; z_1, \dots, p \; \mathsf{said}_d \; z_n$ are the canonical components of $p \; \mathsf{said}_d \; z$. We have to show that every $p \; \mathsf{said}_d \; y_i$ is deducible from $\{p \; \mathsf{said}_d \; z_1, \dots, \; \mathsf{said}_d \; z_n\}$. By the assumption, there is a deduction $x_1, \dots, x_r$ of y_i from $\{z_1, \dots, z_n\}$. Then the sequence $p \; \mathsf{said}_d \; x_1, \dots, p \; \mathsf{said}_d \; x_r$ is a deduction of $p \; \mathsf{said}_d \; y_i$ from $\{p \; \mathsf{said}_d \; z_1, \dots, \; \mathsf{said}_d \; z_n\}$. □

Definition 5.4. A deduction is *simple* if it only uses the rules (P-Del) and (P-Cansay0∞). And y is *simply deducible* from X if it can be obtained from X by a simple deduction.

Definition 5.5. A term x is an H*-germ* for a term y if either $x = y$ or else

- x has the form $p \; \mathsf{cansay} \; f$,
- y has the form $p \; \mathsf{cansay}_{d_0} \; q_1 \; \mathsf{cansay}_{d_1} \dots q_n \; \mathsf{cansay}_{d_n} \; f$, and
- $q_1, \dots, q_n \in H$.

In the case $x \neq y$, we say that x is a *strict* germ for y and that $q_1, \dots, q_n$ (and no other principals) are *essential* for obtaining y from x.

We write *pref* H exist to abbreviate the set of facts *pref* q exists, $q \in H$.

Claim 5.6. *A fact pref* y *is simply deducible from a set* X *of facts if and only if there exist a fact* x *and a set* H *so that*

- x *is an* H*-germ for* y*, and*

- $\{pref\ x,\ pref\ H\ \mathsf{exist}\} \subset X$.

Proof. The right-to-left direction is immediate. For the left-to-right direction, let X' be the set of facts *pref* y such that there exist x and H satisfying the two conditions. $X \cup X'$ contains all the facts in X and is closed under the rules (P-Del) and (P-Cansay0∞). □

We intend to show that if y is deducible from Z, then there is a deduction of y that can be broken into two parts: a "conservative" part which avoids uses of (P-Del), followed by a simple part. But first we define exactly what we mean by a conservative deduction. We introduce the following superfluous rule, superfluous in the sense that it is a consequence of the rules given above, and therefore does not produce new deducible facts. The point is that the superfluous rule compensates, at least partly, for the loss of (P-Del) in conservative deductions.
(P-Superfluous) consists of the following subrules.

(1) If x is an H-germ for $p\ \mathsf{cansay}_d\ f$, then $pref\ f \lesssim \&\{pref\ p\ \mathsf{said}_d\ f,\ pref\ x,\ pref\ H\ \mathsf{exist}\}$.
(2) If x_i is an H-germ for f_i, then $pref\ f_i \lesssim \&\{pref\ p\ \mathsf{cansay}_d\ f_1 \& \ldots \& f_n,\ pref\ p\ \mathsf{said}_d\ x_1, \ldots,\ pref\ p\ \mathsf{said}_d\ x_n,\ pref\ p\ \mathsf{said}_d\ H\ \mathsf{exist}\}$.
(3) If $q\ \mathsf{cansay}\ \bar{g}$ is a R-germ for $q\ \mathsf{cansay}\ \bar{x}$, then $pref\ q\ \mathsf{cansay}\ \bar{x} \lesssim \&\{pref\ p\ \mathsf{cansay}\ \bar{x},\ pref\ p\ \mathsf{said}_d\ q\ \mathsf{cansay}\ \bar{g},\ pref\ p\ \mathsf{said}_d\ R\ \mathsf{exist}\}$.

It is easy to check that the first two subrules follow from the previous rules. For example in the case of subrule (1), $pref\ p\ \mathsf{cansay}_d\ f$ ensues $(pref\ x)\ \&\ (pref\ H\ \mathsf{exist})$ by Claim 5.6, and $pref\ f$ ensues $(pref\ p\ \mathsf{cansay}_d\ f)\ \&\ (pref\ p\ \mathsf{said}_d\ f)$ by (P-CansaySaid). The case of subrule (2) is similar. As for subrule (3):

(1) $pref\ p\ \mathsf{said}_d\ q\ \mathsf{cansay}\ \bar{x}$ ensues $(pref\ p\ \mathsf{said}_d\ q\ \mathsf{cansay}\ \bar{g})\ \&\ (pref\ p\ \mathsf{said}_d\ R\ \mathsf{exist})$, by Claim 5.6.
(2) $pref\ p\ \mathsf{cansay}_d\ q\ \mathsf{cansay}\ \bar{x}$ ensues $(pref\ p\ \mathsf{cansay}\ \bar{x})\ \&\ (pref\ q\ \mathsf{exists})$ by (P-Cansay0∞) and (P-Del). (Note that $pref\ q$ exists ensues $pref\ p\ \mathsf{said}_d\ q\ \mathsf{cansay}\ \bar{g}$.)
(3) Finally, $pref\ q\ \mathsf{cansay}\ \bar{x}$ ensues the facts obtained in (1) and (2), by (P-CansaySaid).

Definition 5.7. A *conservative deduction* is one which uses any of the (P) rules, including (P-Superfluous), except (P-Del). x is *conservatively deducible* from Z if it can be obtained in a conservative deduction from Z.

Definition 5.8. A deduction of y from Z is *normal* if it has the following form: A conservative deduction from Z of all facts in a set X, followed by a simple deduction of y from X. y is *normally deducible* from Z if there is a normal deduction of y from Z.

Remark 5.9. By Claim 5.6, a fact $pref\ y$ is normally deducible from Z if and only if there is a fact x and a set H so that:

- $pref\ x$ and $pref\ H\ \mathsf{exist}$ are conservatively deducible from Z, and
- x is an H-germ for y.

The lemmas below show that: (a) there is an algorithm that determines whether y is simply deducible from X; (b) there are only finitely many facts that can be obtained from Z by conservative deductions, with a computable from Z bound on the number of these facts, and hence there is an algorithm that produces them all; (c) every fact which is deducible from Z is normally deducible from Z. The combination of these results lets us compute ensue, as we shall see below. It is in part (c) that we require the superfluous rule, as a partial replacement for the loss of (P-Del) in conservative deductions. Notice that (P-Del) must be given up in order for (b) to hold, since its iterations clearly produce infinitely many facts.

Lemma 5.10. *There is an algorithm that, given y and X, determines whether y is simply deducible from X.*

Proof. Immediate from Claim 5.6. □

Lemma 5.11. *Suppose that y ensues & Z. Then y is normally deducible from Z.*

The converse of Lemma 5.11 is clear, as a normal deduction of y is a deduction of y.

Proof of Lemma 5.11. Suppose y ensues & Z. Then there is a deduction $x_1, \dots, x_r$ of y from Z using the (P) rules other than (P-Superfluous). We work by induction on the length r of this deduction.

If y is an element of Z then the lemma holds trivially. Suppose then that $y \notin Z$ and in particular $r > 1$.

We consider cases, depending on the rule used to deduce $y = x_r$ from the previous facts in the list.

Suppose to begin with that y is obtained through a use of (P-Del), on facts x and *pref* q exists say. By induction there are normal deductions of x and of *pref* q exists. The normal deduction of *pref* q exists must be conservative, in other words its simple part must be trivial, since a non-trivial simple part would result in a fact of the form *pref* q cansay_d f, not *pref* q exists. Combining this conservative deduction with the normal deduction of x it follows that there is a normal deduction $\mathcal{D}$ of x whose conservative part deduces also *pref* q exists. Appending a use of (P-Del) to the simple deduction ending $\mathcal{D}$ we obtain a normal deduction of y.

A similar (but even simpler) argument handles the case that y is obtained through a use of (P-Cansay0∞), as (P-Cansay0∞), like (P-Del) above, is allowed in simple deductions.

Suppose next that y is obtained through a use of (P-Said0∞). Then y has the form *pref* p said f, and by induction there is a normal deduction of *pref* p said_0 f. By Remark 5.9 there is a conservative deduction of facts *pref* p said_0 $\bar{f}$ and *pref* p said_0 H exist, where $\bar{f}$ is an H-germ of f. Appending uses of rule (P-Said0∞) to this conservative deduction we see that there is a conservative deduction of the facts *pref* p said $\bar{f}$ and *pref* p said H exist. (Note the change from said_0 to said.) By Remark 5.9 it follows that *pref* p said f is normally deducible.

An argument similar to the above handles the cases of the rules (P-SelfQuote) and (P-Role). The argument is trivial in many instances. For example the argument for the attribute case of (P-Role) is trivial unless the attribute begins with cansay.

Suppose that y is obtained through a use of (P-Del^{-}). So y has the form *pref* p cansay_d f, and by induction *pref* p cansay_d p cansay_d f is normally deducible, using a deduction $\mathcal{D}$ say. If the simple part of $\mathcal{D}$ does not use (P-Del), then *pref* p cansay_d p cansay_d f is conservatively deducible, and appending a use of (P-Del^{-}) to the conservative deduction we see that y too is conservatively deducible and therefore certainly normally deducible. If the simple part does use (P-Del), then *pref* p cansay f must occur in the simple part of $\mathcal{D}$, and so this simple part can be modified to produce *pref* p cansay_d f.

Suppose that y is obtained through a use of (P-Exists). Then y has the form *pref* q exists, and there is f so that y regcomp f and *pref* f is normally deducible. By Remark 5.9 there is a set H and an H-germ g for f so that *pref* g and *pref* H exist are conservatively deducible. Since q is a regular component of f, it must either be a regular component of g or belong to H. Either way it follows that *pref* q exists is conservatively deducible.

We reach now the final, and main, case, the case that y is obtained through a use of (P-CansaySaid). It is in this case that we use (P-Superfluous). The case consists of three subcases.

First, suppose that (P-CansaySaid) is used with $n \geq 2$. By induction, and noting that a fact of the form *pref* p cansay_d $f_1 \& \ldots \& f_n$ for $n \geq 2$ can only be simply deduced from itself, we see that y has the form *pref* f_i, and there is a conservative deduction of facts:

- *pref* p cansay_d $f_1 \& \ldots \& f_n$,
- *pref* p said_d g_i for $i \leq n$,
- *pref* p said_d H exist

where for each $j \leq n$, g_j is an H-germ for f_j. Appending a use of subrule (2) of (P-superfluous) to this conservative deduction, we see that $y =$ *pref* f_i is conservatively deducible from Z.

Suppose next that (P-CansaySaid) is used with $n = 1$. So y has the form *pref* f, and both of *pref* p cansay_d f and *pref* p said_d f appear among $x_1, \ldots, x_{r-1}$. By induction, there is a conservative deduction of facts:

(i) *pref* p said_d g,
(ii) *pref* p said_d R exist,
(iii) *pref* x, and
(iv) *pref* S exist,

where g is an R-germ for f, and x is an S-germ for p cansay_d f.

If g is equal to f, then there is a conservative deduction of *pref* p said_d f, and from this, conditions (iii) and (iv) above, and subrule (1) of (P-Superfluous), it follows that there is a conservative deduction of $y =$ *pref* f.

Similarly, if x is equal to p cansay_d f, then there is a conservative deduction of *pref* p cansay_d f, and from this, conditions (i) and (ii) above, and subrule (2) of (P-Superfluous), it follows that there is a conservative deduction of $y =$ *pref* f.

We may therefore assume that x is a strict S-germ for p cansay_d f, and g is a strict R-germ for f. It follows from this that p cansay_d f and f must have the forms

$$\begin{aligned} p \ \mathsf{cansay}_d \ f &= p \ \mathsf{cansay}_d \ q_1 \ \mathsf{cansay}_{d_1} \ \ldots\ldots \ q_l \ \mathsf{cansay}_{d_l} \ \bar{x}, \text{ and} \\ f &= \phantom{p \ \mathsf{cansay}_d \ } q_1 \ \mathsf{cansay}_{d_1} \ \ldots\ldots \ q_k \ \mathsf{cansay}_{d_k} \ \bar{g}, \end{aligned}$$

where,

(v) $q_1, \ldots, q_l \in S$ and x (a strict S-germ for p cansay_d f) has the form p cansay $\bar{x}$,
(vi) $q_2, \ldots, q_k \in R$ and g (a strict R-germ for f) has the form q_1 cansay $\bar{g}$.

In these conditions $l \geq 0$ and $k \geq 1$. But we may assume that l too is at least 1, since otherwise p cansay_d f would be equal to p cansay_d $\bar{x}$, hence it would be conservatively deducible from x, and an argument using subrule (2) of (P-Superfluous), similar to one given above, would establish that y is conservatively deducible.

Note that the forms of p cansay_d f and f are such that:

(vii) $q_1 \ \mathsf{cansay}_{d_1} \ \ldots \ q_l \ \mathsf{cansay}_{d_l} \ \bar{x} = q_1 \ \mathsf{cansay}_{d_1} \ \ldots \ q_k \ \mathsf{cansay}_{d_k} \ \bar{g}$. (Both are equal to f.)

Recall that we are handling the final case in the proof of Lemma 5.11, and our goal is to establish that $y = \mathit{pref}\, f$ is normally deducible. It is enough to establish that either one of

$$\mathit{pref}\ q_1\ \mathsf{cansay}\ \bar{x}, \\ \mathit{pref}\ q_1\ \mathsf{cansay}\ \bar{g}$$

is conservatively deducible. In the case of the former it would follow that

$$\mathit{pref}\ \ f = \mathit{pref}\ \ q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_l\ \mathsf{cansay}_{d_l}\ \bar{x}$$

is normally deducible since $q_2, \ldots, q_l \in S$ and $\mathit{pref}\, S$ exist is conservatively deducible by (iv). In the case of the latter it would follow that

$$\mathit{pref}\ \ f = \mathit{pref}\ \ q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_k\ \mathsf{cansay}_{d_k}\ \bar{g}$$

is normally deducible since $q_2, \ldots, q_k \in R$ and $\mathit{pref}\, R$ exist is conservatively deducible by (ii), rule (P-Exists), and the fact that q $\mathsf{regcomp}$ p said_d q exists.

We consider two cases, depending on whether $l \le k$ or $k < l$. We will show that $\mathit{pref}\, q_1$ cansay $\bar{x}$ is conservatively deducible in the first case, and that $\mathit{pref}\, q_1$ cansay $\bar{g}$ is conservatively deducible in the second.

Suppose $l \le k$. Condition (vii) can then be presented visually as stating that the following two facts are equal:

$$q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_l\ \mathsf{cansay}_{d_l}\ \bar{x} \\ q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_l\ \mathsf{cansay}_{d_l}\ q_{l+1}\ \mathsf{cansay}_{d_{l+1}}\ \ldots\ldots\ q_k\ \mathsf{cansay}_{d_k}\ \bar{g}.$$

It is clear then that $\bar{x}$ is equal to q_{l+1} $\mathsf{cansay}_{d_{l+1}}$ $\ldots\ldots$ q_k cansay_{d_k} $\bar{g}$, and it follows from this that:

- $g = q_1$ cansay $\bar{g}$ is an R-germ for q_1 cansay $\bar{x}$.

Collecting from the conditions above we see in addition that:

- $\mathit{pref}\, p$ said_d q_1 cansay $\bar{g}$ is conservatively deducible, by (i).
- $\mathit{pref}\, p$ cansay $\bar{x}$ is conservatively deducible, by (iii) and the fact that $x = p$ cansay $\bar{x}$.
- $\mathit{pref}\, p$ said_d R exist is conservatively deducible, by (ii).

We are thus in a position to apply subrule (3) of (P-Superfluous) to conclude, as promised, that $\mathit{pref}\, q_1$ cansay $\bar{x}$ is conservatively deducible.

Suppose $k < l$. Condition (vii) can in this case be presented visually as stating that the following two facts are equal:

$$q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_k\ \mathsf{cansay}_{d_k}\ q_{k+1}\ \mathsf{cansay}_{d_{k+1}}\ \ldots\ldots\ q_l\ \mathsf{cansay}_{d_l}\ \bar{x} \\ q_1\ \mathsf{cansay}_{d_1}\ \ldots\ldots\ q_k\ \mathsf{cansay}_{d_k}\ \bar{g}.$$

It is clear then that $\bar{g}$ is equal to q_{k+1} $\mathsf{cansay}_{d_{k+1}}$ $\ldots\ldots$ q_l cansay_{d_l} $\bar{x}$. It follows from this that p cansay $\bar{x}$ is an S-germ for p cansay $\bar{g}$, and therefore also an S-germ for p cansay_d q_1 cansay $\bar{g}$. (Notice that $q_1, q_{k+1}, \ldots, q_l$ all belong to S, by (v).) In other words:

- x is an S-germ for p cansay_d g.

Collecting from the conditions above we see in addition that:

- $\mathit{pref}\, p$ said_d g is conservatively deducible, by (i).
- $\mathit{pref}\, x$ is conservatively deducible, by (iii).
- $\mathit{pref}\, S$ exist is conservatively deducible, by (iv).

We are thus in a position to apply subrule (1) of (P-Superfluous) to conclude that *pref* g is conservatively deducible. In other words *pref* q_1 cansay $\bar{g}$ is conservatively deducible, as promised. This completes the proof of Lemma 5.11. □

Let B be the set of all regular components of the facts in Z. A *B-perturbation* of a canonical fact $x = p_1\ \mathsf{said}_{d_1}\ p_2\ \mathsf{said}_{d_2}\ \cdots\ p_k\ \mathsf{said}_{d_k}\ p_{k+1}$ *attribute* is a fact of the form $y = p_1^*$ $\mathsf{said}_{d_1^*}\ p_2^*\ \mathsf{said}_{d_2^*}\ \cdots\ p_k^*\ \mathsf{said}_{d_k^*}\ p_{k+1}^*$ *attribute** where $p_i^* \in B$, $d_i^* \in \{0, \infty\}$, and *attribute** is either equal to *attribute*, or, if *attribute* starts with cansay, it may be obtained from *attribute* by changing the initial cansay to cansay_0.

Remark 5.12. The number of B-perturbations of a fact x of quotation depth k is at most $(2|B|)^{k+1}$.

Definition 5.13. Let x be a canonical fact. The *contractions* of x are obtained through the following rules:

(1) x itself is a contraction of x.
(2) If *pref* p said_d f is a contraction of x, then so is *pref* f.
(3) If *pref* p cansay_d f is a contraction of x, and f has the form $f_1 \& \ldots \& f_n$, then for each i such that f_i has the form q_i cansay_{d_i} $\bar{f}_i$, *pref* f_i is a contraction of x.

Thus the contractions of x are canonical facts obtained from x by (1) removing occurrences of said_d in the prefix; (2) removing a strict initial segment of the initial occurrences of cansay_d in facts that result from x by distributing cansay_d over conjunctions and separating the conjuncts.

We define the *delegation depth* of a canonical fact as follows: If x has the form *pref* p *attribute* where *attribute* is not of the form cansay_d f, then the delegation depth of x is 0. If x has the form *pref* p cansay_d $f_1 \& \ldots \& f_n$ (with $f_1, \ldots, f_n$ canonical facts) then the delegation depth of x is $1 + \max\{d_i \mid i \le n\}$ where d_i is the delegation depth of f_i.

Remark 5.14. The number of contractions of a fact x is at most $2^\delta l(c+1)$, where δ is the quotation depth of x, l is the delegation depth of x, and c is the number of occurrences of the symbol & in x.

Let U be the closure of Z under B-perturbations and under contractions. Notice that U can be obtained by first taking the closure of Z under contractions, and then taking the closure of the resulting set under B-perturbations; there is no need to then close again under contractions. From this and the last two remarks it follows that:

Claim 5.15. *U is finite, and its size is bounded by $|Z| \cdot |B|^{\delta+1} \cdot 2^{2\delta+1} \cdot l \cdot (c+1)$ where δ bounds the quotation depths of facts in Z, l bounds the delegation depths of facts in Z, and c bounds the number of occurrences of & in facts in Z.*

Let U^+ consist of the facts in U, plus all facts of the form *pref* p exists, where *pref* is a prefix of a fact in U, and $p \in B$. In forming U^+ it is enough to take the maximal prefix from each fact in U, since U is closed under contractions. It follows that $|U^+| = (|B|+1)|U|$, and hence certainly:

Claim 5.16. *The size of $|U^+|$ is bounded by $|Z|(2|B|)^{2\delta+2}l(c+1)$.*

Lemma 5.17. *Every fact which can be obtained from Z by a conservative deduction belongs to U^+.*

Proof. Going over the (P) rules other than (P-Del) we see that in each rule, the fact on the left of $\lesssim$ is either a b-perturbation of at least one of the facts on the right of $\lesssim$, where b consists of the regular components of these facts, or else a contraction of one of these facts, or else of the form $\mathit{pref}\ p$ exists with pref coming from a fact on the right of $\lesssim$ and $p \in b$. (Note that this is true also in each of the subrules of (P-Superfluous). Subrule (2) is the only place where we require condition (3) in the definition of contractions.) The lemma follows from this, and from the definition of U^+, by induction on the length of the deduction. □

Corollary 5.18. *There is an algorithm that, given Z, produces the set X of all canonical facts which can be obtained from Z by conservative deductions.*

Proof. X is the smallest set which contains Z and is closed under the (P) rules other than (P-Del). X is contained in U^+ by the previous lemma, and so the number of iterations required to reach the fixed point X is bounded: it is at most the size of U^+ for which there is a bound which depends algorithmically on Z. The corollary follows. □

We now have the results (a), (b), and (c) promised earlier, and by combining them we can reach a decision procedure for ensue:

Remark 5.19. The following algorithm, with input consisting of facts y and z, determines whether $y \lesssim z$.

First, write z as a conjunction $\& Z$ of canonical facts, and write y as a conjunction $\& Y$ of canonical facts. Generate the finite set U of all contractions and B-perturbations of facts in Z, where B is the set of regular components of the facts in Z.

Second, compute the set X of all canonical facts that are conservatively deducible from Z. This can be done using a logic program based on the (P) rules other than (P-Del) (and including all instances of (P-Superfluous) that use facts in U). X is the least fixed point of the program. Notice that X is contained in the finite set U^+ by Lemma 5.17. From this and Theorem A.4 it follows that X can be computed.

Third, check if each fact in Y is simply deducible from facts in X. This can be done using a simple program relying on Definition 5.5 and the characterization of simple deducibility in Claim 5.6.

By Lemma 5.11, $y \lesssim z$ if and only if the conclusion reached in the final step above is that each fact in Y is indeed simply deducible from facts in X.

We do not write the algorithm in any greater detail, since our goal is not to compute ensue, but to compute knowledge. An algorithm for computing knowledge is obtained in the next subsection. The (P)-rules and results in this subsection are just an intermediary.

5.2. Basic queries.

Fix a substrate X. We describe in this subsection an algorithm which computes the answer to a basic query $Q = (a\ \mathsf{knows}_i\ t(v_1, \ldots, v_k))$ under an authorization policy $\mathcal{A}$, from input consisting of the policy and the query.

Our plan is to isolate a finite substructure $\mathring{X}$ of the substrate X that is rich enough so that the computation of the answer can be done using a logic program $\bar{\Pi}$ over $\mathring{X}$. We then use Theorem A.4 to compute the fixed point for $\bar{\Pi}$.

$\mathring{X}$ must be rich enough to determine the ensue relation over it. We shall achieve this by closing $\mathring{X}$ under contractions, B-perturbations for a large enough set B of regular elements, and "exists" facts. The results of the previous subsection will then allow us to compute the ensue relation over $\mathring{X}$.

Let us begin with the precise definition of $\mathring{X}$. Without loss of generality we may assume that all fact terms in the assertions of $\mathcal{A}$ are canonical. Recall that by Claim 5.1, every fact term f is equivalent to a conjunction $f_1 \;\&\cdots\&\; f_n$ of canonical fact terms. An assertion α with the left side q knows_d f or q says_d f to p can be replaced with n assertions obtained from α by replacing f with $f_1, \ldots, f_n$ respectively. An assertion α with q knows_d f on the right can be replaced with the assertion obtained from α by replacing q knows_d f with q knows_d $f_1 \wedge \cdots \wedge q$ knows_d f_n.

Without loss of generality, we may assume that the query term t is the conjunction $t_1 \;\&\cdots\&\; t_m$ of canonical terms, the *canonical components* of t.

Let L consist of all fact terms x which appear on the left of assertions $q :_d\; x \;\leftarrow\; x_1, \ldots, x_n, con$ in $\mathcal{A}$, and all fact terms q said_d x for which there is an assertion $q :_d\; x$ to $p \;\leftarrow\; x_1, \ldots, x_n, con$ in $\mathcal{A}$. Let R consist of all fact terms which appear on the right in assertions in $\mathcal{A}$, plus the canonical components of the fact term t of the query we are answering. Let B consist of the values in X of the following ground terms of regular sorts: the term a (for the principal whose knowledge we intend to query), all terms for principals who placed assertions in $\mathcal{A}$, and all ground regular components of the head facts of the assertions in $\mathcal{A}$. Let R^* and L^* consist of the facts obtained from terms in R and L respectively by substituting elements of B for variables. Let U be the closure of L^* under contractions and B-perturbations. Let F be the set of all prefixes used in U. Let U^+ consist of U plus all facts of the form *pref* p exists for *pref* $\in F$ and $p \in B$.

Let T be the set of all terms in $\mathcal{A}$ or Q (so that T is closed under subterms). Let T^* be the set of the values of terms in T using variables assignments with values in B. Now we are ready to define the finite structure mentioned in the preamble of Section 5. $\mathring{X}$ is the restriction of the substrate X to the domain $B \cup U^+ \cup R^* \cup T^*$. Note that $\mathring{X}$ is indeed finite; a bound on its size can be computed from $\mathcal{A}$ and Q using the results of the previous subsection:

Definition 5.20. The *width* of an assertion $A :_d\; x \;\leftarrow\; x_1, \ldots, x_n, con$ is the number of variables in the assertion. The width of an assertion $A :_d\; x$ to $p \;\leftarrow\; x_1, \ldots, x_n, con$ is the number of variables in the assertion excluding the variable p. The width of a fact term t is the number of variables in t.

Claim 5.21. *The size of $\mathring{X}$ is bounded by $n + f \cdot n^w \cdot (2n)^{2\delta+2} \cdot l \cdot (c+1)$, where*

- *n is the number of regular ground terms in the policy $\mathcal{A}$ or the query Q,*
- *f is the number of terms in $\mathcal{A}$ or Q,*
- *w bounds the widths of assertions in $\mathcal{A}$ and width of the query Q.*
- *δ bounds the quotation depths of these terms,*
- *l bounds their delegation depths,*
- *c bounds the number of occurrences of & in the terms, and*

Proof. Note that the size of $L^* \cup R^* \cup T^*$ is bounded by $f \cdot n^w$. Now apply Claim 5.16. □

We now pass to the matter of writing a program $\bar{\Pi}$ on $\mathring{X}$, which for purposes of answering our query is equivalent to the program Π on X.

The program $\bar{\Pi}$ consists of all the assertions in the policy $\mathcal{A}$, and modified house rules (S-1)–(S-8) as follows. Each item below is a rule *schema*, that is a list of rules, parameterized by facts. The only variables in the rules are of type Regular. The first two schemas, (S-1) and (S-2), deal with the house rules K0∞ and Say2know. The remaining schemas deal with consequences on knowledge of the (P) rules for ensue deductions.

(S-1) r knows $f \leftarrow r$ knows_0 f, for each fact $f \in U^+$.
(S-2) r knows q said_d $f \leftarrow q$ says_d f to r, for each fact q said_d $f \in U$.
(S-3) r knows_d $\mathit{pref}\, q$ $\mathsf{exists} \leftarrow q$ $\mathsf{regcomp}$ $f \wedge r$ knows_d $\mathit{pref}\, f$, for every fact $\mathit{pref}\ f \in U$.
(S-4) r knows_d $\mathit{pref}\, p$ said $f \leftarrow r$ knows_d $\mathit{pref}\, p$ said_0 f, for each fact $\mathit{pref}\, p$ said_0 $f \in U$.
(S-5) Adaptations of each of the rules (P-SelfQuote), (P-Cansay0∞), (P-CansaySaid), (P-Del$^-$), and (P-Role), but not (P-Del), similar to the adaptation of the rule (P-Said0∞) in the previous item.
(S-6) r knows_d $\mathit{pref}\, f \leftarrow r$ knows_d $\mathit{pref}\, p$ said_{d^*} $f \wedge r$ knows_d $\mathit{pref}\, x \wedge r$ knows_d $\mathit{pref}\, H$ exist, whenever x is an H-germ for p cansay_{d^*} f, the facts $\mathit{pref}\, p$ said_{d^*} f and $\mathit{pref}\, x$ belong to U, and the members of H are essential to obtaining p cansay_{d^*} f from x.
(S-7) Adaptations of subrules (2) and (3) of (P-Superfluous), similar to the adaptation of subrule (1) in the previous item.
(S-8) r knows_d $y \leftarrow r$ knows_d $x \wedge r$ knows_d H exist, whenever $x \in U$, $y \in R^*$, x is an H-germ for y, and the members of H are essential for obtaining y from x.

Claim 5.22.
- *If* $\bar\Pi(\mathring{X}) \models r$ knows_d z *and* b $\mathsf{regcomp}$ z*, then* $b \in B$*.*
- *If* $\bar\Pi(\mathring{X}) \models q$ says_d z to r*, then* $q \in B$*, and if* b $\mathsf{regcomp}$ 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 knows_d z, and b $\mathsf{regcomp}$ z, then:
  (a) $b \in B$, or
  (b) there is a clause r $\mathsf{knows}_{d'}$ z' in the premise of R so that b $\mathsf{regcomp}$ z', or
  (c) there is a clause q $\mathsf{says}_{d'}$ z' to r in the premise of R so that $b = q$ or b $\mathsf{regcomp}$ z'.
(2) If R is an instance of a rule in $\bar\Pi$ with conclusion q says_d z 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 $\mathsf{knows}_{d'}$ z' in the premise of R so that b $\mathsf{regcomp}$ z'.

Inspection of the rules (S-1)–(S-8) immediately establishes conditions (1) and (2) for their instances. (The third possibility in condition (1) appears because of rule (S-2).) It remains to check the conditions for instances of assertions in $\mathcal{A}$.

We deal first with instances of assertions of the form (As1). Suppose that

$$A :_d \ x \ \leftarrow \ x_1, \dots, x_n, \mathit{con}$$

is an assertion of this form in $\mathcal{A}$. Recall from Subsection 3.1 that the full assertion is

$$A \ \mathsf{knows}_d\ x \ \leftarrow A\ \mathsf{knows}_d\ x_1 \wedge \cdots \wedge A\ \mathsf{knows}_d\ x_n \wedge \mathit{con} \wedge$$
$$A\ \mathsf{knows}_d\ \tau_1\ \mathsf{exists}\ \wedge \cdots \wedge A\ \mathsf{knows}_d\ \tau_k\ \mathsf{exists}$$

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

Suppose next that

$$A :_d \ x\ \mathsf{to}\ p \ \leftarrow \ x_1, \dots, x_n, \mathit{con}$$

is an assertion of the form (As2) in $\mathcal{A}$. Recall from Subsection 3.1 that the full assertion is

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

where again $\tau_1, \ldots, \tau_k$ include all non-ground regular components of x. The value of A belongs to B by definition, and so do the values of the ground regular components of x. Condition (2) follows. □

The program $\bar{\Pi}$ is quite large, as each of the items (S-1)–(S-8) above is a schema. Inspecting the definition of $\bar{\Pi}$ we obtain the following immediate bound on the number of rules in the program:

Claim 5.23. *The number of rules in* $\bar{\Pi}$ *is* $O(r + (|U^+| + |R^*|) \cdot l \cdot c)$ *where* r *is the number of rules in* $\mathcal{A}$, l *bounds the delegation depths of facts in* U, *and* c *bounds the number of conjunctions in facts in* U.

Notice also that the rules of the program $\bar{\Pi}$ are easily computed from $\mathcal{A}$ and the query. From this and the fact that $\mathring{X}$ is finite, it follows that:

Proposition 5.24. $\bar{\Pi}(\mathring{X})$ *is computable from* $\mathcal{A}$ *and the query.*

We shall thus be done with computing the answer to the query $a\ \mathsf{knows}_i\ t(v_1, \ldots, v_k)$ under the policy $\mathcal{A}$ if we can show that,

$$(5)\quad \begin{aligned} &\Pi(X) \models \big[a\ \mathsf{knows}_i\ s(b_1, \ldots, b_k) \wedge a\ \mathsf{knows}_i\ b_1\ \mathsf{exists}\ \wedge \ldots \wedge a\ \mathsf{knows}_i\ b_k\ \mathsf{exists}\big] \iff \\ &\bar{\Pi}(\mathring{X}) \models \big[a\ \mathsf{knows}_i\ s(b_1, \ldots, b_k) \wedge a\ \mathsf{knows}_i\ b_1\ \mathsf{exists}\ \wedge \ldots \wedge a\ \mathsf{knows}_i\ b_k\ \mathsf{exists}\big] \end{aligned}$$

for each canonical component s of the query term t.

The right-to-left implication is clear, and indeed the same direction is clear for all facts and principals, and for both the predicates knows_d and says_d, namely:

Claim 5.25. (1) *If* $\bar{\Pi}(\mathring{X}) \models p\ \mathsf{says}_d\ f\ \mathsf{to}\ q$, *then* $\Pi(X) \models p\ \mathsf{says}_d\ f\ \mathsf{to}\ q$.
(2) *If* $\bar{\Pi}(\mathring{X}) \models p\ \mathsf{knows}_d\ f$, *then* $\Pi(X) \models p\ \mathsf{knows}_d\ f$.

Proof. Each of the rules in the program $\bar{\Pi}$ is true in the superstrate $\Pi(X)$. Hence everything said (respectively known) in $\bar{\Pi}(\mathring{X})$ is also said (respectively known) in $\Pi(X)$. □

We shall prove the left-to-right implication in (5) by expanding the interpretations of knows_d and says_d in $\bar{\Pi}(\mathring{X})$ to relations knows^*_d and says^*_d over X (without adding any new instances within $\mathring{X}$), and proving that these expanded relations are equal to the interpretations of knows_d and says_d in $\Pi(X)$.

The definition of the star-relations below is done in a very naive manner, starting from the knowledge in $\bar{\Pi}(\mathring{X})$, and then sequentially adding consequences resulting from assertions in $\mathcal{A}$, and from the rules Say2know, K0∞, K&, and KMon. The point here is that, although knows_d and says_d are generally produced by a simultaneous recursion using the assertions and rules, all the mutual dependence is already handled in $\bar{\Pi}(\mathring{X})$. Thus, when extending knows_d and says_d from $\mathring{X}$ to X, direct, non-recursive definitions suffice. Those direct definitions are the definitions of the star-relations below.

Set $c\ \mathsf{says}^*_d\ f\ \mathsf{to}\ b$ just in case that this follows from an assertion in $\mathcal{A}$ based on the knowledge in $\bar{\Pi}(\mathring{X})$. Precisely, $c\ \mathsf{says}^*_d\ f\ \mathsf{to}\ b$ just in case that:

- there is an instance $c :_d\ f \ \mathsf{to}\ b \ \leftarrow\ f_1, \dots, f_n, \mathit{con}$ of an assertion in $\mathcal{A}$, and
- the body of this instance holds in $\bar{\Pi}(\mathring{X})$.

Set $b\ \mathsf{knows}^1_d\ f$ just in case this follows from an assertion in $\mathcal{A}$ based on the knowledge in $\bar{\Pi}(\mathring{X})$. Precisely this means that:

- there is an instance $b :_d\ f \ \leftarrow\ f_1, \dots, f_n, \mathit{con}$ of an assertion in $\mathcal{A}$, and
- the body of this instance holds in $\bar{\Pi}(\mathring{X})$.

Set $b\ \mathsf{knows}^2_\infty\ f$ just in case this follows from the rule $p\ \mathsf{knows}^2_\infty\ q\ \mathsf{said}_d\ x \leftarrow q\ \mathsf{says}^*_d\ x\ \mathsf{to}$ p, which paraphrases Say2know. Precisely, $b\ \mathsf{knows}^2_\infty\ f$ just in case that:

- f has the form $c\ \mathsf{said}_d\ x$, and
- $c\ \mathsf{says}^*_d\ x\ \mathsf{to}\ b$.

Set $b\ \mathsf{knows}^3_d\ f$ to hold just in case that this follows by K0∞ from the knowledge in knows^1_d and knows^2_d. Precisely:

- $b\ \mathsf{knows}^3_0\ f$ just in case that $b\ \mathsf{knows}^1_0\ f$.
- $b\ \mathsf{knows}^3_\infty\ f$ just in case that $b\ \mathsf{knows}^1_0\ f$, $b\ \mathsf{knows}^1_\infty\ f$, or $b\ \mathsf{knows}^2_\infty\ f$.

Finally, set $b\ \mathsf{knows}^*_d\ f$ just in case that this follows by K& and KMon from the knowledge in knows^3_d. Precisely, $b\ \mathsf{knows}^*_d\ f$ just in case that there is a set Z of canonical facts so that:

- f ensues $\mathbf{\&}\, Z$, and
- $b\ \mathsf{knows}^3_d\ z$ for each $z \in Z$.

Claim 5.26. (1) *If $b\ \mathsf{says}^*_d\ f\ \mathsf{to}\ c$, then $\Pi(X) \models b\ \mathsf{says}_d\ f\ \mathsf{to}\ c$.*
(2) *If $b\ \mathsf{knows}^*_d\ f$, then $\Pi(X) \models b\ \mathsf{knows}_d\ f$.*

Proof. says^*_d is defined using rules which hold of says_d in $\Pi(X)$, on the basis of knowledge which holds in $\Pi(X)$ (by Claim 5.25). Hence all true instances of says^*_d are true in $\Pi(X)$ of says_d. A similar argument applies to knows^*_d. □

We say that a fact *pref* y is a *simple consequence* of *pref* x and *pref* H exist, if x is an H-germ for y. (Note in this case that knowledge of *pref* x and *pref* H exist entails knowledge of *pref* y.)

Claim 5.27. *Let $b \in B$ (hence $b \in \mathring{X}$).*

1. *If $c\ \mathsf{says}^*_d\ f\ \mathsf{to}\ b$ then $\bar{\Pi}(\mathring{X}) \models c\ \mathsf{says}_d\ f\ \mathsf{to}\ b$. Moreover, $c\ \mathsf{said}_d\ f$ belongs to U.*
2. *If $b\ \mathsf{knows}^1_d\ f$ then $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ f$. Moreover, f belongs to U.*
3. *If $b\ \mathsf{knows}^2_\infty\ f$ then $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_\infty\ f$. Moreover, f belongs to U.*
4. *Suppose f is canonical. If $b\ \mathsf{knows}^*_d\ f$ then there is a fact pref g and a set H so that $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d$ pref g, $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d$ pref H exist, and f is a simple consequence of pref g and pref H exist. Moreover, the facts pref g and pref H exist belong to U^+.*

Proof. We begin with (1). Suppose $c\ \mathsf{says}^*_d\ f\ \mathsf{to}\ b$. By definition of says^*_d, there must be an assertion

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

in $\mathcal{A}$, and an assignment ξ of values to the variables of the assertion, so that the instance of the assertion generated by making the assignment ξ has the form

(b) $$c :_d\ f\ \mathsf{to}\ b\ \leftarrow\ f_1, \dots, f_n, \mathit{con}$$

with the body of the instance true in $\bar{\Pi}(\mathring{X})$.

Note that c must belong to B by the definition of B, since $\mathcal{A}$ has an assertion placed by A, and c is the value of A. Thus certainly c belongs to $\mathring{X}$.

Recall from Subsection 3.1 that the full form of (a) is

$$\text{(c)} \qquad \begin{aligned} A \ \mathsf{says}_d \ x \ \mathsf{to} \ p \ \leftarrow \ & A \ \mathsf{knows}_d \ x_1 \wedge \cdots \wedge A \ \mathsf{knows}_d \ x_n \wedge \mathit{con} \wedge \\ & A \ \mathsf{knows}_d \ \tau_1 \ \mathsf{exists} \ \wedge \cdots \wedge A \ \mathsf{knows}_d \ \tau_k \ \mathsf{exists} \end{aligned}$$

where $p, \tau_1, \ldots, \tau_k$ include all the variables of the assertion. The variable p must be assigned value b by ξ, to reach the instance (b). This value belongs to B by assumption of the claim. All other variables must be assigned values b_i with $\bar{\Pi}(\mathring{X}) \models A \ \mathsf{knows}_d \ b_i \ \mathsf{exists}$, to make the body of the instance (b) true in $\bar{\Pi}(\mathring{X})$. Using Claim 5.22, these values too must belong to B.

Thus, the instance (b) is obtained by assigning values in B to the variables of the assertion (a). Since the assertion (a) is part of the program $\bar{\Pi}$, and since B is included in the universe of $\mathring{X}$, it follows that $\bar{\Pi}(\mathring{X})$ satisfies the instance (b). Since the body of this instance is true in $\bar{\Pi}(\mathring{X})$, the head must be true too. Thus $\bar{\Pi}(\mathring{X}) \models c \ \mathsf{says}_d \ f \ \mathsf{to} \ b$. Further, the term $A \ \mathsf{said}_d \ x$ belongs to L by definition, and since $c \ \mathsf{said}_d \ f$ is an instance of this term using variable values from B, it belongs to U. This proves condition (1) of the claim.

The proof of condition (2) is similar, again using the fact that only assertion instances using variable values from B can have bodies true in $\bar{\Pi}(\mathring{X})$. (And again, the proof of this fact uses the full assertion form in Subsection 3.1, and Claim 5.22.)

Condition (3) is an immediate consequence of condition (1), the definition of knows^2_∞, and the inclusion of (S-2) in $\bar{\Pi}$.

It remains to address condition (4). Note that if $b \ \mathsf{knows}^3_d \ f$ then $\bar{\Pi}(\mathring{X}) \models b \ \mathsf{knows}_d \ f$, and $f \in U$. This follows from conditions (2) and (3) because knows^3_d is defined from knows^1_d and knows^2_∞ using the rule K0∞, and because this rule is part of $\bar{\Pi}$ (it is included as the schema (S-1)). To establish condition (4) of the claim we must pass from knows^3_d to knows^*_d. We will use the fact that all the rules for conservative ensue deductions were incorporated into $\bar{\Pi}$ through the schemas (S-3)–(S-7).

Suppose then that $b \ \mathsf{knows}^*_d \ f$, f is canonical, and $b \in B$. Using the definition of knows^*_d, it follows that there is a set Z of canonical facts so that:

(i) f ensues $\& Z$, and
(ii) $b \ \mathsf{knows}^3_d \ z$ for each $z \in Z$,

which, as we noted above, implies:

(iii) $\bar{\Pi}(\mathring{X}) \models b \ \mathsf{knows}_d \ z$, and $z \in U$, for each $z \in Z$.

Since f ensues $\& Z$, there is, by the results of the previous subsection, a normal ensue deduction of f from Z, namely a conservative deduction followed by a simple deduction. Hence there is a fact $\mathit{pref}\, g$ and a set H so that:

(iv) $\mathit{pref}\, g$ and $\mathit{pref}\, H$ exist are conservatively deducible from Z, and
(v) f is a simple consequence of $\mathit{pref}\, g$ and $\mathit{pref}\, H$ exist.

Every regular component of a fact in Z must belong to B by (iii) and Claim 5.22. U is by definition closed under contractions and B-perturbations, and by (iii) it contains Z. By Lemma 5.17 it follows from this and from (iv) that $\mathit{pref}\, g$, $\mathit{pref}\, H$ exist, and all the facts in the conservative deduction leading to them, belong to U^+ and therefore belong to $\mathring{X}$.

The KMon consequences of each of the conservative deduction rules were incorporated into $\bar{\Pi}$, through the schemas (S-3)–(S-7). It follows from this, from condition (iv), from condition (iii), and from the conclusion of the last paragraph, that $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ \mathit{pref}\, g$ and $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ \mathit{pref}\, H\ \mathsf{exist}$. This completes the proof of (4). □

Remark 5.28. Note how the "A-bound" restrictions in the assertion forms in Subsection 3.1 are used in the decidability proof. The restriction that all non-ground regular components of x must be A-bound is used in the proof of Claim 5.22. The restriction that all variables (other than p, in the case of form (As2)) must be A-bound is used in the proof of conditions (1) and (2) of Claim 5.27.

Corollary 5.29. *Let $b \in B$ and suppose that $b\ \mathsf{knows}^*_d\ f$ where f is an instance of a fact term in R (namely a fact term appearing on the right side of an assertion in $\mathcal{A}$, or a canonical component of the query term t). Then $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ f$.*

Proof. Note that f, being an instance of a term in R, is canonical. By condition (4) of the previous claim, there must exist a fact $\mathit{pref}\, g$ and a set H so that:

(i) $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ \mathit{pref}\, g$,
(ii) $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ \mathit{pref}\, H\ \mathsf{exist}$, and
(iii) f is a simple consequence of $\mathit{pref}\, g$ and $\mathit{pref}\, H\ \mathsf{exist}$.

If f is equal to $\mathit{pref}\, g$ then we are done. Otherwise, g must have the form $\mathit{pref}\, b_1\ \mathsf{cansay}\ \bar{g}$, and f must have the form $\mathit{pref}\, b_1\ \mathsf{cansay}_{d_1}\ \ldots\ b_n\ \mathsf{cansay}_{d_n}\ \bar{g}$, with $b_2, \ldots, b_n \in H$. It follows that all regular components of f must be regular components of $\mathit{pref}\, g$ or of $\mathit{pref}\, H\ \mathsf{exist}$, and either way they belong to B by Claim 5.22. From this and the fact that f is an instance of a term in R, it follows that f belongs to R^*. Now using conditions (i)–(iii) and the schema (S-8) it follows that $\bar{\Pi}(\mathring{X}) \models b\ \mathsf{knows}_d\ f$. □

Claim 5.30. (1) *If $\Pi(X) \models b\ \mathsf{says}_d\ f\ \mathsf{to}\ c$, then $b\ \mathsf{says}^*_d\ f\ \mathsf{to}\ c$.*
(2) *If $\Pi(X) \models b\ \mathsf{knows}_d\ f$, then $b\ \mathsf{knows}^*_d\ f$.*

Proof. It is enough to show that says^*_d and knows^*_d satisfy the rules K0∞, Say2know, K&, and KMon, and satisfy all the assertions in $\mathcal{A}$. The implications in the claim then follow since the interpretations of says_d and knows_d in $\Pi(X)$ form the *least* fixed point for these rules.

By their very definition, says^*_d and knows^*_d satisfy the rules K0∞, Say2know, K&, and KMon. We must verify that they satisfy the assertions in $\mathcal{A}$.

Consider first an instance of an assertion in $\mathcal{A}$ of the form $b :_d\ f\ \mathsf{to}\ c\ \leftarrow\ f_1, \ldots, f_n, \mathit{con}$. Note that $b \in B$, since B by definition has all principals who place assertions in $\mathcal{A}$. Suppose that the body of the assertion is true with the interpretation of knowledge given by knows^*_d. By Corollary 5.29, the body of the assertion is true in $\bar{\Pi}(\mathring{X})$. From this and the definition of says^*_d it follows that $b\ \mathsf{says}^*_d\ f\ \mathsf{to}\ c$, as required.

A similar argument, again using Corollary 5.29, handles the case of an assertion of the form $b :_d\ f\ \leftarrow\ f_1, \ldots, f_n, \mathit{con}$. □

We now have established the equivalence (5). The right-to-left direction follows from Claim 5.25, and the left-to-right direction follows from Claim 5.30 and Corollary 5.29.

The answer to the query Q is therefore equal to the set of all $\langle b_1, \dots, b_k \rangle$ so that

$$\bar{\Pi}(\mathring{X}) \models a \ \mathsf{knows}_i \ b_1 \ \mathsf{exists} \ \ \wedge \cdots \wedge a \ \mathsf{knows}_i \ b_k \ \mathsf{exists} \ \text{, and}$$

$$\bar{\Pi}(\mathring{X}) \models a \ \mathsf{knows}_i \ s(b_1, \dots, b_k)$$

for each canonical component s of the query term t. Since $\mathring{X}$ is finite, the entire superstrate $\bar{\Pi}(\mathring{X})$ is computable. (The proof of Theorem A.4 contains an algorithm for computing the superstrate of a logic program over a finite substrate $\mathring{X}$, of course with calls to an algorithm for computing the substrate relations and functions.) It follows that the answer is computable. We proved

Theorem 5.31. *Let X be a substrate. There is an algorithm which computes the answer to a query a knows_i t under an authorization policy $\mathcal{A}$, from input consisting of the policy and the query.*

This is Theorem 3.6 in the notation of Section 3. We end by noting the time complexity of our algorithm for answering queries. Recall that for the purpose of finding the time complexity we assume (see §A.2) that there is an algorithm Eval that evaluates the basic functions and relations of X in constant time. We also assume that constants and variables are strings in a fixed finite alphabet, so that all expressions in the language of our model are strings in a finite alphabet. The length of a term, an assertion, or a query, is simply its length when written as a string, and the length of an authorization policy is the sum of the lengths of its assertions. For reader's convenience, we restate Theorem 3.7.

Theorem 5.32. *The time complexity of our algorithm for computing the answer to a query $Q = (a \ \mathsf{knows}_i \ t)$ under an authorization policy $\mathcal{A}$ is bounded by a polynomial in* $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+w}$*, where δ bounds the quotation depths of canonical facts in t and in $\mathcal{A}$, and w bounds the widths of t and of assertions in $\mathcal{A}$.*

In particular, assuming a fixed bound on widths and quotation depths, the time complexity is polynomial in the length of $\mathcal{A}$ and Q.

Proof. Our algorithm begins by determining the universe of $\mathring{X}$ (that is, determining the set of strings in the universe of X that constitutes the universe of $\mathring{X}$), and determining the program $\bar{\Pi}$. Inspecting the definitions of $\mathring{X}$ and $\bar{\Pi}$, and using Claims 5.21 and 5.23, it is clear that this can be done in time complexity bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+w}$. Let us just note on this matter that the arguments n, f, l, c, and r in Claims 5.21 and 5.23 are all smaller than $\text{length}(\mathcal{A}) + \text{length}(Q)$.

Our algorithm proceeds by computing the superstrate $\bar{\Pi}(\mathring{X})$.

By Claim 5.21, the size N of $\mathring{X}$ is bounded by a polynomial in $(\text{length}(\mathcal{A})+\text{length}(Q))^{\delta+w}$. The number n of regular elements of $\mathring{X}$ is bounded by $\text{length}(\mathcal{A}) + \text{length}(Q)$. From Claim 5.23 and inspection of the individual rules (S-1)–(S-8) it follows that the length ℓ of the program $\bar{\Pi}$ is bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+w}$.

By Theorem A.4, the time complexity of computing $\bar{\Pi}(\mathring{X})$ is bounded by $\ell \cdot n^w$ times a polynomial in N. Combining this with the observations of the previous paragraph we infer that the time complexity of computing $\bar{\Pi}(\mathring{X})$ is bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+w}$.

Finally, our algorithm produces all tuples $\langle b_1, \dots, b_k \rangle$ of elements of B, so that $\bar{\Pi}(\mathring{X}) \models a$ knows_i $s(b_1, \dots, b_k)$ for each canonical component s of the query term t, and $\bar{\Pi}(\mathring{X}) \models a$

$\mathsf{knows}_i\ \{b_1,\dots,b_k\}\ \mathsf{exist}$. (Note that it is enough to go over $b_1,\dots,b_k \in B$, since by Claim 5.22 only the existence of elements of B can be known to a.) There are at most $n^k \le n^w$ tuples to go over, and the number of components s is smaller than the length of t which is smaller than the length of Q, so this step too has time complexity bounded by a polynomial in $(\text{length}(\mathcal{A}) + \text{length}(Q))^{\delta+w}$. □

6. 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 2 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) $p\ \mathsf{says}_d\ x \leftarrow p\ \mathsf{knows}_d\ x$
(2) $p\ \mathsf{knows}_d\ x \leftarrow p\ \mathsf{says}_d\ x$

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 6.1. 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 6.4 in this connection. By translating SecPAL to the original DKAL we gain access to the complexity results in Section 5, specifically Theorem 5.32.

6.0.1. **Substrate.** The SecPAL document [2] 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 6.2. As far as the basic constraint domain of [2] is concerned, it is straightforward to view it as a many-sorted first-order structure, except for the relation e $\mathsf{matches}$ *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 [2], 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 $\mathsf{matches}$ *pattern* to $s(e)$ $\mathsf{matches}$ *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.

6.0.2. **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).

6.0.3. **Constraints.** If *con* is a SecPAL constraint then $\tau(\mathit{con}) = \mathit{con}$.

6.0.4. **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 $c_1, \dots, c_j$ are SecPAL constants such that $(c_1, \dots, c_j) \in \mathrm{Dom}(P)$ then $\tau P(\tau c_1, \dots, \tau c_j)$, syntactically equal to $P(c_1, \dots, c_j)$, is an attribute in Sub. In particular, if $j = 0$ then τP is an Attribute constant. For example, a nullary predicate $\mathsf{is\ a\ friend}$ becomes an Attribute constant.

6.0.5. **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 facts in our model.

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

It is easy to check by induction that, if we ignore the difference between $\mathsf{can\ say}$ and cansay, and between $\mathsf{can\ act\ as}$ and $\mathsf{canActAs}$, 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.

Lemma 6.3. *For every SecPAL fact f and substitution θ, we have $\tau(\theta(f)) = \theta(\tau(f))$.*

The proof is obvious.

6.0.6. **Assertions.** A SecPAL assertion has the form A says f if $f_1, \dots, f_n, \mathit{con}$, 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 $\tau(A\ \mathsf{says}\ f\ \mathsf{if}\ f_1, \dots, f_n, \mathit{con})$ to be the following DKAL double assertion:

$$A_d : \ f \ \mathsf{to}\ \mathit{all} \ \leftarrow \ f_1, \dots, f_n, \mathit{con}.$$

Remark 6.4. 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 can say$_{d_1}$ $\ldots$ e_n 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$_{d_{k+1}}$ $\ldots$ e_n can say$_{d_n}$ g, $0 \leq k \leq n$, as a *subfact* of f. Define $\tau(A \textsf{ says } f \textsf{ if } f_1, \ldots, f_n, con)$ to be the set of the following DKAL assertions:

$$A_d : f \ \leftarrow f_1, \ldots, f_n, con$$

$$A_d : f' \textsf{ to all } \leftarrow f'$$

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

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

$$B \textsf{ says foo} \leftarrow$$

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. □

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

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

Theorem 6.5 (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 term, and d range over* $\{0, \infty\}$. *If*

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

in SecPAL then

$$\Pi(\text{Sub}) \models A \textsf{ says}_d\ f$$

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 term τf (aka f) are variables. Thus, the parts A $\textsf{knows}$ t_i $\textsf{exists}$ in the full forms of assertions $A :_d f$ $\textsf{to}$ $all \leftarrow f_1, \ldots, f_n, con$ in $\tau(\text{AC})$ hold automatically under all relevant assignments. □

Remark 6.6. The following analog of the embedding theorem is true if one uses the translation of Remark 6.4, 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 $\bigcup_{\alpha\in\text{AC}} \tau\alpha$ with the translation τ of Remark 6.4. Further, let A be a SecPAL principal constant, f be a SecPAL ground fact term, and d range over $\{0, \infty\}$. Suppose f is either flat, or nested with all its constants appearing in AC. If

$$\text{AC}, d \vdash A \ \textsf{says}\ f \tag{A}$$

in SecPAL then

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

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

in DKAL, and the assertions

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

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

Proof. Consider the following SecPAL assertion context AC:

(1) A $\textsf{says}$ B $\textsf{cansay}$ D $\textsf{cansay foo}$.
(2) B $\textsf{says}$ C $\textsf{cansay foo}$. (It's $\textsf{foo}$, not D $\textsf{cansay foo}$.)
(3) C $\textsf{says}$ D $\textsf{cansay foo}$.
(4) D $\textsf{says foo}$.

In SecPAL, you get only these consequences:

5. C $\textsf{says foo}$ (from 3 and 4).
6. B $\textsf{says foo}$ (from 2 and 5).

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

7. B $\textsf{says}$ C $\textsf{cansay}$ D $\textsf{cansay foo}$ (from 2).
8. B $\textsf{says}$ D $\textsf{cansay foo}$ (from 7 and 3).
9. A $\textsf{says}$ D $\textsf{cansay foo}$ (from 1 and 8).
10. A $\textsf{says foo}$ (from 9 and 4).

The difference between SecPAL and DKAL appears in line 7. In line 2, B gives C the right to promulgate $\textsf{foo}$, and to delegate with no depth restrictions. In DKAL, but not in SecPAL, this results in line 7, which formulates an instance of delegatability of this right on the part of C. □

We see Theorem 6.7 as an advantage of DKAL: more justified requests get positive answers.

Appendix A. Logic

We recall existential fixed-point logic, introduced in [3], in the form appropriate for our purposes in the main part of this paper. We presume that the reader is familiar with the basics of first-order logic; one popular logic textbook is [5]. Nevertheless, we recall some basic definitions and facts. By default our logics are many-sorted logics with equality.

A.1. **Existential first-order logic.**

A.1.1. **Vocabulary.** A vocabulary consists of sort symbols, function symbols and relation symbols. Each function symbol has an integer arity $r \geq 0$. Nullary function symbols are known as *constants*. The *type* of a constant is a sort symbol. A function of positive arity r has a type of the form $D \to S$ where S is a sort symbol and D is a union of components of the form $T_1 \times \cdots \times T_r$ where each T_i is a union of sorts. For example, in the main part of the paper we have a binary function of the type

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

where Regular is the union of so called regular sorts. (Without loss of generality, we could require that each T_i is a single sort. But the more liberal condition is more convenient.) Each relation symbol has an integer arity $r \geq 1$ and a type of the form D where D is as above. Every vocabulary contains the equality sign.

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 constants may be infinite. Each constant is a string in a fixed finite alphabet. Similarly, we will require that each variable is a string in a fixed finite alphabet.

A.1.2. **Total structures.** A *total structure* X of some vocabulary Υ consists of a nonempty set, the *universe* of X, together with interpretations of the vocabulary symbols over the universe: sorts, basic functions and basic relations. The sorts are subsets of the universe and their union is the entire universe; a sort may be empty. Elements of a sort S are also elements of type S. Thus sorts are types, but types are not necessarily sorts. (We do not insist that sorts partition the universe, but do not explicitly use the possibility that an element may have several sorts.)

All basic functions are total. The interpretations of basic functions and relations conform to their types. For example, a basic function of type

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

has one Principal argument and another argument whose type is either Attribute or Speech; its values are of type Fact. The interpretation of a function or relation name N is denoted N_X. The equality sign is interpreted in obvious way. If φ is a first-order Υ formula and ξ is an assignment of elements of X to the free variables of φ then φ is either true or false in X under ξ. In the first case we write $X, \xi \models \varphi$. If φ is a sentence, that is a formula with no free variables, we write $X \models \varphi$.

A nonempty subset of X, closed under all basic function, gives rise to a *total substructure* of X.

Remark A.1. The name of a basic function or relation is its identifier. For example, two constants with the same value are different constants. However, we won't be too pedantic about the distinction between a function or relation and its name.

A.1.3. **Partial structures.** A *partial structure* is like a total structure except that basic functions may be partial. Total structures are special partial structures.

Let X be a partial structure of some vocabulary Υ. An assignment ξ of elements of X to the variables of a Υ term t is *safe* over X, if the value of t is defined in X under ξ. An assignment ξ of elements of X to the variables of a quantifier-free formula φ of vocabulary Υ is *safe* over X if ξ is safe for every term of φ.

A non-empty subset of X gives rise to a *partial substructure* of X. Suppose that Y is a partial substructure of X, and φ is a quantifier-free formula of vocabulary Υ. For every safe assignment ξ of elements of Y to the variables of φ, we have that $X, \xi \models \varphi$ if and only if $Y, \xi \models \varphi$.

Contrary to logic tradition, in this appendix our structures and substructures are by default partial.

A.1.4. **Substrate and superstrate.** Fix a vocabulary Υ. In the rest of this appendix, by default, terms 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.1.5. **Existential first-order logic.** *Existential first-order logic*, in short EFO logic, of vocabulary Υ is the fragment of first-order logic of vocabulary Υ given by the following restrictions.

- The only propositional connectives are conjunction, disjunction and negation.
- Negation can be applied only to substrate atomic formulas.
- The universal quantifier is forbidden. Only the existential quantifier is used.

Lemma A.2 (EFO monotonicity). *Every EFO formula φ is monotone in every superstrate relation P. In other words, suppose that φ holds in a structure X under some safe assignment ξ of elements of X to its free variables. If you enlarge the interpretation P_X of P 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.**

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

An (EFO) *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 terms, then $\sigma(R)$ is the rule obtained from R by simultaneously replacing every variable v with term $\sigma(v)$; 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 term in R is defined.

An (EFO) *logic program* Π of vocabulary Υ is a finite set of EFO 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

$$H_1, \dots, H_m \leftarrow B$$

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

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

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: $Y \le Z$ 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 $\Gamma_\Pi(Y) \ge Y$. 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 $(a_1, \dots, a_r)$ satisfying the following condition: there exists a rule $P(t_1, \dots, t_r) \leftarrow B$ 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 [11], 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 [3, Theorem 9]. It is easy to check by induction on n that $X_n \le Y$ for every fixed point Y of Γ_Π. It follows that $X_\omega \le Y$ 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.3. 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.4. *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.5. (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.6. *A rule $P(t_1, \dots, t_r) \leftarrow B$ is equivalent to*

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

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

Proposition A.7 (Successive recursion)**.** *Consider logic programs Π_1 and Π_2 such that their head relation symbols are disjoint sets of head relation symbols and such that the head relation symbols of Π_2 do not occur in Π_1. Then $\Pi_2(\Pi_1(X)) = (\Pi_1 \cup \Pi_2)(X)$ for any substrate structure X.*

The conditions of the proposition assure that the program composition $\Pi_2 \circ \Pi_1$ is well defined. The claim is that the composition is equivalent to $\Pi_1 \cup \Pi_2$, so that, in this case, successive recursions have the same effect as the appropriate simultaneous recursion.

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 $Y_2 \le Z$ and $Z \le Y_2$.

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 $Y_2 \le Z$ 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 $Z \le Y_2$ by the definition of Z. □

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] 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.

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

[4] Sabrina De Capitani di Vimercati, Pierangela Samarati and Sushil Jajodia, "Policies, Models, and Languages for Access Control", Springer Lecture Notes in Computer Science 3433 (2005), 225–237.

[5] Herbert Enderton, "Mathematical Introduction to Logic", Elsevier, 2000.

[6] Wilfrid Hodges, "Model Theory", Cambridge University Press, 1993.

[7] 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.

[8] Yuri Matiyasevich, *Hilbert's Tenth Problem,* MIT Press, 1993.

[9] OASIS. Security Assertion Markup Language (SAMcfL). `www.oasis-open.org/committees/security`.

[10] Oxford English Dictionary, 2nd edition, Oxford University Press, 1989.

[11] Alfred Tarski, "A lattice-theoretical fixpoint theorem and its applications", *Pacific Journal of Mathematics* 5:2 (1955), 285–309.